INDEX

LAB OBJECTIVE

Upon successful completion of this Lab the student will be able to:

1. You will be able to know about Object oriented programming.
2. Use Abstract Data Types in the programs.
3. Application of Non recursive functions.
4. OOP principles like Encapsulation Inheritance Polymorphism were frequently used.
5. Trees –B and AVL Trees and their operations were used.
6. Different sorting techniques (Quick sort, Merge sort, Heap sort)were used.
7. Hashing Techniques are implemented.

INTRODUCTION ABOUT LAB

There are 66 systems (Compaq Presario) installed in this Lab. Their configurations are as follows :

Processor	:	AMD Athelon ™ 1.67 GH_z
RAM	:	256 MB
Hard Disk	:	40 GB
Mouse	:	Optical Mouse
Network Interface card	:	Present

Software

- All systems are configured in **DUAL BOOT** mode i.e, Students can boot from Windows XP or Linux as per their lab requirement.

This is very useful for students because they are familiar with different Operating Systems so that they can execute their programs in different programming environments.

- Each student has a separate login for database access

Oracle 9i client version is installed in all systems. On the server, account for each student has been created.

This is very useful because students can save their work (scenarios', pl/sql programs, data related projects ,etc) in their own accounts. Each student work is safe and secure from other students.

- Latest Technologies like **DOT NET** and **J2EE** are installed in some systems. Before submitting their final project, they can start doing mini project from 2nd year onwards.

- **MASM (Macro Assembler)** is installed in all the systems

Students can execute their assembly language programs using MASM. MASM is very useful students because when they execute their programs they can see contents of Processor **Registers** and how **each instruction** is being executed in the **CPU**.

- Rational Rose Software is installed in some systems

Using this software, students can depict UML diagrams of their projects.

- Software's installed : C, C++, JDK1.5, MASM, OFFICE-XP, J2EE and DOT NET, Rational Rose.

- **Systems are provided for students in the 1:1 ratio.**

- **Systems are assigned numbers and same system is allotted for students when they do the lab.**

Guidelines to Students

- Equipment in the lab for the use of student community. Students need to maintain a proper decorum in the computer lab. Students must use the equipment with care. Any damage is caused is punishable.
- Students are required to carry their observation / programs book with completed exercises while entering the lab.
- Students are supposed to occupy the machines allotted to them and are not supposed to talk or make noise in the lab. The allocation is put up on the lab notice board.
- Lab can be used in free time / lunch hours by the students who need to use the systems should take prior permission from the lab in-charge.
- Lab records need to be submitted on or before date of submission.
- Students are not supposed to use floppy disks
- Use of computer network is encouraged.

Algorithm,Flowchart,Programe development

What is algorithm:

An Algorithm is a deterministic procedure that, when followed, yields a definite solution to a problem.

An Algorithm is a design or plan of obtaining a solution to the problem. it is logically process of analyzing a mathematical problem and data step by step so as to make it easier to understand and implement solution to the problem .it is composed of a finite set of steps, each of which may require one or more operation. it may have zero or more inputs and produces one or major outputs. it should terminate after a finate number of operation.

Important features of an algorithm:

Finiteness:

An algorithm terminates after a fixed number of steps.

Definiteness:

Each step of the algorithm is precisely defined.

Effectiveness:

All the operations used in the algorithm can be performed exactly in a fixed duration of time.

Input:

An algorithm has certain precise inputs before the execution of the algorithm begins.

Output:

An algorithm has one or more outputs.

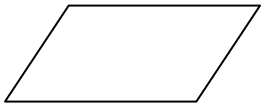
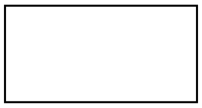
Flowchart:

Flowchart is a graphical representation of an algorithm. it makes use of the basic operations in programming. All symbols are connected among themselves to indicate the flow of information and processing.

A Flowchart is a diagrametic representation of the various steps involved in the solution of a problem.

Flowchart is a symbolic diagram of operation sequence,data flow,control flow and processing logic in information processing.The symbols used are simple and easy to learn.

Symbols used with flowcharts:

Name	Symbol	Purpose
Terminal		Beginning/End of the flowchart
Input/Output		Input/Output data
Process		Mathematical calculations, data transfer and logic operations
Decision Making		Alternate paths
Connector		Transfer to another point in the flowchart
Flow direction		Path of a logic flow in a program

How to run C++ programs

Step 1:

```
#include<iostream.h>
class example
          {

                    public:
                              void ex()
                              {
                              cout<<"This is example program";
                              }
          };
void main()
{
example e;
e.ex();
}
```

Step 2: Submit the file to CC (the C Compiler)(CTRL+F9)
If the program has no errors, the compiled the output is as follows

Output:

```
This is example program
```

List of Lab Exercises

Syllabus Programs (JNTU)

S.No	Programs
1	Write a C++ program to implement the following using an array a) Stack ADT b) Queue ADT
2	Write a C++ program to implement the following using a singly linked list a. Stack ADT b. Queue ADT.
3	Write C++ Program to implement the deque (double ended queue) ADT using a doubly linked list.
4	Write a C++ program to perform the following operations: a) Insert an element into a binary search tree. b) Delete an element from a binary search tree. c) Search for a key element in a binary search tree.
5	Write a C++ program to implement circular queue ADT using an array.
6	Write a C++ program that use non –recursive functions to traverse the given binary tree in a) Preorder b)inorder and c)post order
7	Write C++ programs for the implementation of bfs and dfs for a given graph
8	Write C++ programs for implementing the following sorting methods: a) Quick sort b) Merge Sort c) Heap Sort.
9	Write a C++ program to perform the following operations. a) insertion into a B-tree b) Deletion from a B-tree
10	Write a C++ program to perform the following operations. a) insertion into a AVL-tree b) Deletion from a AVL-tree
11	Write a C++ program to implement Kruskal's algorithm to generate a minimum spanning tree

Additional Programs

1. Write a program with constructor function and another for destruction function.

2. Write a program to explain function overloading with different number of arguments.

3. Write a program to explain function overloading with type, order, and sequence of arguments.

4. Write a program on overloading operator++

5. Write a program for overloading operator++ and operator—using friend functions.

6.Write a program to explain Single inheritance

7.Write a program to explain multiple inheritance

8. Write a program to explain virtual functions.

OOP Concepts:

The object oriented paradigm is built on the foundation laid by the structured programming concepts. The fundamental change in OOP is that a program is designed around the data being operated upon rather upon the operations themselves. Data and its functions are encapsulated into a single entity.OOP facilitates creating reusable code that can eventually save a lot of work. A feature called polymorphism permits to create multiple definitions for operators and functions. Another feature called inheritance permits to derive new classes from old ones. OOP introduces many new ideas and involves a different approach to programming than the procedural programming.

Benefits of object oriented programming:

- Data security is enforced.
- Inheritance saves time.
- User defined data types can be easily constructed.
- Inheritance emphasizes inventions of new data types.
- Large complexity in the software development cn be easily managed.

Basic C++ Knowledge:
C++ began its life in Bell Labs, where Bjarne Stroustrup developed the language in the early 1980s. C++ is a powerful and flexible programming language. Thus, with minor exceptions, C++ is a superset of the C Programming language.
The principal enhancement being the object –oriented concept of a **class.**
A Class is a user defined type that encapsulates many important mechanisms. Classes enable programmers to break an application up into small, manageable pieces, or **objects.**

Basic concepts of Object-oriented programming:

Object:
Objects are the basic run time entities in an object-oriented system.
They may represent a person, a place, a bank account, a table of data or any item that the program has to handle.

class:

The entire set of data and code of an object can be made of a user defined data type with the help of a class in fact Objects are variables of the type class. Once a class has been defined , we can create any number of objects belonging to that class. A class is thus a collection of objects of similar type.
For example: mango, apple, and orange are members of the class fruit.

ex:

fruit mango;
will create an object mango belonging to the class fruit.

Data Abstraction and Encapsulation:

the wrapping up of data and functions in to a single unit is known as encapsulation.Data encapsulation is the most striking feature of a class. The data is not accessable to the outside world and only those functions which are wrapped in the class can access. This insulation of the data from direct access by the program is called data hiding.

Abstraction :

Abstraction refers to the act of representing essential features without including the background details or explanations. since the classes use the concept of data abstraction ,thy are known as abstraction data type(ADT).

Inheritance :

Interitance is the process by which objects of one class acquire the properities of objects of another class.
for example:
The bird 'robin ' is a part of the class 'flying bird' which is again a part of the class 'bird'.
The concept of inheritance provide the idea of reusability.

POLYMORPHISM:

ploymorphism is another important oop concept. Ploymorphism means the ability to take more than one form.an operation may exhibit different instances. The process of making an operator to exhibit different behaviours in different instance is known as operator overloading.

Input/output Statements::

```
Cout<<"example";
Cin>>n;
```

Class definition:

class definition has two components: the class head, the class body.

```
Class vector //the class head
{
// all class members
};
```

<u>Solutions for Additional Programs</u>

Program 1:

Do an example program with constructor function.

Constructors:

It is convenient if an object can initialize itself when it is first created, without need to make a separate call to its member functions. C++ provides a non static member function called as constructor that is automatically called when object is created. After objects are created, their members can be initialized by using constructors.

Important points to be noted while using constructors

- Constructor name must be the same as that of its class name in which it belongs.
- Constructors are invoked automatically when objects are created.
- Constructors cannot specify return types nor return values.
- Constructors cannot be declared static or const.
- Constructors should have public access within the class.

Destructors:

A destructor is a member function that is called automatically when an object is destroyed.
~class name ();
Where the class name is the name of the destructor and is preceded by a tilde (~) symbol.

1.Do an example program with constructor function.And Example program for destruction:
Algorithm:

```
step1:create a maths class
step2:declared within a maths class private int vatible, constructor and a showdata function
step3:initialize a variable 'a' to 100 within the constructor
step4:display 'a' value with the help of showdata function
step4:within the main function create a instance of the maths class to 'm'
step5:call a method of showdata function using 'm'
```

Program:

```
# include<iostream.h>
class maths
{
  int a;
  public:
           maths();
           void showdata();
};
maths :: maths()
{
           cout<<"\n this is a constructor";
           a=100;
}
void maths :: showdata()
{
           cout<<"\n the value of a="<<a;
}

void main ()
{
           maths m;
           m.showdata();
}
```

Output:

```
This is a constructor
The value of a=100
```

Program1:
Example program for destruction:
Algorithm:

```
step1:initialize n=0
step2: create a call class
step3: declared constructor and a destructor function within a call class
step4:display ++n value using constructor
step5: display --n value using destructor
step6: within the main function create two instances of the call class ‘c1’ and ’c2’
```

Program:

```
#include<iostream.h>
int n=0;
class call
{
        public:
                call ()
                        {
                        cout<<”\n constructor”<<++n;
                        }
                ~call ()
                        {
                        cout<<”\n destructor”< --no;
                        }
};

main ()
{
call c1, c2;
}
```

OUTPUT:

```
constructor 1
constructor 2
destructor 1
destructor 0
```

Program :2

Write a program to explain function overloading with different number of arguments

Function Overloading:

C++ enables two or more functions with same name but with different types of arguments or with different number of arguments .This capability to overload a function is called function overloading.

Write a program to explain function overloading with different number of arguments.

Algorithm:

```
step1:create two functions with the same name and same data type but different number of arguments
step2: within the main function initialize integer values x=5,y=10,z=20,a=0,b=0
step3:a)s=call sum(x,y)
        print 's'
      b) s=call sum(x,y,z)
        prin t's'
step4:a)with in the called function of sum(x,y)
        return x+y
        b)with in the called function of sum(x,y,z)
        return x+y+z
```

Program:2

```
#include<iostream.h>
#include<conio.h>
int sum(int, int);
int sum(int, int, int)
void main()
{
        int x=5,y=10,z=20,a=0,b=0;
```

```
            clrscr();
            a=sum(x,y);
            b=sum(x,y,z);
            cout<<" sum of two integers=<<a<<endl;
            cout<<"sum of three integers="<<b<<endl;
            getch();
}

int sum(int x,int y)
            {
            return(x+y);
            }

int sum(int x,int y,int z)
            {
            return(x+y+z);
            }
```

Output:

```
sum of two intezers=15
sum of three integers=35
```

Program 3:

Write a program to explain function overloading with type, order, and sequence of arguments.

Algorithm:

```
step1:create more functions with the same name and different data types and different number of arguments
step2: within the main function initialize different data type varibles
step3:a)sum=call all functions
        print 'sum'
step4:a)write all called functions return 'sum of values'
```

Program:

```
#include<iostream.h>
#include<conio.h>
int sum(int,int);
int sum(int,int,int);
float sum(float,int,float);
double sum(double,float);
double sum(int,float,double);
long double sum(long double,double);
void main()
{
        int a=sum(4,6);
        int b=sum(2,4,6);
        float c=sum(3.5f,7,5.6f);
        double d=sum(7,8,1.2f);
        double e=sum(1,2.2f,8.6);
        long double f=sum(100,80);
        clrscr();
        cout<<"sum(int,int) ="<<a<<endl;
        cout<<" sum(int,int,int) ="<<b<<endl;
        cout<<"sum(float,int,float) ="<<c<<endl;
        cout<<" sum(double,float) ="<<d<<endl;
        cout<<" sum(int,float,double) ="<<e<<endl;\
        cout<<"sum(long double,double) = "<<f;
        getch();
}
int sum(int x,int y)
        {
        return(x+y);
        }
int sum(int x,int y,int z)
        {
        return(x+y+z);
        }
```

```
float sum(float x,int y,float z)
        {
         return(x+y+z);
        }
double sum(double x,float y)
        {
         return(x+y);
        }
double sum(int x,float y,double z)
        {
         return(x+y);
        }
long double sum(long double x,double y)
        {
        return(x+y);
        }
```

Output:

```
sum(int,int) =10
sum(int,int,int) =12
sum(float,int,float) =16.1
sum(double,float) =9
sum(int,float,double) =11.8
sum(long,double,double) =180
```

Program 4:

Write a program on overloading operartor++

Operator Overloading:

We know that operator is a symbol that performs some operations on one or more operands. The ability to overload operators is one of the C++'s most powerful features++ allows us to provide new definitions to some built – in operators. We can give several meaning to an operator, depending upon the types of arguments used. This capability to overload an operator is called operator overloading.

Operators are overloaded by creating operator functions.an operator function consists of a function definition, similar to a normal function except that the function name now becomes the keyword OPERATOR followed by the symbol being overloaded. The general form or syntax of an operator function is

```
Return type class name:: operator <operator symbol>(argument list)
{
//Body of the function
}
```

Where return type is the data type of the value return after the specific operation, operator symbol is the operator being overloaded, that is preceded by the keyword operator, and Operator function is usually a member function or a friend function.

An operator that deals with only one operand is called as an unary operator. Some of the commonly used unary operators are ++,--,!,~ and unary minus.

Overloading increment and decrement operators: The increment operator (++) and the decrement operator (_ _) are unary operators since they act on only one operand. Both the increment and decrement operators have two forms. They are prefix(i.e do the operation and then use the content of the variable.) and postfix (i.e use the content of the variable and then do the operation)notation. The general syntax form or the syntax of the prefix and postfix operator functions are

```
//prefix increment
return type operator++()
{
// Body of the prefix operator
}
//postfix increment
Return operator++(int x)
{
//body of the postfix operator
}
```

4)a)Write a program on overloading operartor++

Algorithm:

```
#include<iostream.h>
#include<conio.h>
class pen
{
        private:
                int count;
        public:
                pen()
                    {
                    count=0;
                    }
                pen(int p)
                        {
                       count=p;
                        }
                void showcount()
                        {
                        cout<<" the value of count is "<<count;
                        }
                pen operator++()
                        {
                        return pen(++count);
                        }
                pen operator++(int)
                        {
                        return pen(++count);
                        }
};
void main()
{
pen p1,p2;
clrscr();
cout<<"\n for object p1:";
++p1;
p1.showcount();
p1++;
p1++;
p1.showcount();
cout<<"\n for object p2:";
++p2;
++p2;
p2.showcount();
p2++;
p2.showcount();
getch();
}
```

Output:

```
for object p1:
the value of count is 1
the value of count is 3
for object p2:
the value of count is 2
the value of count is 3
```

Program: 5
Write a program for overloading operator++ and operator— using friend functions.

```
#include<iostream.h>
#include<conio.h>
class book
{
  private:
          int x,y;
  public:
          book(int a,int b)
                  {
                  x=a;
                  y=b;
                  }
          void showdata()
                  {
                  cout<<” \n value of x :”<<x<<endl;
                  cout<<”\n value of y:”<<y<<endl;
                  }
          friend book operator ++(book &abc);
          friend book operator --(book &abc);
};

book operator ++(book &abc)
{
abc.x++;
abc.y++;
return abc;
}
book operator –(book &abc)
{
abc.x--;
abc.y--;
return abc;
}
void main()
{
book b1(5,25);
clrscr();
b1.showdata();
++b1;
b1.showdata();
--b1;
b1.showdata();
getch();
}
```

Output:

```
the value of x:5
the value of y:25
the value of x:6
the value of y:26
the value of x:5
the value of y:25
```

Inheritance:
 A class have specified data type and if you need another data type similar to the previous one with some additional features,instead of creating a new data type,C++ allows to inherit the members of the existing type and can add some more features to the new class. This property is refered to as inheritance.The old class is called as base class and new class is called as derived class or child class.
The general form or the syntax of specifying a derived class is :

```
Class derivedclassname  :acessspecifies  baseclassname
{

//body of the derived class

}
```

The colon indicates that,the derived class named derivedclassname is derived from the base class named baseclassname.The access specifer of the base class must be either private,protected or public.if no accessspecifier is present is assumed private by default.The body of the ferived class contains member data and member functions of its own.
Single inheritance: A derived class with only one base class is called as single Inheritance.The derived class can access all the members of the base class and can add members of its own.

Program :6

Write a program to explain Single inheritance.

Algorithm:
step1:Take two classess teacher and student
step2:withina teacher class Enter name and number values with the help of getdata() function and display this data using putdata() function.
step3:within the student class Enter and display marks m1,m2,m3 with the help of getdata(),putdata() functions.
step4:within the student class extends teacher class and using data and their functions to the student class

Program:

```
#include<iostream.h>
class teacher
{
        private:
        char name[25];
        long number;
        public:
        void getdata()
        {
        cout<<"\n \t Enter your name:";
        cin>>name;
        cout<<"\n \t Enter your number:";
        cin>>number;
        }
        void putdata()
        {
        cout<<"\n \t name:"<<name;
        cout<<"\n \t number  : "<<number;
        }
};
class student : public teacher
{
        private:
        int m1,m2,m3;
        public:
        void getdata()
        {
        teacher :: getdata();
        cout<<"\n \t Enter marks in three subjects:";
        cin>>m1>>m2>>m3;
        }
        void putdata()
        {
        teacher :: putdata();
        cout<<"\n \t Marks of three subjects:"<<m1<<m2<<m3;
        }

};
void main()
{
        student s1,s2;
        cout<<"\n Enter data for student 1:\n":
        s1.getdata();
        cout<<"\n Enter data for student2 :\n";l
        s2.getdata();
        cout<<"\n the data of student 1:";
        s1.putdata();
        cout<<"\n the data of student 2:";
        s2.putdata();
}
```

Output:

```
Enter data for student1
            Enter your name:James
            Enter your number:1
            Enter 3 subject mark:75 65 85
Enter data for student 2
            Enter your name :Susila
            Enter your number:2
            Enter marks in 3 subjects:65 85 75

The data for student 1
            Name:James
            Number:1
            Marks of three subjects:75 65 85
The data for student 2
            Name :Susila
            Number :2
            Marks of three subjects:65 85 75
```

Multiple Inheritance: Very often situations may arise in which a class has to derive features from more than one class. Multiple inheritance facilitates a class to inherit features from more than one class. The derived class can access two or more classes and can add properties of its own.

Program :7

Write a program to explain multiple inheritances
Algorithm:
step1:Take three classess student,employee,manager
step2:withina student class Enter name and school, college data with the help of getdata() function and display this data using putdata() function.
step3:within the employee class Enter and display company, age data with the help of getdata(),putdata() functions.
step4:within the manager class extends student and employee classess and using data and their functions to the manager class and Enter & display disegnation ,salary of the manager using putdata(),getdata() functions
step5:within the main function create instance of manager class 'm' and call getdata() and putdata() functions.

Program:

```
#include<iostream.h>
#include<conio.h>
Const int MAX=50;
Class student
{
        Private:
        Char name[MAX];
        Char school[MAX];
        Char college[MAX];
        Public:
        Void getdata()
        {
        Cout<<"\n \t Enter name :";
        Cin>>name;
        Cout<<"\t Enter school name:";
        Cin>school;
        Cout<<"\n \t Enter College name:";
        Cin>>college;
        }
        Void putdata()
        {
        Cout<<"\n \t Name  :"<<name;
        Cout<<"\n \t School Name  :"<<school;
        Cout<<"\n \t College Name  :"<<college;
        }
};
Class employee
```

```
{
        Private
        Char company[MAX];

        Int age;
        Public :
        Void getdata()
        {
        Cout<<"\n \t Enter the company name :";
        Cin>>company;
        Cout<<"\n \t Enter age:";
        Cin>>age;
        }
        Void putdata()
        {
        Cout<<"\n \t Company Name :"<<company;
        Cout<<"\t Age:"<<age;
        }
};
Class manager : private employee,private student
{
        Private:
        Char design[MAX];
        Float salary;
        Public :
        Void getdata()
        {
        Student ::getdata();
        Employee ::getdata();
        Cout<<"\n \t Enter your Designation :";
        Cin>>design;
        Cout<<"\n \t Enter salary:";
        Cin>>salary;
        }
        Void putdata()
        {
        Student ::putdata();
        Employee::putdata();
        Cout<<"\n \t Designation :"<<design;
        Cout<<"\n \t Salary :"<<salary;
        }
};
Void main(0
{
        Manager m;
        Clrscr();
        Cout<<"\n Enter data for manager:"<<endl;
        m.gatdata();
        cout<<"\n The data for manager is :"<<endl;
        m.putdata();
}
```

Output:

```
Enter data for manager :
        Enter name : Ram
        Enter name of school: GMSS
        Enter name of college :GCET
        Enter name of company :CMC
        Enter age :24
        Enter designation :Analyst
        Enter salary :10000
The data for manager is
        Student  Name :Ram
        School Name :GMSS
        College Name :GCET
        Company Name :CMC
        Age :24
        Designation :Analyst
        Salary:10000
```

Program :8

Virtual Functions:

Virtual means existing in effect but not in reality. A virtual function is one that does not really exist but nevertheless appears real to some parts of program. Virtual functions provide a way for a program to decide, when it is running, what function to call. Virtual function allows greater flexibility in performing the same kinds of action, on different kind of objects.

While using virtual functions:

- It should be a member function of a class.
- Static member functions cannot be virtual functions.
- A virtual function should be declared in the base class specifications.
- A virtual function may or may not have function body.
- Virtual functions can be overloaded.
- Constructors cannot be declared as virtual functions.
- There can be virtual destructors.

8)Write a program to explain virtual functions.

```
#include<iostream.h>
Class baseclass
{
        Public:
        Void putdata()
        {
        Cout<<"\n this is base class:";
        }
};
Class derivedclass : public baseclass
{
        Public:
        Void putdata()
        {
        Cout<<"\n This is derived class:";
        }
};
Class derivedclass2 : public baseclass
{
        Public :
        Void putdata()
        {
        Cout<<"\n This is derived class2:";
        }
};
Void main()
{
        Derivedclass d1;
        Derivedclass2 d2;
        Baseclass *ptr;
        Ptr->putdata();
        Ptr=&dc2;
        Ptr->putdata();
}
```

Output:

```
This is a base class
This  is a base class
```

SOLUTIONS FOR PROGRAMS (AS PER JNTU SYLLABUS)

STACKS:

A stack is an ordered collection of data items into which new items may be inserted and from which data items may be deleted at one end. Stack are also called Last-In-First-out(LIFO) lists.

Representation of a stack:

```
6    F          →    TOP
5    E
4    D
3    C
2    B
1    A
```

Basic terminology associated with stacks:

Stack Pointer(TOP):Keeps track of the current position the stack.
Overflow:Occurs when we try to insert(push) more information on a stack than it can hold.
Underflow:Occurs when we try to delete(pop) an item off a stack,which is empty.

Basic Operation Associated with Stacks:

1) Insert (Push) an into the stack.
2) Delete (Pop) an item from the stack.

1) a)Algorithm For Inserting an Item into the Stack s:

```
Procudure PUSH(S,SIZE,TOP,ITEM)
S     Array
SIZE Stack size
TOP   Stack Pointer
ITEM value in a cell
Step1:{Check for stack overflow}
If TOP>=SIZE then
        Printf('Stack overflow')
        Return
Step2:{Increment pointer top}
        TOP=TOP+1
Step 3:{Insert ITEM at top of the Stack}
        S[TOP]=ITEM
        Return
```

1) a) Algorithm For Deleting an Item into the Stack

```
function POP(S,TOP)
S     Array
TOP   Stack Pointer
Step1:{Check for stack underflow}
If TOP=0 then
        Printf('Stack underflow')
        Return
Step2:{Return former top element of stack}
        ITEM=(S[TOP]);
 3:{Decrement pointer TOP}
        TOP=TOP-1
        Return
```

1) a)Algorithm For display Items into a Stack S

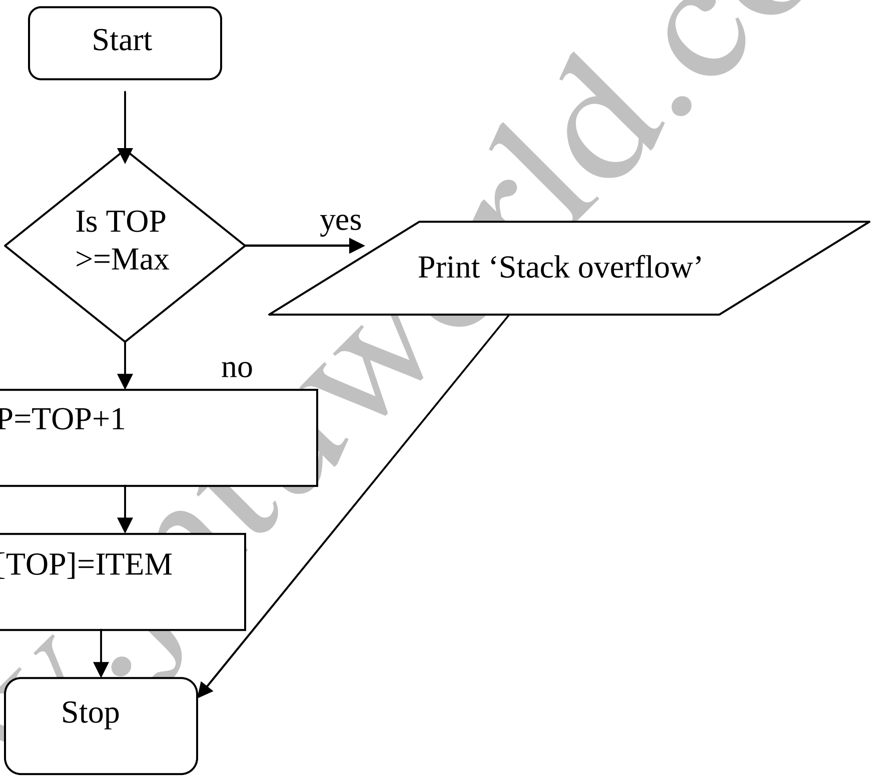
```
function POP(S,TOP)
S     Array
TOP   Stack Pointer
Step1:{Check for stack underflow}
If TOP=0 then
        Printf('stack is empty')
        Return
Step2:{display stack elements until TOP value}
        Print(S[TOP])
        TOP=TOP+1
```

1) a)Algorithm For display top item from the Stack S

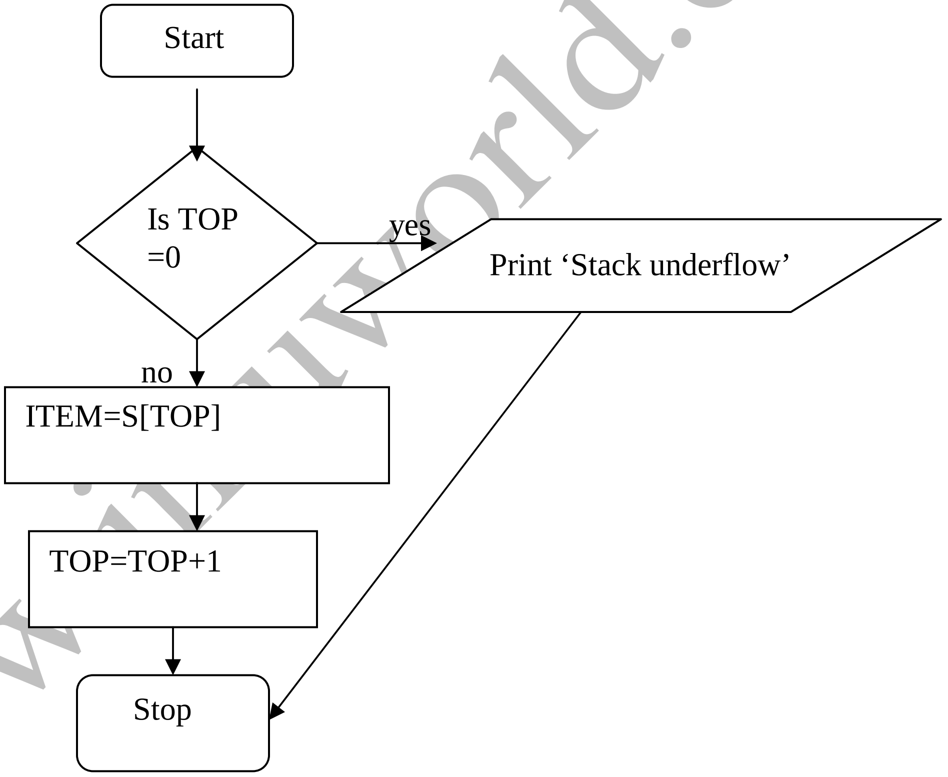
```
function POP(S,TOP)
S     Array
TOP   Stack Pointer
Step1:{Check for stack underflow}
If TOP=0 then
        Printf('stack is empty')
        Return
Step2:{display TOP value into the Stack}
        Print(S[TOP])
```

1)a)Flowchart For Inserting an Item into the Stack s:

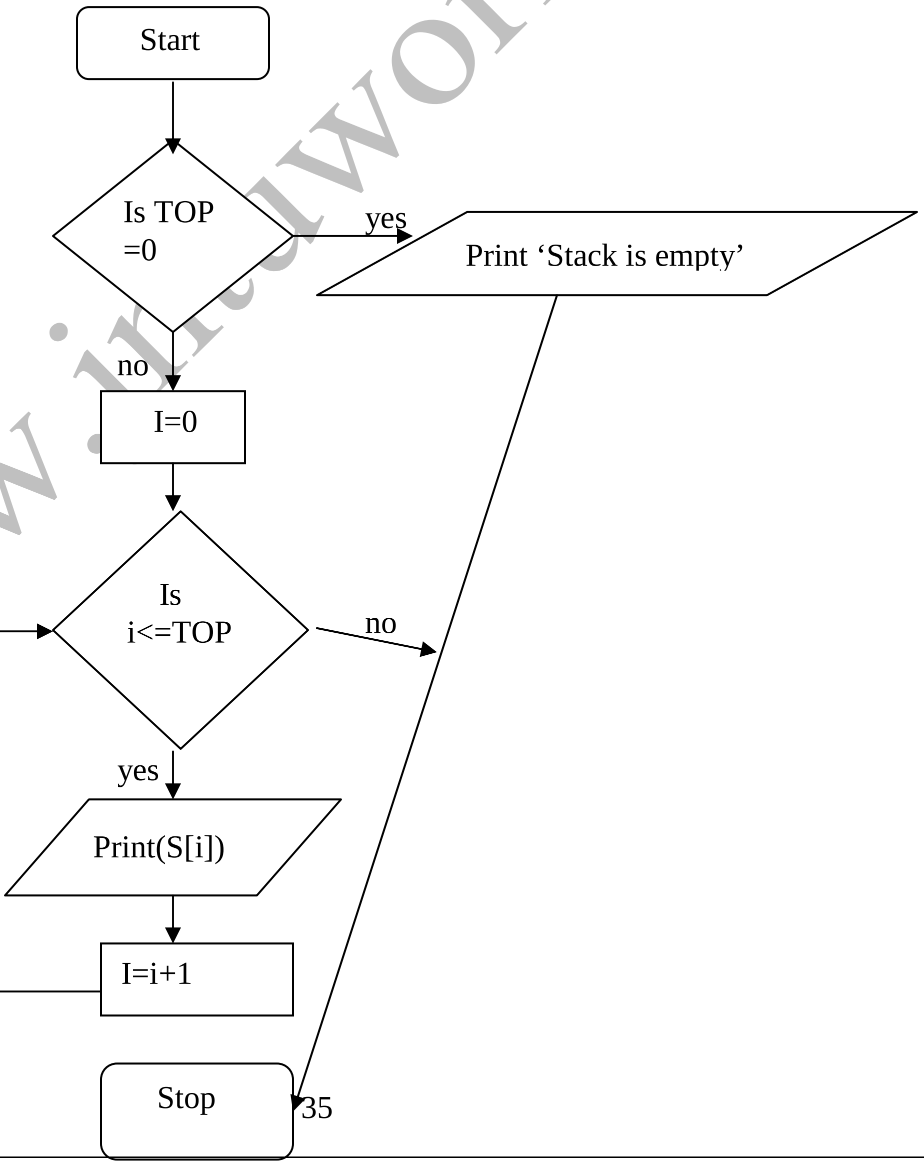
S   Array
SIZE Stack size
TOP   Stack Pointer
ITEM value in a cell



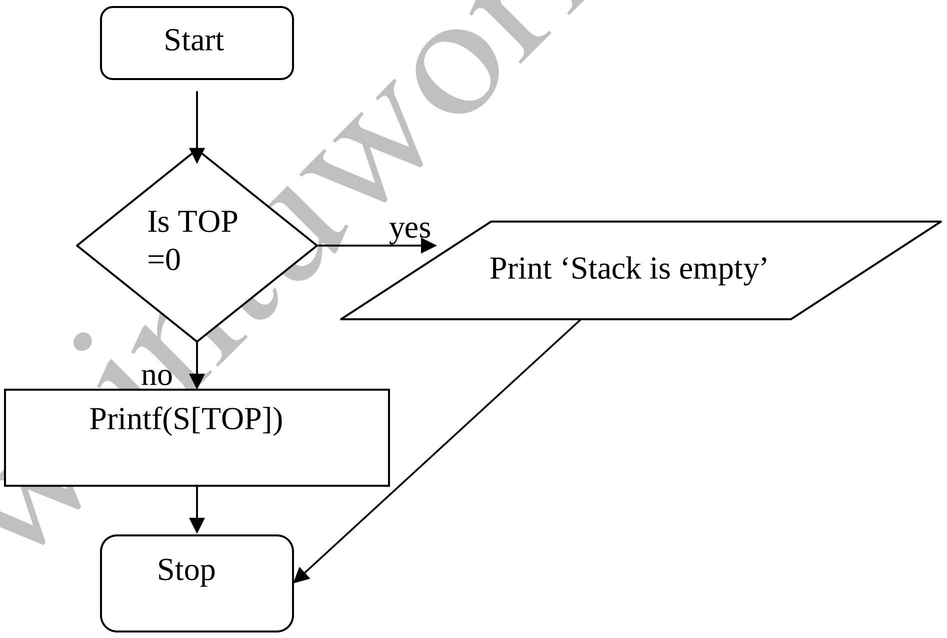
1) a) Flowchart For Deleting an Item into the Stack
function POP(S,TOP)
S   Array
TOP   Stack Pointer



1) a)Flowchart For display top item from the Stack S
function POP(S,TOP)
S   Array
TOP   Stack Pointer



1) a)Flowchart For display top item from the Stack S
function POP(S,TOP)
S   Array
TOP   Stack Pointer



Program 1)a):stack ADT using arrays.

```
#define size 10
#include<iostream.h>
#include<conio.h>
template<class t>
class stack
{
   private:
   t s[size];
   int top;
   public:
   void init();
   isfull();
   isempty();
   void first();
   operation();
   void push(t x);
   t pop();
   void display();
};
template<class t>
void stack<t>::init()
```

```
{
   top=-1;
}
template<class t>

void stack<t>::push(t x)
{
   if(top==size-1)
   cout<<"\n stack is full, insertion is not possible";
   else
   {
   top++;
   s[top]=x;
   cout<<"\n insertion";
   }
}
template<class t>
t stack<t>::pop()
{
   t x;
   if(top==-1)
   return(-1);
   else
   {
   x=s[top];
   top--;
   return(x);
   }
}



template<class t>
void stack<t>::display()
{
   int i;
   if(top==-1)
   cout<<"\nstack is empty";
   else
   {
   cout<<"\nstack contains as follows\n";
   for(i=top;i>=0;i--)
   cout<<s[i]<<" ";
   }
}
template<class t>
void stack<t>::first()
{
   int i;
   if(top==-1)
   cout<<"\nstack is empty";
```

```
    else
    {
    cout<<"\ntop of the stack element is";
    cout<<s[top];
    }
}

void menu1()
{
    cout<<"\n\t 1.integer stack";
    cout<<"\n\t 2.float stack";
    cout<<"\n\t 3.char stack";
    cout<<"\n\t 4.exit";
}
void menu2()
{
    cout<<"\n \t 1.insertion";
    cout<<"\n \t 2.deletion";
    cout<<"\n \t 3.display stack contents";
    cout<<"\n \t 4.top of the stack element is";
    cout<<"\n \t 5.exit";
}
template<class t>
void operation(stack<t>a)
    {
```

```
a.init();
int ch;
menu2();
cout<<"\nEnter your choice:";
cin>>ch;
while(ch<5)
{

switch(ch)
{
        int x;
        case 1:
        {
        cout<<"\nEnter the element to be inserted:";
        cin>>x;
        a.push(x);
        break;
        }
        case 2:
        {
        x=a.pop();
        if(x==-1)
        cout<<"\nstack is empty,deletion is not possible";
        else
        cout<<"deleted element is"<<x<<"\n";
```

```
                break;
                }
                case 3:
                {
                a.display();
                break;
                }
                case 4:

        {
                a.first();
                break;
                }
          }
          menu2();
          cout<<"\nEnter your choice";
          cin>>ch;
          }
        }
        template<class t>
        int stack<t>::isfull()
        {
          if(top==size-1)
          return(1);
          else
```

```
   return(0);
}
template<class t>
int stack<t>::isempty()
{
   if(top==-1)
   return(1);
   else
   return(0);

}
main()
{
   clrscr();
   int ch;
   menu1();
   cout<<"\nEnter your choice";
   cin>>ch;
   while(ch<4)
   {
   switch(ch)
   {
   case 1:
   {
   stack<int>a;
```

```
    operation(a);
    break;
    }
    case 2:
    {
    stack<float>a;
    operation(a);
    break;
    }
    case 3:

    {
    stack<char>a;
    operation(a);
    break;
    }
    }
    menu1();
    cout<<"\nEnter your choice";
    cin>>ch;
    }
    getch();
    return(0);
}
```

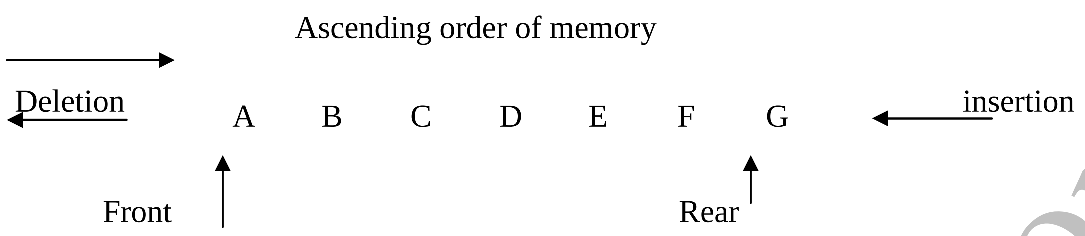
Output:

```
            1.integer stack
            2.float stack
            3.char stack.
            4.exit.
Enter yourchoice:1
            1.insertion
            2.deletion
            3.display stack contents
            4.display top most element
            5.exit
Enter your choice:1
Enter the element to be inserted:10
element is inserted
            1.insertion
            2.deletion
            3.display stack contents
            4.display top most element
            5.exit
Enter your choice:1
Enter the element to be inserted:20
element is inserted
            1.insertion
            2.deletion
            3.display stack contents
            4.display top most element
            5.exit

Enter your choice:1
Enter the element to be inserted:30
element is inserted
            1.insertion
            2.deletion
            3.display stack contents
            4.display top most element
            5.exit
Enter your choice:4
top of the stack element is:30
            1.insertion
            2.deletion
            3.display stack contents
            4.display top most element
            5.exit
Enter your choice:3
stack contains as follows:
30 20 10
            1.insertion
            2.deletion
```

```
          3.display stack contents
          4.display top most element
          5.exit

Enter your choice:2
deleted element is:30
          1.insertion
          2.deletion
          3.display stack contents
          4.display top most element
          5.exit
Enter your choice:5
          1.integer stack
          2.float stack
          3.char stack.
          4.exit.
Enter your choice:4
```

QUEUE:

Queue is an ordered collection of data such that the data is inserted at one end and deleted from other end.It is a collection of items to be processed on a First-In-First-Out(FIFO) or First Come First Served(FCFS) basics.



Basic Operation Associated on Queues:
1) Insert an item into the Queue.
2) Delete an item into the Queue.

1) b)Algorithm For Inserting an Item into a Queue Q:

```
Procudure INSERT(Q,SIZE,F,R,ITEM)
Q     Array
SIZE Queue size
F front Pointer
R rear pointer
ITEM information to be inserted at the rear of queue.
Step1:{Check for Queue overflow}
If  R>=SIZE then
        Printf('Queue overflow')
        Return
Step2:{Increment rear pointer}
        R=R+1
Step 3:{Insert new element at rear end of queue}
        Q[R]=ITEM
Step 4:{If initially,the queue is empty,adjust the fron pointer}
        If F=0,then F=1
```

1) b) Algorithm For deleting an Item from a Queue Q:

```
function DELETE(Q,F,R)
Q     Array
F front Pointer
R rear pointer
ITEM information to be inserted at the rear of queue.

Step1:{Check for Queue underflow}
If F=0 then
          Printf('Queue underflow')
          Return
Step2:{Delete the queue element at front end and store it into item}
          ITEM=Q[F]
 3:{If queue is empty after deletion,set front and rear pointers to 0}
          If F=R then
          F=0
          R=0
     {Otherwise increment front pointer}
          Else
          F=F+1
          Return(ITEM)
```

1) b)Algorithm For display Items into the Queue Q

```
function Dispaly(Q)
Q     Array
Step1: {Check queue values}
          If  F<0
                    Print('Queue is empty')
Step2:{display Queue values}
          For I value F to R
          Print(Q[I])
          I=I+1
```

1) b)Algorithm For display Front Item into the Queue Q

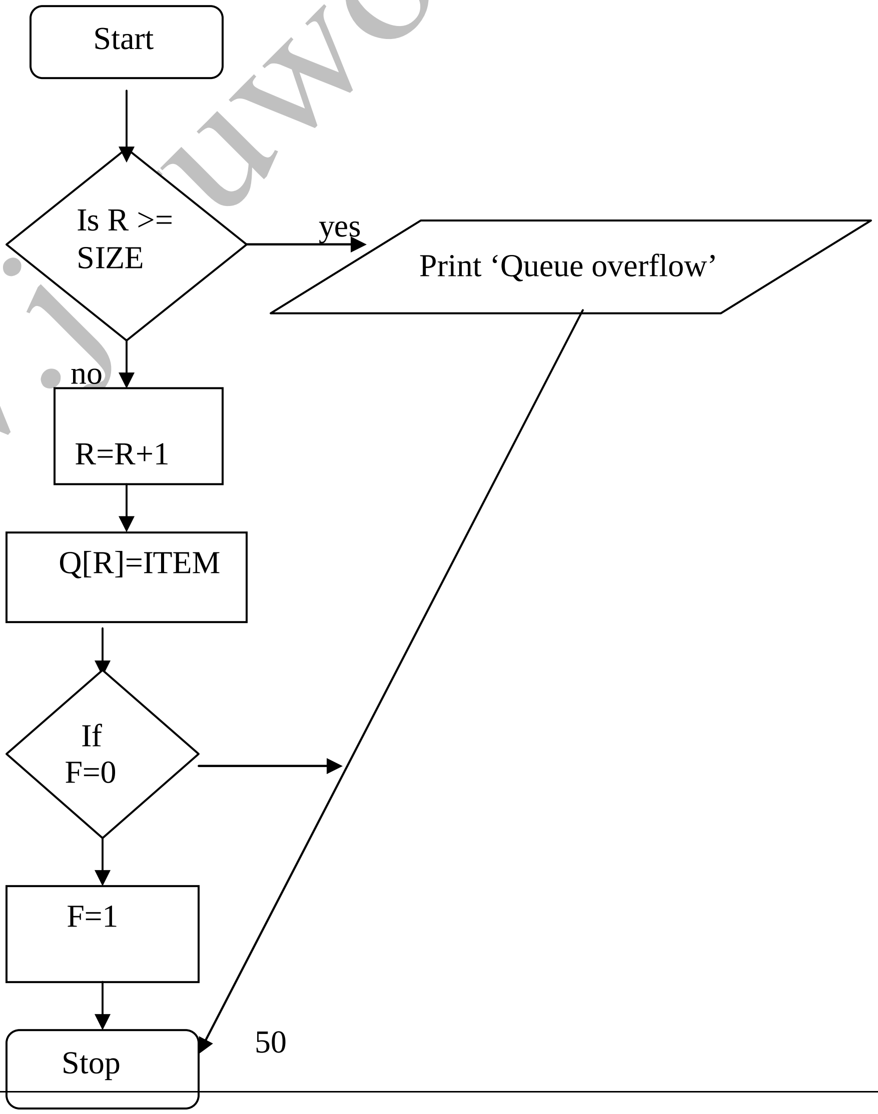
```
function Dispaly(Q,F)
Q     Array
Step1: {Check queue values}
          If  F<0
                    Print('Queue is empty')
Step2:{display Front item of the Queue}
          Print(Q[F])
```

1) b)Algorithm For display Rear Item into the Queue Q

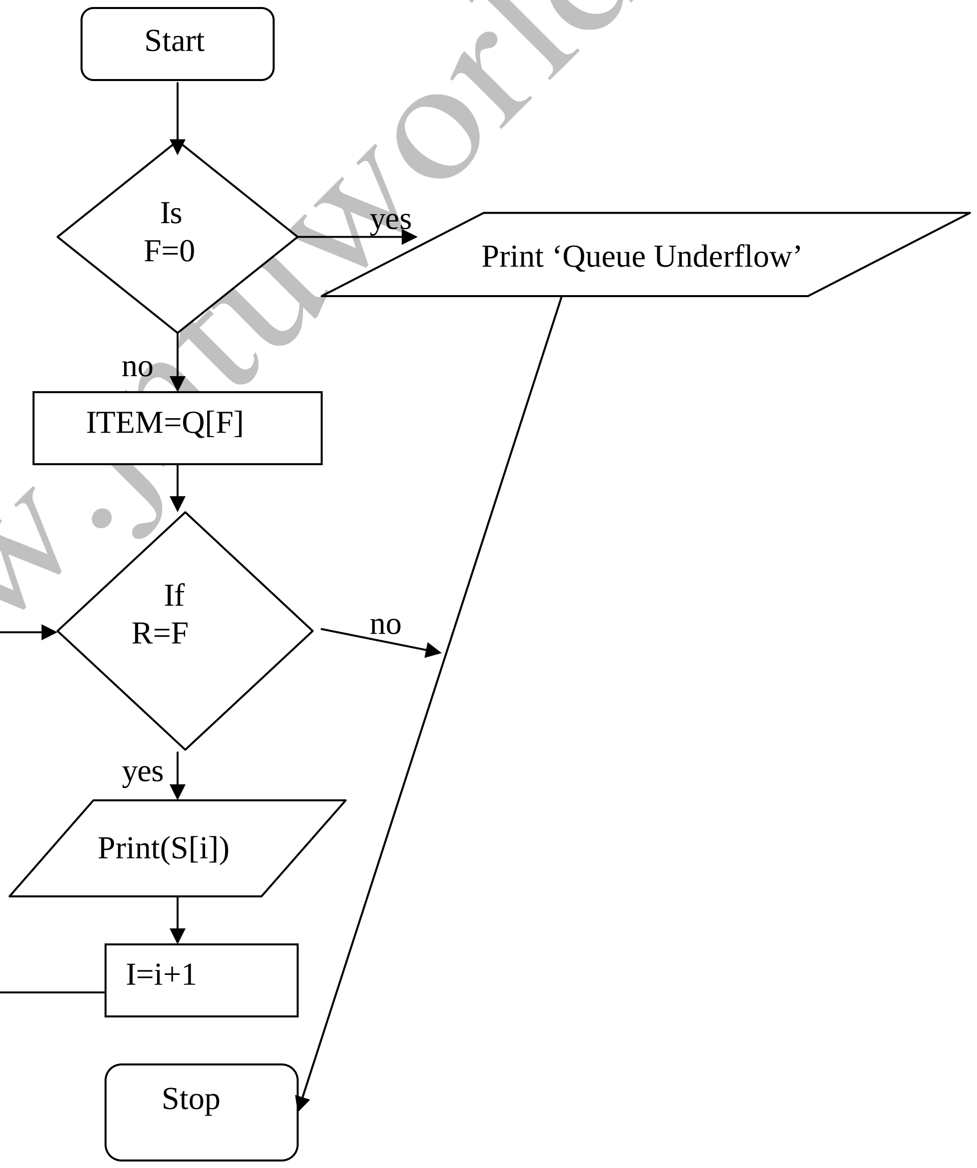
```
function Dispaly(Q,R)
Q     Array
Step1: {Check queue values}
          If  R<0
                    Print('Queue is empty')
Step2:{display Front item of the Queue}
          Print(Q[R])
```

1) b)Flowchart For Inserting an Item into a Queue Q:

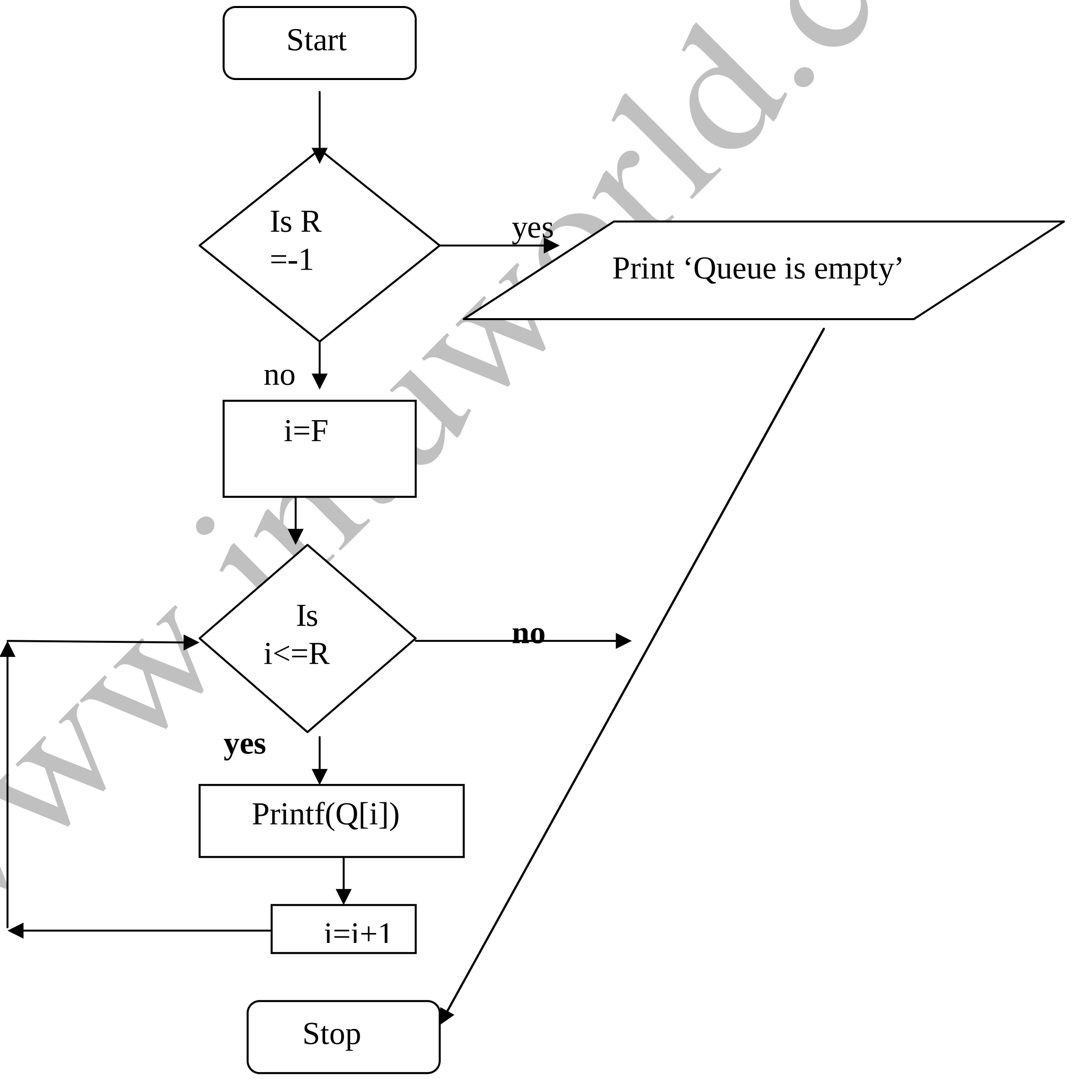
Procudure INSERT(Q,SIZE,F,R,ITEM)
Q   Array
SIZE Queue size
F front Pointer
R rear pointer
ITEM information to be inserted at the rear of queue



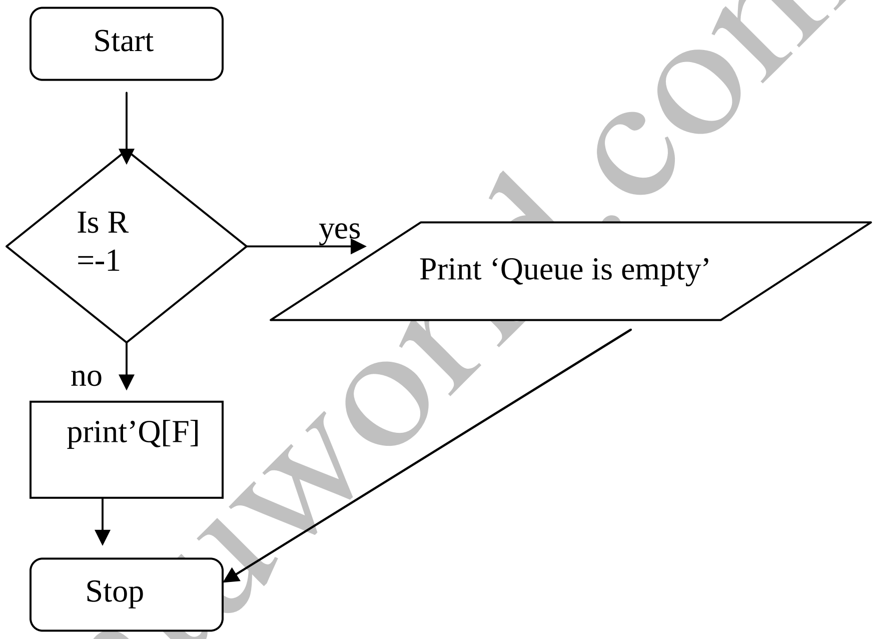
1) b) Flowchart For deleting an Item from a Queue Q:
function DELETE(Q,F,R)
Q   Array
F front Pointer
R rear pointer
ITEM information to be inserted at the rear of queue



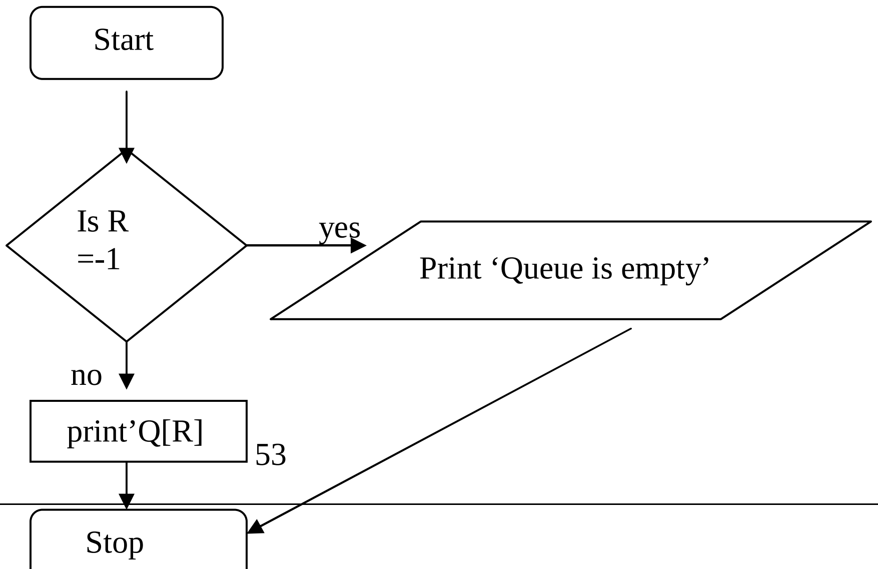
1) b)Algorithm For display Items into the Queue Q
function Dispaly(Q)
Q   Array



1) b)Algorithm For display Front Item into the Queue Q
function FRONT(Q,F)
Q   Array



1) b)Algorithm For display Rear Item into the Queue Q
function REAR(Q,F)
Q   Array



Program 1 :
(b) Queue ADT using Arrays.

```
#include<iostream.h>

#include<conio.h>

#define size 10

template<class T>

class queue

{

   T q[size];    int f,r;

   public:

   int isfull();  T last();    T first();

   void  display();

   void init();

   int isempty();

   void insert(T x);

   T del();

};

template<class T>

void queue<T>::init()

{

   f=0;     r=0;

}
```

```
template<class T>
void queue<T>::insert(T x)
{
   if(r>=size)

cout<<"\n insertion is not possible";
   else{
   r++;
   q[r]=x;
   cout<<"\n element is inserted";
   f=1;
   }
}
template<class T>
T queue<T>::del()
{
   T x;
   if(f==0)
   return(0);
   else
   if(f==r)
   {
   x=q[f];
   f=0;
   r=0;
   return(x);
   }
   else
   {
   x=q[f];

f++;
   return(x);
   }
}
template<class T>
int queue<T>::isfull()
{
   if(r==size)
   return(1);
   else
   return(0);
}
template<class T>
int queue<T>::isempty()
{
   if(r==0)
   return(1);
   else
   return(0);
```

```
}
template<class T>
T queue<T>::first()
{
   return(q[f]);
}


template<class T>
T queue<T>::last()
{
   return(q[r]);
}
template<class T>
void queue<T>::display()
{
   if(isempty()==1)
   cout<<"\n queue is empty";
   else
   {
   cout<<"\n queue contents is";
   for(int i=f;i<=r;i++)
   cout<<q[i];
   cout<<"\n";
   }
}
```

```
void menu1()
{
   cout<<"\n 1.integer queue";
   cout<<"\n 2.floating stack";
   cout<<"\n 3.char queue";
   cout<<"\n 4.exit";
}

void menu2()
{
   cout<<"\n 1.insertion";
   cout<<"\n 2.deletion";
   cout<<"\n 3.display queue";
   cout<<"\n 4.display first element";
   cout<<"\n 5.display last element";
   cout<<"\n 6.exit";
}
template<class T>
void operation(queue<T>a)
{
   int ch;
   int x;
   menu2();
   cout<<"\nEnter your choice";
   cin>>ch;
   while(ch<6)
   {
   switch(ch)
   {
   case 1:
   {
   cout<<"\n Enter the element to be inserted";
   cin>>x;

a.insert(x);
   break;
   }
   case 2:
   {
   x=a.del();
   if(x==-1)
   cout<<"\n queue is empty deletion is not possible";
   else
   cout<<"\n deleted element is"<<x;
   break;
   }
   case 3:
   {
   a.display();
   break;
```

```
    }
    case 4:
    {
    x=a.first();
    cout<<"\n first element is"<<x;
    break;
    }
    case 5:
    {

x=a.last();
    cout<<"\n the last element is"<<x;
    break;
    }
    }
    menu2();
    cout<<"\nEnter your choice";
    cin>>ch;
    }
}
main()
{
    int ch;
    menu1();
    cout<<"\n Enter your choice";
```

```
cin>>ch;
while(ch<4)
{
switch(ch)
{
case 1:
{
queue<int>a;
operation(a);
break;

}
case 2:
{
queue<float>a;
operation(a);
break;
}
case 3:
{
queue<char>a;
operation(a);
break;
}
}
```

```
            menu1();

            cout<<"Enter your choice";

            cin>>ch;

            }

            getch();

            return(0);


         }
```

Output:

```
        1.integer queue
        2.float queue
        3.char queue.
        4.exit.
Enter yourchoice:1
        1.insertion
        2.deletion
        3.display queue contents
        4.display front element
        5.display rear element
        6.exit
Enter your choice:1
Enter the element to be inserted:10
element is inserted
        1.insertion
        2.deletion
        3.display queue contents
        4.display front element
        5.display rear element
        6.exit

Enter your choice:1
Enter the element to be inserted:20
element is inserted
```

```
            1.insertion
            2.deletion
            3.display queue contents
            4.display front element
            5.display rear element
            6.exit

Enter your choice:1
Enter the element to be inserted:30
element is inserted
            1.insertion
            2.deletion
            4.display front element
            5.display rear element
            6.exit
Enter your choice:4
front of the queue is:10
            1.insertion
            2.deletion
            3.display queue contents
            4.display front element
            5.display rear element
            6.exit



Enter your choice:3
queue contains as follows:
10 20 30
            1.insertion
            2.deletion
            3.display queue contents
            4.display front element
            5.display rear element
            6.exit
Enter your choice:5
rear of the queue is:30
            1.insertion
            2.deletion
            3.display queue contents
            4.display front element
            5.display rear element
            6.exit

deleted element is:10
            1.insertion
            2.deletion
            3.display queue contents
            4.display front element
            5.display rear element
```

```
        6.exit
Enter your choice:6
        1.integer queue
        2.float queue
        3.char queue.
        4.exit.

Enter your choice:4
```

LINKED LISTS:

Lisst is a linear collection of data items/nodes explicitly ordered by a field,which indicates the successor node in the list and it is a flexible structure.Items can be inserted and deleted dynamically and easily and it facilitates dynamic allocation of nodes.

Node consists of two fields INFO and LINK

```
                INFO        LINK
                      NODE
```

INFO field contains the information about the being stored in the list.
LINK field contains the address/pointer to the next item in the list.
Node structure declaration

```
struct node
{
        int info;
        struct node *link;
};
struct node *first;
```

empty list
first=NULL;

function to allocate a node and to initialize it

```
                INFO          LINK
                X              ^
                      NEW
```

```
Struct node *new;
{
        struct node *new;
        new=(struct node *)malloc(sizeof(struct node));
        new->info=x;
        new->link=NULL;
        return(new);
}
```

SINGLE LINKED LIST:

Linked list consists of an ordered set of nodes/

$$\begin{tikzcd}
\text{INFO} & \text{LINK} \arrow[r] & \text{--} & \text{INFO} & \text{LINK} \arrow[r] & \text{---} & \text{INFO} & \text{LINK} \\
\text{FIRST} & & & & & & &
\end{tikzcd}$$

FIRST   Address/pointer which gives the location of the first node of the list.
/(NULL) signals the end of the list.
->(Arrow) indicates the successor node.
Example:

$$\begin{tikzcd}
\text{A} & 2010 \arrow[r] & \text{B} & 2002 \arrow[r] & \text{C} & 2012 \arrow[r] & \text{D} & 2006 \arrow[r] & \text{E} & \wedge \\
\text{First=2000} \arrow[u] & & 2010 \arrow[u] & & 2002 \arrow[u] & & 2012 \arrow[u] & & 2006 \arrow[u] &
\end{tikzcd}$$

ADDRESS	INFO	LINK
2000	A	2010
2001		
2002	C	2012
2003		
2004		
2005		
2006	E	
2007		
2008		
2009	B	2002
2010		
2011		
2012	D	2006
2013		

Program 2:
a)Stack ADT using Linked Lists.
2) a)Algorithm For Inserting an Item into the Stack s:

```
Procudure PUSH(s)
s    pointer
x   value in a cell

Step1:{create a structure}
        struct node
        {
                int data;
                struct node *link;
        };
        struct node *top;
Step2:if top>=size then
        print‘stack overflow’
        else
        node *p;
        p=new node;
        p->data=x;
        p->link=top;
        top=p;
```

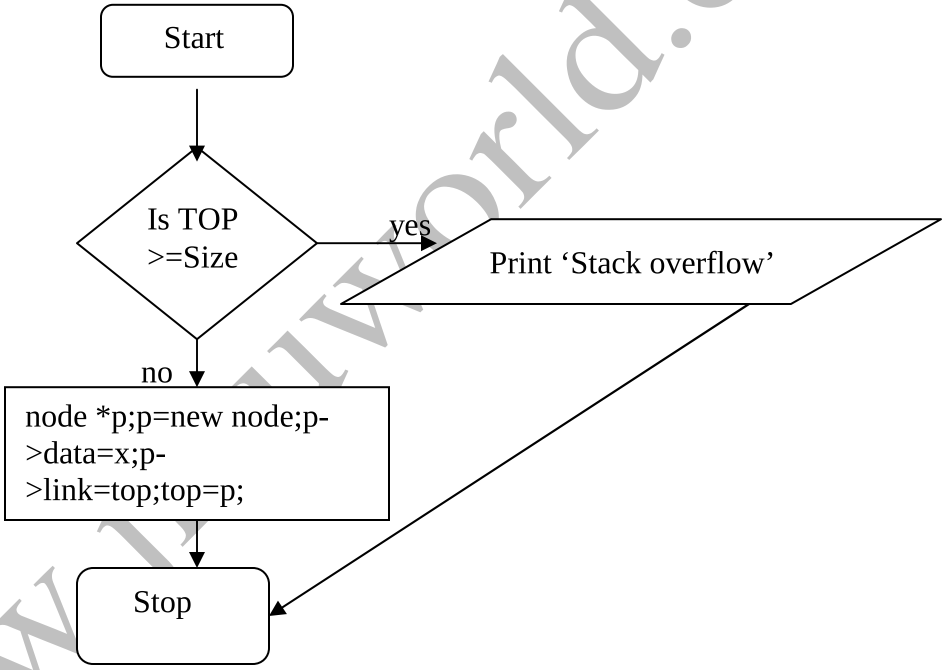
2) a) Algorithm For Deleting an Item into the Stack

```
function POP()

Step1:int x;
        node *p;
        if top==0 then
        print(‘stack under flow’
        else
        top=top->link;
        x=p->data;
        delete’p’;
        Return(x);
```

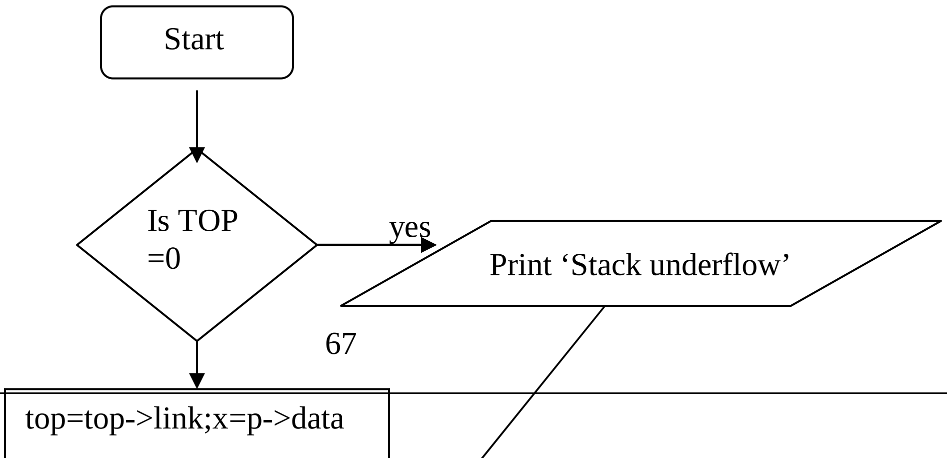
2) a)Algorithm For display Items into a Stack s

```
function DISPLAY()
Step1:{Check for stack underflow}
        If top=0 then
                Printf('stack is empty')
                Return
        else
                temp=top
                repeate loop until temp is equal to zero
                print'temp->data
                top=top->link
```

2)a)Flowchart For Inserting an Item into the Stack s:



2) a) Flowchart For Deleting an Item into the Stack



no

Program 2)a):

```
#include<iostream.h>
#include<conio.h>
class linklist
{
    private:
    struct node
    {
    int data;
    node* link;
    }*top;
    public:
    void Init();
    void display();
    int isempty();
    ~linklist();
    void push(int);
    int pop();
```

```
   int Topmost();
};
void linklist::Init()
{
   top=0;
}
int linklist::isempty()
{

if(top==0)
   return(1);
   else
   return(0);
}
int linklist::Topmost()
{
   if(top==0)
   return(-1);
   else
   return(top->data);
}
void linklist::push(int x)
{
   node *p;
   p=new node;
   p->data=x;
   p->link=top;
   top=p;
}
int linklist::pop()
{
   int x;
   if(isempty()==1)
   return(-1);

else
   {
   x=top->data;
   delete top;
   top=top->link;
   return(x);
   }
}

linklist::~linklist()
{
   node *p;
   while(top!=0)
   {
   p=top;
   top=top->link;
   delete p;
   }
}

void linklist::display()
{
   node *temp;
   clrscr();
   if(top==0)

cout<<"stack is empty\n";
   else
   {
   temp=top;
   while(temp!=0)
   {
   cout<<temp->data<<"\t"<<temp->link<<"\n";
   temp=temp->link;
   }
   }
}
void menu2()
{
   cout<<"\n \t 1.insertion";
   cout<<"\n \t 2.deletion";

   cout<<"\n \t 3.display stack contents";

   cout<<"\n \t 4.top of the element";

   cout<<"\n \t 5.exit";

}

void main()

{

   int ch;

   linklist l;

   menu2();

   cout<<"\nEnter your choice";


cin>>ch;

   while(ch<5)

   {

   switch(ch)

   {

            int x;

            case 1:

            {

            cout<<"\nEnter element to be inserted:";

            cin>>x;

            l.push(x);

            cout<<"\nelement is inserted";

            break;
            }
            case 2:
            {
            x=l.pop();
            if(x==-1)
            cout<<"\nstack is empty,deletion is not possible";
            else
            cout<<"deleted element is"<<x<<"\n";
            break;
            }
            case 3:
            {

l.display();
            break;
            }
            case 4:
            {
            int c;
            c=l.Topmost();
            cout<<"\ntop of the stack is:"<<c;
            break;
            }
   }
   menu2();
```

```
    cout<<"\nEnter your choice";
    cin>>ch;
    }
    getch();
}
```

Output:

```
            1.insertion
            2.deletion
            3.display stack contents
            4.top of the stack
            5.exit
Enter your choice:1
Enter the element to be inserted:10
element is inserted
            1.insertion
            2.deletion
            3.display stack contents
            4.top of the stack
            5.exit
Enter your choice:1
Enter the element to be inserted:20
element is inserted
            1.insertion
            2.deletion
            3.display stack contents
            4.top of the stack
            5.exit

Enter your choice:1
Enter the element to be inserted:30
element is inserted
            1.insertion
            2.deletion
            3.display stack contents
            4.top of the stack
            5.exit
Enter your choice:3
stack contains as follows:
30 1004
20  1002
10   1000
            1.insertion
            2.deletion
            3.display stack contents
            4.top of the stack
            5.exit
Enter your choice:4
top of the element is:30
            1.insertion
            2.deletion
            3.display stack contents
            4.top of the element
            5.exit
```

```
Enter your choice:2
deleted element is:30
        1.insertion
        2.deletion
        3.display stack contents
        4.top of the element
        5.exit
Enter your choice:3
stack contains as follows:
20 1002
10 1000
        1.insertion
        2.deletion
        3.display stack contents
        4.top of the element
        5.exit

Enter your choice:5
```

2)b) Queue ADT using Linked Lists.

2) b)Algorithm For Inserting an Item into a Queue Q:

```
Procudure INSERT(ITEM)
ITEM information to be inserted at the rear of queue.
step1:struct node{
        t data;
        node *link
        }*f,*r;
Step2:{Check for Queue overflow}
If  R>=SIZE then
        Printf('Queue overflow')
        Return
Step3:node *p;
        p=new node;
        p->data=x;
        p->link=o;
Step 4:if(f=r=0) then
        f=r=p;
        else
        r->link=p;
        r=p;
```

2) b) Algorithm For deleting an Item from a Queue Q:

```
function DELETE()

Step1:{Check for Queue underflow}
If F=R=0 then
        Printf('Queue underflow')
        Return
Step2:{Delete the queue element at front end and store it into item}
        node *p;
        int x;
 3:{If queue is empty after deletion,set front and rear pointers to 0}
        If F=R then
        p=f
        f=r=0;
        x=p->data;
        delete p;
        Return(x);
        else
        p=f;
        f=f->data;
        delete p;
        return(x);
```

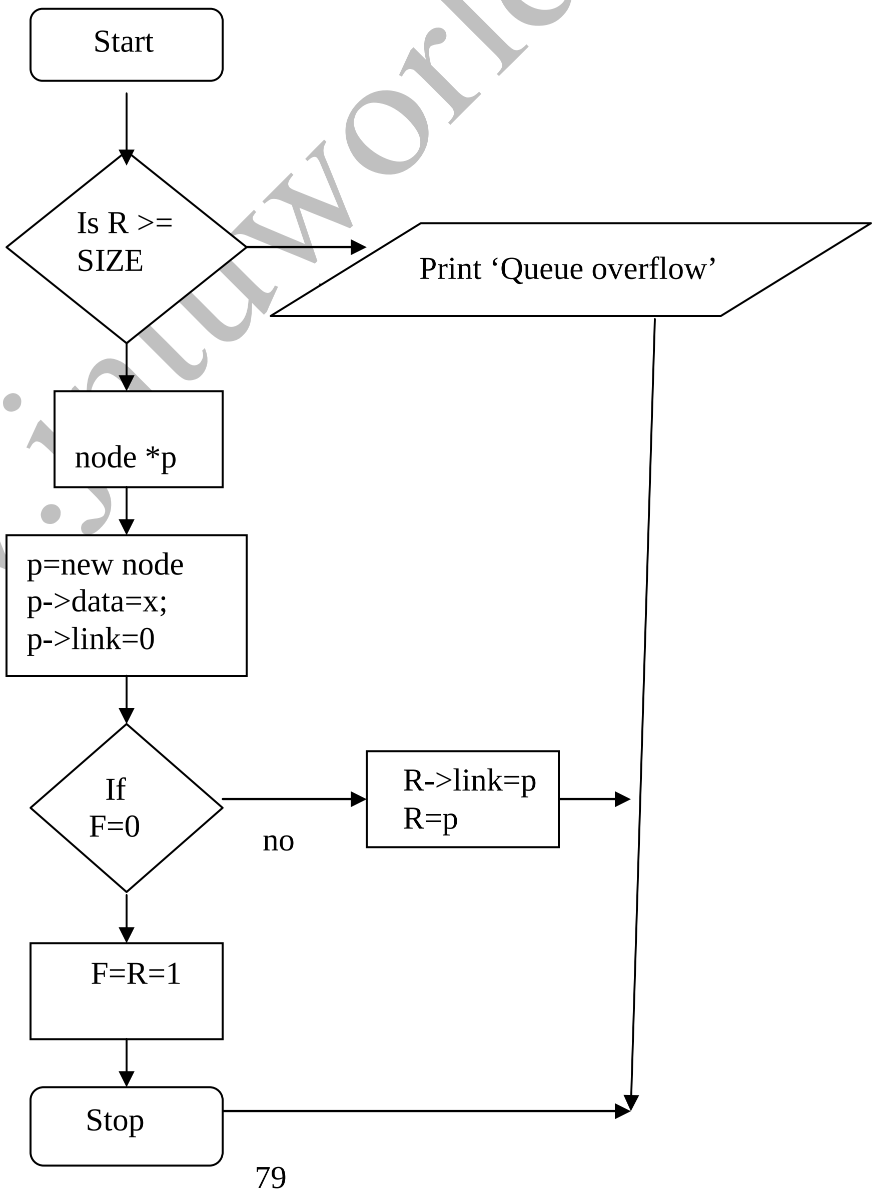
2) b)Algorithm For display Items into the Queue Q

```
function Dispaly(Q)
Q     Array
Step1: {Check queue values}
          If  F<0
                    Print('Queue is empty')
Step2:{display Queue values}
          temp=f;
          repeate until temp not equal to zero
          Print(temp->data)
          temp=temp->link;
```

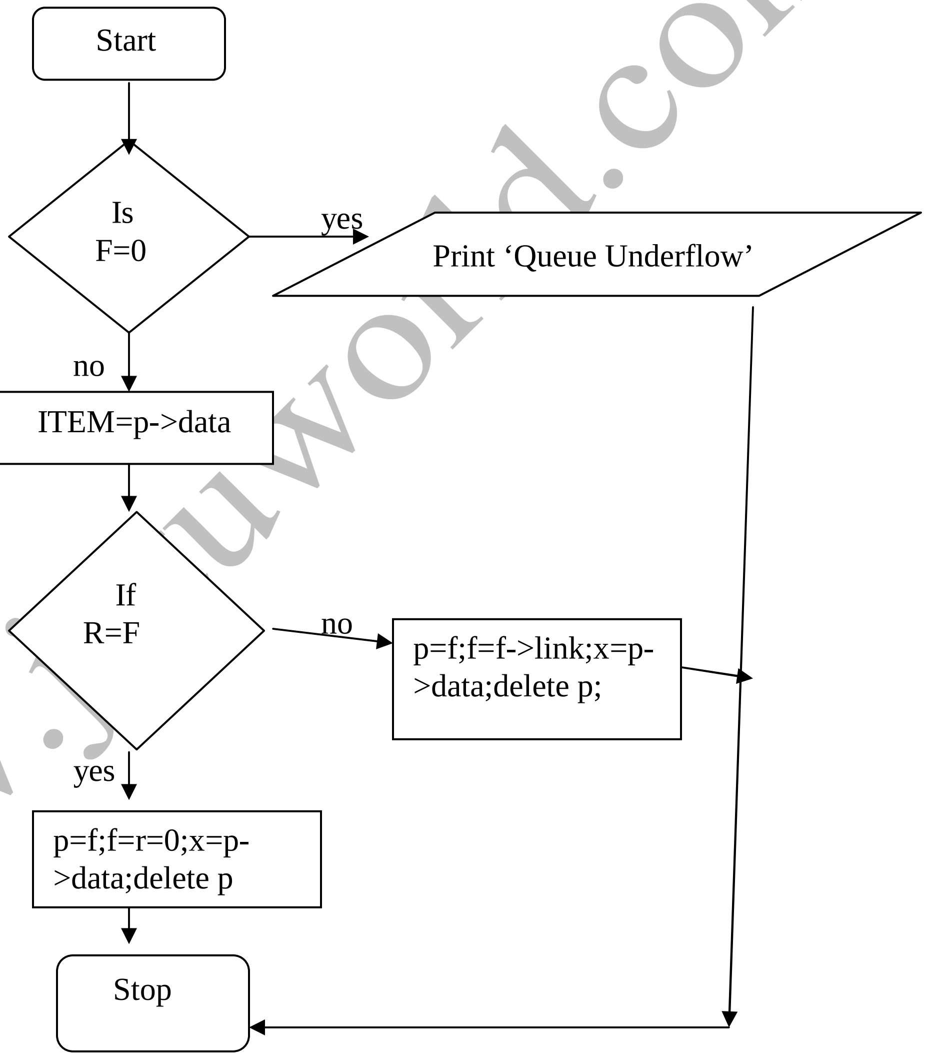
2) b)Flowchart For Inserting an Item into a Queue Q:

Procudure INSERT(ITEM)

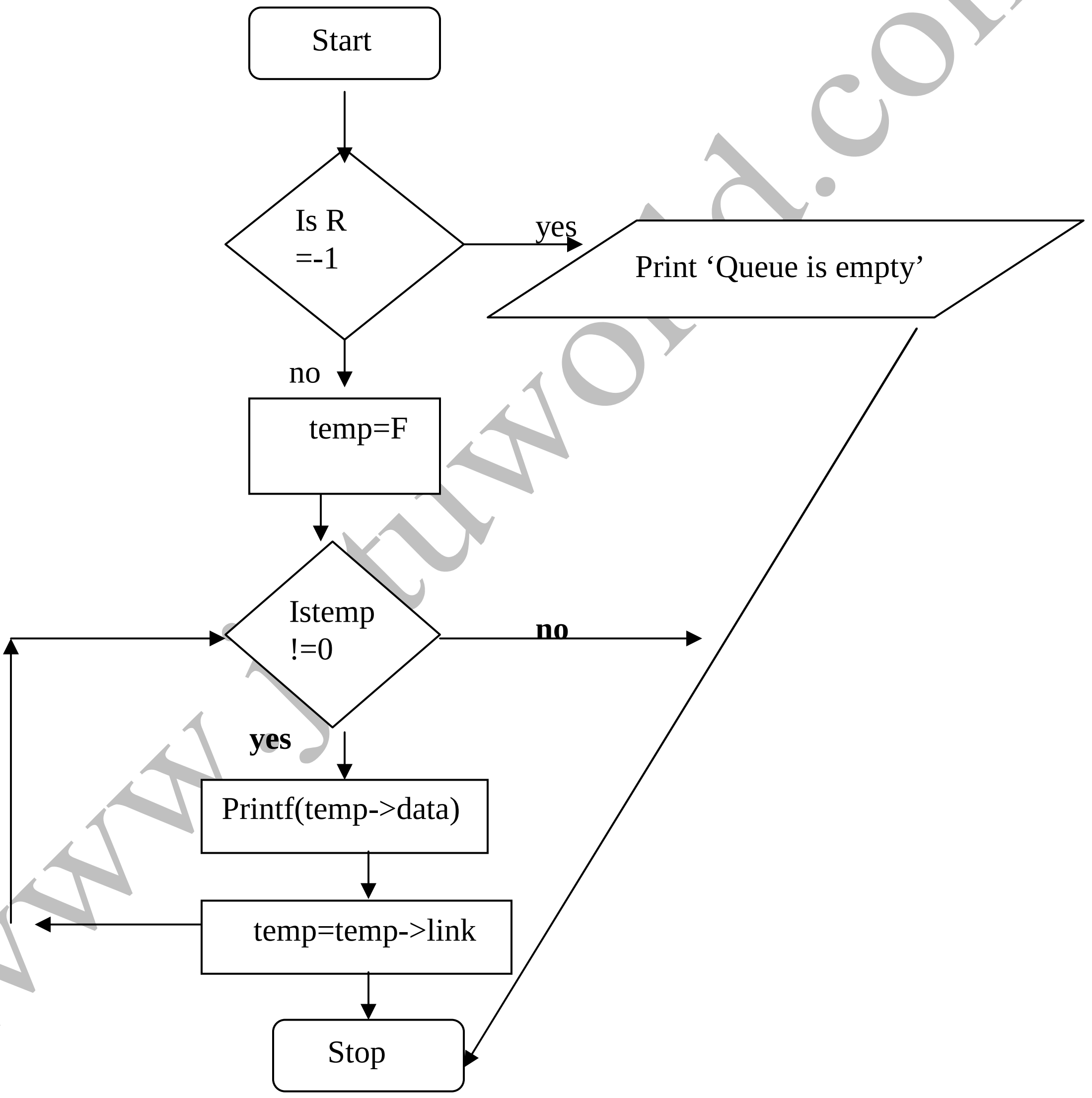
ITEM information to be inserted at the rear of queue



2) b) Flowchart For deleting an Item from a Queue Q:
function DELETE()
ITEM information to be inserted at the rear of queue



2) b)Algorithm For display Items into the Queue Q
function Dispaly(Q)
Q   Array



Program 2)b): Queue ADT using Linked Lists.

```
#include<iostream.h>
#include<conio.h>
class queue
{
   private:
   struct node
   {
   int data;
   node *link;
   }*f,*r;
   public:
   void init();
   int first();
   int last();
   int isempty();
   void insert(int);
   int del();
   void display();
   ~queue();
};
void queue::init()
{
   f=r=0;
```

```
}

int queue::isempty()
{
   if((f==0)&&(r==0))
   return(1);
   else
   return(0);
}
void queue::display()
 {
    node *temp;
    if(isempty()==1)
    cout<<"\nqueue is empty";
    else{
    temp=f;
    while(temp!=0)
    {
    cout<<temp->data<<"\t"<<temp->link<<"\n";
    temp=temp->link;
    }
    }
 }
void queue::insert(int x)
 {
    node *p;
    p=new node;

 p->data=x;
    p->link=0;
    if(isempty()==1)
    {
    f=r=p;
    }
    else
    {
    r->link=p;
    r=p;
    }
    if(f==0)
    f=p;
 }
int queue::del()
 {
   node *p;
    int x;
    if(isempty()==1)
    return(-1);
    else
    if(f==r)
```

```
{
p=f;
f=r=0;

x=p->data;
delete p;
return(x);
}
else
{
p=f;
f=f->link;
x=p->data;
delete p;
return(x);
}
}
queue::~queue()
{
while(f!=0)
{
node *temp;
temp=f;
f=f->link;
delete(temp);
```

```
 }
 r=0;
 }
 void menu2()

 {
 cout<<"\n"<<"1.insert"<<"\n"<<"2.deletion"<<"\n"<<"3.display queue
contents"<<"\n"<<"4.exit\n";
  }
void main()
{
    clrscr();
    queue q;
    int ch;
    menu2();
    cout<<"\n Enter your choice:";
    cin>>ch;
    while(ch<4)
    {
    switch(ch)
    {
    case 1:
    {
    int x;
    cout<<"\nEnter element of the queue:";
    cin>>x;
    q.insert(x);
    cout<<"\nelement is inserted";
    break;
    }
    case 2:

    {
    cout<<"\ndeleted element is:"<<q.del();
    break;
    }
    case 3:
    {
    cout<<"\ndisplay elements of the queue is:";
    q.display();
    break;
    }
    }
    menu2();
    cout<<"\n Enter your choice:";
    cin>>ch;
     }
     getch();
}
```

Output:

```
            1.insertion
            2.deletion
            3.display queue contents
            4.exit
Enter your choice:1
Enter element of the queue:10
element is inserted
            1.insertion
            2.deletion
            3.display queue contents
            4.exit
Enter your choice:1
Enter element of the queue is:20
element is inserted
            1.insertion
            2.deletion
            3.display queue contents
            4.exit
Enter your choice:1
Enter element of the queue is:30
element is inserted
            1.insertion
            2.deletion
            3.display contents
            4.exit
Enter your choice:3
queue contains as follows:
10  1000
20 1002
30 1004
            1.insertion
            2.deletion
            3.display queue contents
            4..exit
Enter your choice:2
deleted element is:10
            1.insertion
            2.deletion
            3.display queue contents
            4.exit
Enter your choice:3
queue contents as follows:
20 1002
30 1004
```

```
        1.insertion
        2.deletion
        3.display queue contents
        4.exit
Enter your choice:4
```

Program 3:

Double ended queue using double linked List

/* PROGRAM TO DEMONSTRATE DE-QUEUE USING LINKEDLIST*/

```
#include<conio.h>

#include<iostream.h>

#include<malloc.h>

#define NULL 0

class dlqueue

{

  public:

  int data;

  dlqueue *next,*prev,*first,*last;

  void insertAtFirst();

  void insertAtLast();

  void deleteAtFirst();

  void deleteAtLast();

  void display();


};

void main()

 {

  dlqueue dl;
```

```
int ch;

clrscr();

 dlqueue *temp;



while(1)

  {

    cout<<"\n     1.Insert Element at first";

    cout<<"\n     2.Insert Element at last";

    cout<<"\n     3.Delete Element at first";

    cout<<"\n     4.Delete Element at last";

    cout<<"\n     5.Display list";

    cout<<"\n       6.exit";

    cout<<"\n        Enter your choice(1/2/3...):   ";

    cin>>ch;


    switch(ch)

  {

    case 1: dl.insertAtFirst(); break;

    case 2: dl.insertAtLast(); break;

    case 3: dl.deleteAtFirst(); break;

    case 4: dl.deleteAtLast(); break;

    case 5: dl.display(); break;

    default :  cout<<"\n\tINVALID ENTRY";

  }
```

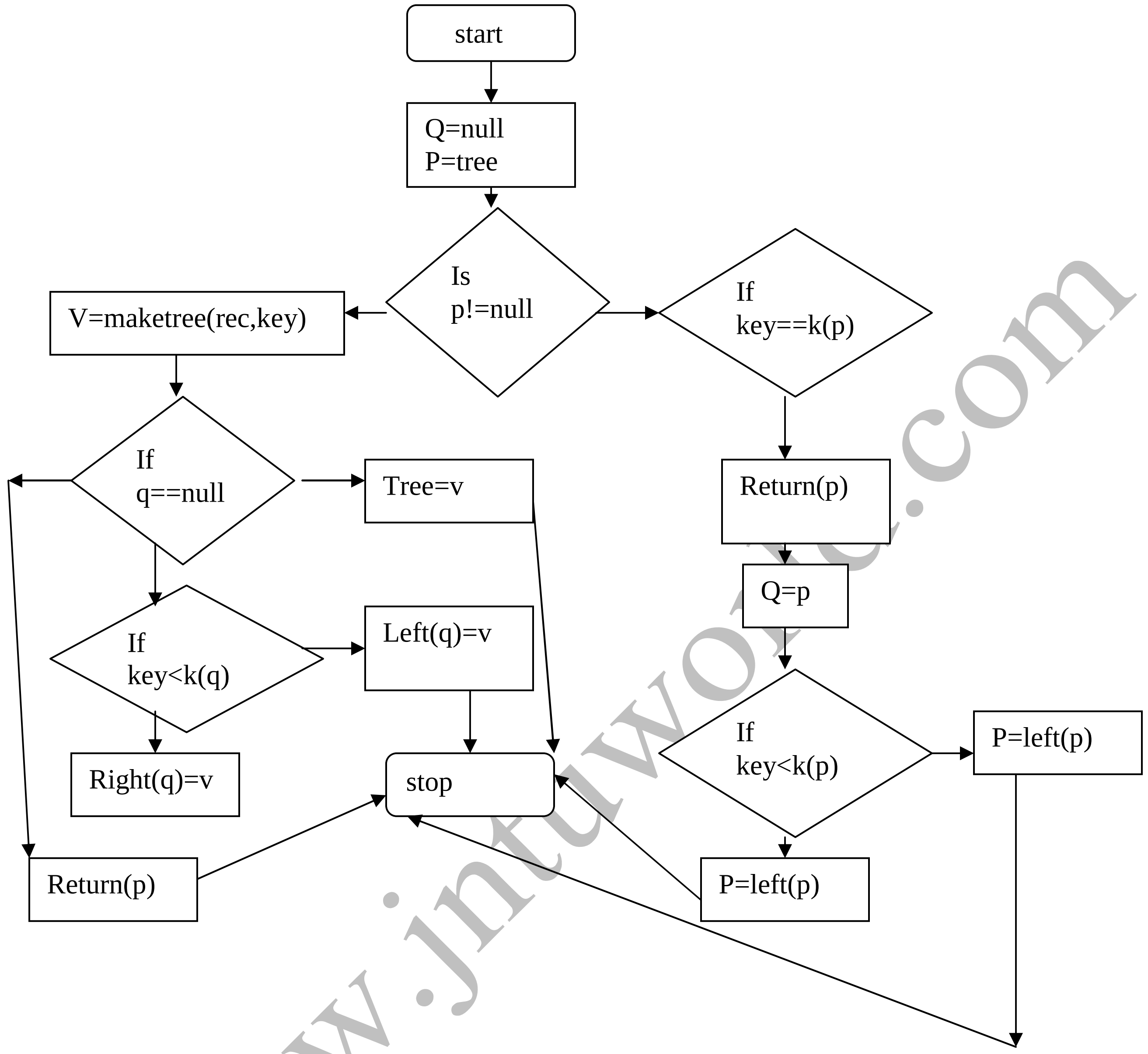
```
      }
   }




 void dlqueue::deleteAtFirst()
  {

    if(first->next==NULL)
      first=last=NULL;
    else
     {
       first=first->next;
       first->prev=NULL;
      }

    display();
    return;
   }

 void dlqueue::deleteAtLast()
  {
    if(first->next==NULL)
      first=last=NULL;
    else
     {
       last=last->prev;
       last->next=NULL;
      }



  display();
    return;
   }

 void dlqueue::insertAtLast()
  {
     dlqueue *temp;
     cout<<"\nEnter element:";
     cin>>temp->data;
     temp=temp->next;
     temp->next=NULL;
     temp->prev=NULL;
    if(first==NULL && last==NULL)
       first=last=temp;
   else
    {
      last->next=temp;
      temp->prev=last;
```

```
     last=temp;
   }


  display();
   return;
 }



void dlqueue::insertAtFirst()
 {
 dlqueue *temp;
   cout<<"\nEnter element:";
    cin>>temp->data;
               temp=temp->next;
    temp->next=NULL;
    temp->prev=NULL;

   if(first==NULL && last==NULL)
      first=last=temp;
      else
    {
      first->prev=temp;
       temp->next=first;
       first=temp;
       first->prev=NULL;
     }
    display();
   return;
  }


void dlqueue::display()
 {
   dlqueue *p;


  p=first;
   if(first==NULL && last==NULL)
     cout<<"\n\tList is Empty";
  else
   {
     cout<<"\n\n";
     while(p!=NULL)
     {
       cout<<p->data<<" ";
       p=p->next;
      }
    }   return;  }
```

Program 4://*binary serach tree searching,inserting,deleting*//
Algorithm:algorithm for inserting an element into the binary search tree

```
Step1:q=null,p=tree
Step2:repeate until p not equal to null
        a)if key==k(p) then
        return(p);
        q=p;
        b)if key<k(p) then
        p=left(p)
        else
        p=right(p);
step3:v=maketree(rec,key);
step4:if q==null then
        tree=v;
        else
        if(key<k(q) then
        left(q)=v;
        else
        right(q)=v;
step5:return(v);
```

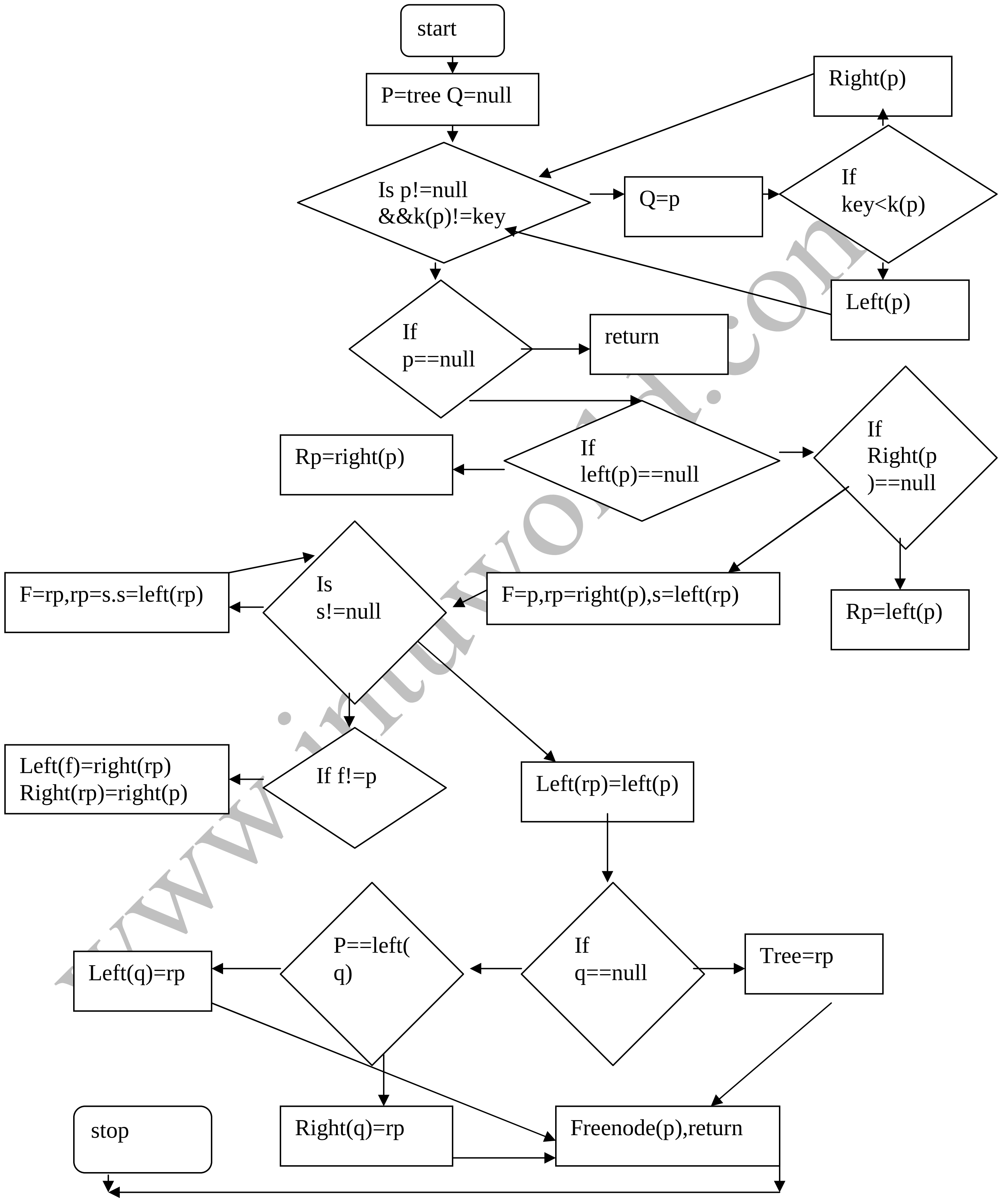
4)Flowchart:flowchart for inserting an element into the binary search tree



4)Algorithm:algorithm for deleting an element into the binary search tree

```
Step1:q=null,p=tree
Step2:repeate until p != null and k(p)!=key
        1)q=p;
        2)p=(key<k(p))?left(p):right(p);
Step3:if p==null
        Return
Step4:if left(p)==null then
        Rp=right(p)
        Else
        If(right(p)==null then
        Rp=left(p)
        Else
        a)f=p;
        rp=right(p);
        s=left(rp)
        b)repeate until s !=null
        f=rp;
        rp=s
        s=left(rp)
        c)if f!=p then
        left(f)=right(rp)
        right(rp)=right(p);
        d)left(rp)=left(p)
step5:a)if q==null
        tree=rp
        else
        if p==left(q) ? left(q)=rp:right(q)=rp
        b)freenode(p)
        return
```

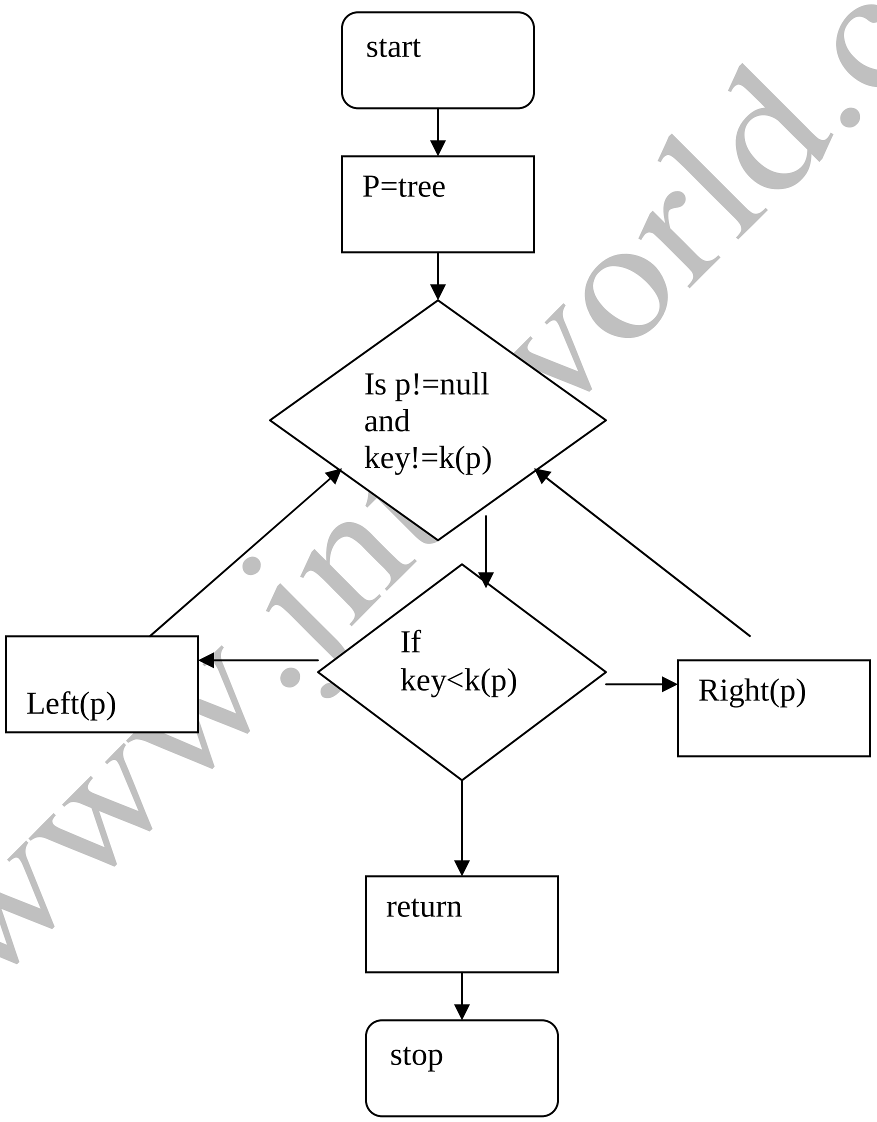
4)Algorithm:algorithm for deleting an element into the binary search tree



4)Algorithm:algorithm for searching an element into the binary search tree

```
Step1:p=tree
Step2:repeate until p!=null and key!=k(p)
        a)p=(key<k(p))?left(p):right(p);
        b)return
```

4)Flowchart: Flochart for searching an element into the binary search tree



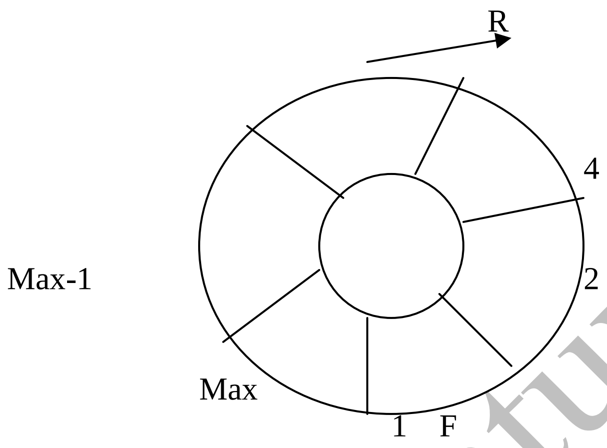
4)program:

```
#include<stdio.h>
#include<conio.h>
template<class e,class k>
class BSTree:public BinaryTree<e>
{
        public:
        bool Search(const k& k,E& e)const;
        BSTree<E,K>&Insert(const E& e);
        BSTree<E,K>&Delete(const K& k,E& e);
        void Ascend()
        {
        InOuput();
        }
};
template<class e,class k>
bool BSTree<E,K>::Search(const K& k,E &e)const
{
        BinaryTreeNode<E>*p=root;
        while(p);
        if(k<p->data)p=p->leftchild;
        else
        if(k>p->data)p=p->rightchild;
        else
        {
        e=p->data;
        rturn true;}
        return false;
}
template<class E,class k>
BSTree<E,K>&BSTree<E,k>::Insert(const E& e)
{
        binaryTreeNode<E>*p=root;
        *PP=0;
        while(p);
        {
        pp=p;
        if(e<p->data)p=p->leftchild;
        else
        if(e>p->data)p=p->Rightchild;
        else
        throw BadInput();
        }
        BinaryTreeNode<E>*r=new BinaryTreeNode<E>(e);
        if(root)
        {
        if(e<pp->data)pp->leftchild=r;
        else
        pp->rightchild=r;
        }
        else
        root=r;

        return *this;
}
template<class E, class K>
BSTree<E,K>& BSTree<E,K>::Delete(const k& k,E& e)
{//Delete element with key k and put it in e.
        // set p to point to node with key k
        BinaryTreeNode<E> *p = root, //search pointer
                        *pp = 0;// parent of p
        while (p && p->data != k)
        {//move to a child of p
        pp=p;
        if (k<P->data)p=p->LeftChild;
        else p=p->RightChild;
        }
        if(!p) throw BadInput(); // no element with key k
        e = P->data;//save element to delete
        //restructure tree
        // handle case when p has two children
        if (P->LeftChild && p->RightChild)
        {//two children
        //convert to zero or one child case
        // find largest element in left subtree of p
        BinaryTreeNode<E> *s = P->LeftChild,
                        *ps = p;//parent of s
        while(s->RightChild)
        {//move largest from s to p
        P->data = S->sata;
        P = s;
        pp = ps;
        // p has at most one child
        // save child pointer in c
        BinaryTreeNode<E> *c;
        if (P->LeftChild) C = P->LeftChild;
        else c + P->RightChild;
        //Delete p
        if (P == root) root = c;
        else{//is p left or right child of pp?
        if (p==pp->LeftChild)
        pp->LeftChild = c;
        else pp->RightChild = c;}
        delete p;
        return *this;
}
```

Circular Queue:

In a circular queue all locations are treated as circular such that the first location Q[1],follows the last location(Q[MAX])

Representation of circular Queue:



Basic Operation Associated on Circular Queue:
1) Insert an item into the Circular Queue.
2) Delete an item into the circular Queue.

1) a)Algorithm For Inserting an Item into a Circular Queue CQ:

```
Procudure CINSERT(CQ,MAX,F,R,ITEM)
CQ     Array
MAX Queue size
F front Pointer
R rear pointer
ITEM information to be inserted at the rear of queue.
Step1:{Reset rear pointer}
If  R=MAX then
        R=1
        Else
        R=R+1
Step2:{Check for overflow}
        If R=F then
        Print('Circular Queue overflow')
Step 3:{Insert an element into a curcular queue}
```

```
        CQ[R]=ITEM
Step 4:{set Front pointer property}
        If F=0,then F=1
        return
```

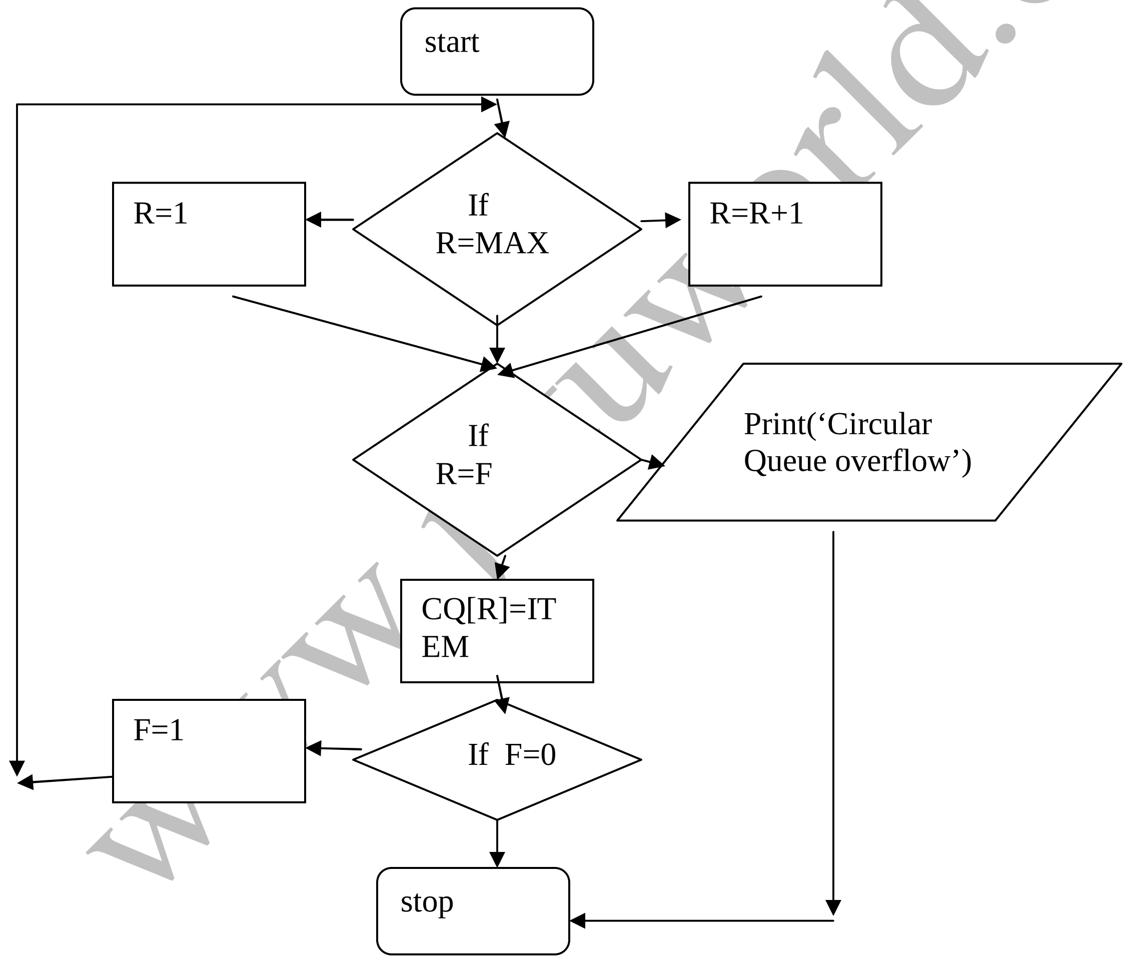
1) b) Algorithm For deleting an Item from a curcular Queue CQ:

```
function CDELETE(CQ,MAX,F,R)
CQ     Array
MAX Queue size
F front Pointer
R rear pointer
ITEM information to be inserted at the rear of queue.

Step1:{Check for underflow}
If F=R then
        Printf('Curcular Queue underflow')
        Return
Step2:
        If F=N then
        F=1
        Else
        F=F+1
Step3:{Delete element }
        ITEM=CQ[F]
        Return(ITEM)
 Step4:{check for circular queue is empty after deletion}
        If F=R then
        F=0
        R=0
    {Otherwise set front pointer}
        Else
        If F=MAX then
        F=1
        Else
        F=F+1
        Return(ITEM)
```

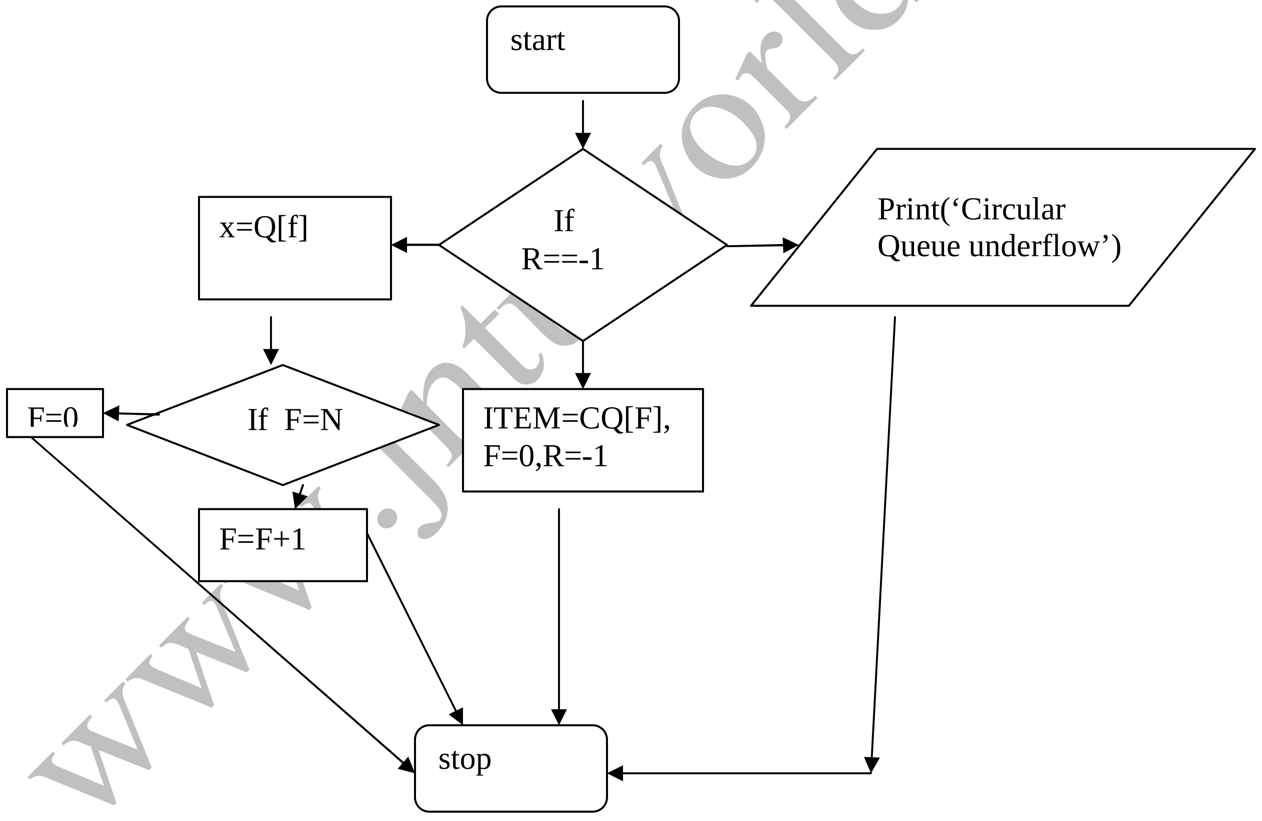
1) c)Algorithm For display Items into the Circular Queue CQ

```
function Dispaly(CQ)
CQ     Array
Step1: {Check queue values}
        If  F<0
                Print('Queue is empty')
Step2:{display Queue values}
        For I value F to R
        Print(CQ[I])
        I=I+1
```

1) a)Algorithm For Inserting an Item into a Circular Queue CQ:
Procudure CINSERT(CQ,MAX,F,R,ITEM)
CQ   Array
MAX Queue size
F front Pointer
R rear pointer
ITEM information to be inserted at the rear of queue.



1) b) Algorithm For deleting an Item from a curcular Queue CQ:
function CDELETE(CQ,MAX,F,R)
CQ Array
MAX Queue size
F front Pointer
R rear pointer
ITEM information to be inserted at the rear of queue.



Program 5:

Implement Circular Queue ADT using an Array

```
#include<iostream.h>

#include<conio.h>

#define size 10

template<class T>

class cq

{

private:

T q[size];

int f,r;

public:

void init();

void insert(T x);

void display();

T del();

int isfull();

int isempty();

T first();

T last();

 };

 template<class T>

 void cq<T>::init()

 {
```

```
f=0;


r=-1;

}

template<class T>

int cq<T>::isfull()

{

if(((r+1)%size==f)&&(r!=-1))

return(1);

else

return(0);

}

template<class T>

int cq<T>::isempty()

{

if(r==-1)

return(1);

else

return(0);

}

template<class T>

T cq<T>::first()

{

return(q[f]);

}
```

```
template<class T>
T cq<T>::last()

{
return(r);
}
template<class T>
void cq<T>::insert(T x)
{
if(isfull()==1)
cout<<"circuler queue is full.inseration is not posible\n";
 else
 {
 if(r==size-1)
 r=0;
 else
 r++;
 q[r]=x;
 cout<<"inserted\n";
 f=1;
 }
 }
 template<class T>
 T cq<T>::del()
 {
```

```
T x;
if(isempty()==1)
return(-1);

else
if(f==r)
{
x=q[f];
f=0;
r=-1;
return(x);
}
else
{
x=q[f];
if(f==size-1)
f=0;
else
f++;
return(x);
}
}
template<class T>
 void cq<T>::display()
 {
  int i;
  clrscr();
  if(isempty()==1)
  cout<<"circular queue is empty\n";

else
  if(f==r)
  {
  cout<<"circulr queue contains only one element\n";
  cout<<q[f]<<"\n";
  }
  else
  {
  if(f<r)
  {
  cout<<"circular contents are as folows\n";
  for(i=0;i<=f-1;i++)
  cout<<"empty\t";
  for(i=f;i<=r;i++)
  cout<<q[i]<<"\t";
  for(i=r-1;i<=size-1;i++)
  cout<<"empty\t";
  cout<<"\n";
  }
  else
  {
  cout<<"circular queue contain are as folows\n";
  for(i=0;i<=r;i++)
  cout<<q[i]<<"\t";
  for(i=r+1;i<=f-1;i++)

  cout<<q[i]<<"\t";
  cout<<"\n";}
  }
}
void menu1()
{
 cout<<"\n1.Interger queue";
 cout<<"\n2.float queue";
 cout<<"\n3.char queue";
 cout<<"\n4.exit";
}
void menu2()
{
 cout<<"\n1.insertion";
 cout<<"\n2.deletion";
 cout<<"\n3.display queue";
 cout<<"\n4.display first element";
 cout<<"\n5.exit";
}
```

```
template<class T>
void operation(cq<T>a)
{
 int ch;
 int x;
 menu2();

 cout<<"Enter your choice\n";
 cin>>ch;
 while(ch<5)
 {
 switch(ch)
 {
 case 1:
 {
 cout<<"Enter the element to be inserted\n";
 cin>>x;
 a.insert(x);
 break;
 }
 case 2:
 {
 x=a.del();
 if(x==-1)
 cout<<"queue is empty,del is not possible\n";
 else
 cout<<"deleted elelment is"<<x<<"\n";
 break;
 }
 case 3:
 {
 a.display();

break;
 }
 case 4:
 {
 x=a.first();
 cout<<"\n first element of the cqueue is:"<<x;
 break;
 }
 case 5:
 {
 x=a.last();
 cout<<"the last element is"<<x<<"\n";
 break;
 }
 }
 menu2();
 cout<<"Enter your choice\n";
 cin>>ch;
 }
}
main()
{
 int ch;
 menu1();
 cout<<"Enter your choice\n";

cin>>ch;
 while(ch<4)
 {
 switch(ch)
 {
 case 1:
 {
 cq<int>a;
 operation(a);
 break;
 }
 case 2:
 {
 cq<float>a;
 operation(a);
 break;
 }
 case 3:
 {
 cq<char>a;
 operation(a);
 break;
 }
 }

 menu1();
 cout<<"Enter your choice\n";
 cin>>ch;
 }
 getch();
 return(0);
}
```

Output:

```
            1.integer queue
            2.float queue
            3.char queue.
            4.exit.
Enter yourchoice:1
            1.insertion
            2.deletion
            3.display queue contents
            4.display front element
            5.exit
Enter your choice:1
Enter the element to be inserted:10
inserted
            1.insertion
            2.deletion
            3.display queue contents
            4.display front element
            5.display rear element
            6.exit

Enter your choice:1
Enter the element to be inserted:20
inserted
            1.insertion
            2.deletion
            3.display queue contents
            4.display front element
            5.display rear element
            6.exit


Enter your choice:1
Enter the element to be inserted:30
element is inserted
            1.insertion
            2.deletion
            4.display front element
            5.exit
Enter your choice:4
first element of the cqueue is:10
            1.insertion
            2.deletion
            3.display queue contents
            4.display front element
            5.exit
Enter your choice:3
```

```
queue contains as follows:
empty 10 20 30 empty empty empty empty empty empty empty empty empty
            1.insertion
            2.deletion
            3.display queue contents
            4.display front element
            5.exit
Enter your choice:2
deleted element of the  cqueue is:10
            1.insertion
            2.deletion
            3.display queue contents
            4.display front element
            5.exit

deleted element is:10
            1.insertion
            2.deletion
            3.display queue contents
            4.display front element
            5.exit
Enter your choice:3
dispalay contents as follows:
empty empty 20 30 empty empty empty empty empty empty empty empty empty

            1.integer queue
            2.float queue
            3.char queue.
            4.exit.

Enter your choice:4
```

Program: 6

Non-recursive function to traverse the given binary tree in in order, Pre order, post order

a)Algorithm: Algorithm for preorder traversal

```
Int function preorder(node *)
Step1:if pointer not equal to null then
        a)print p->data
        b)call function preorder(p->lchild)
        c)call function preotder(p->rchild)
```

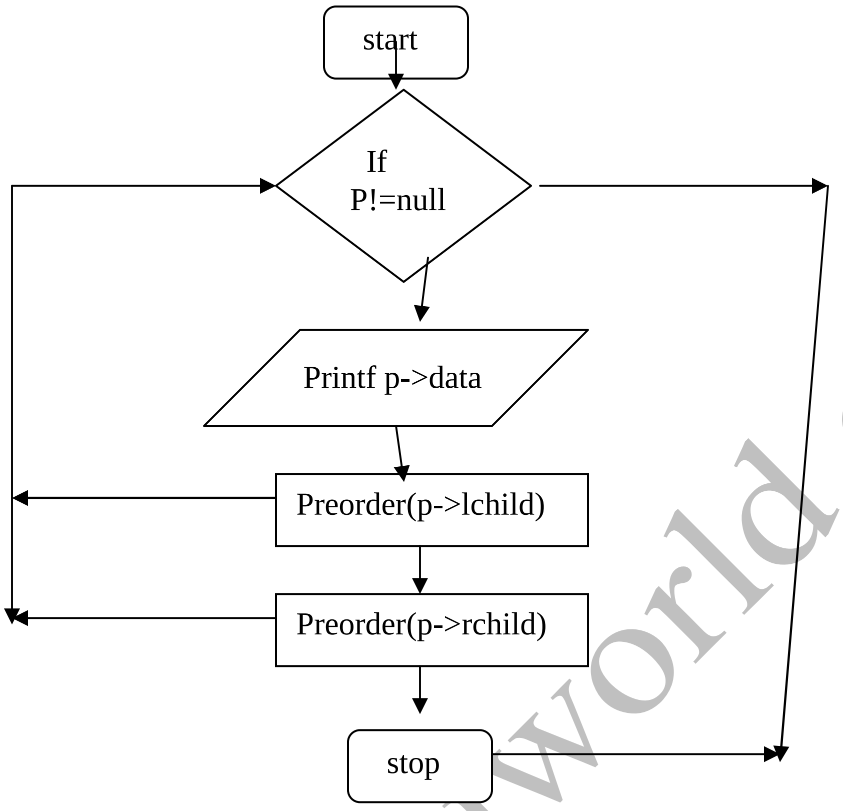
b)Algorithm: Algorithm for inorder traversal

```
Int function preorder(node *)
Step1:if pointer not equal to null then
        a)call function inorder(p->lchild)
        b)print p->data
        c)call function inorder(p->rchild)
```

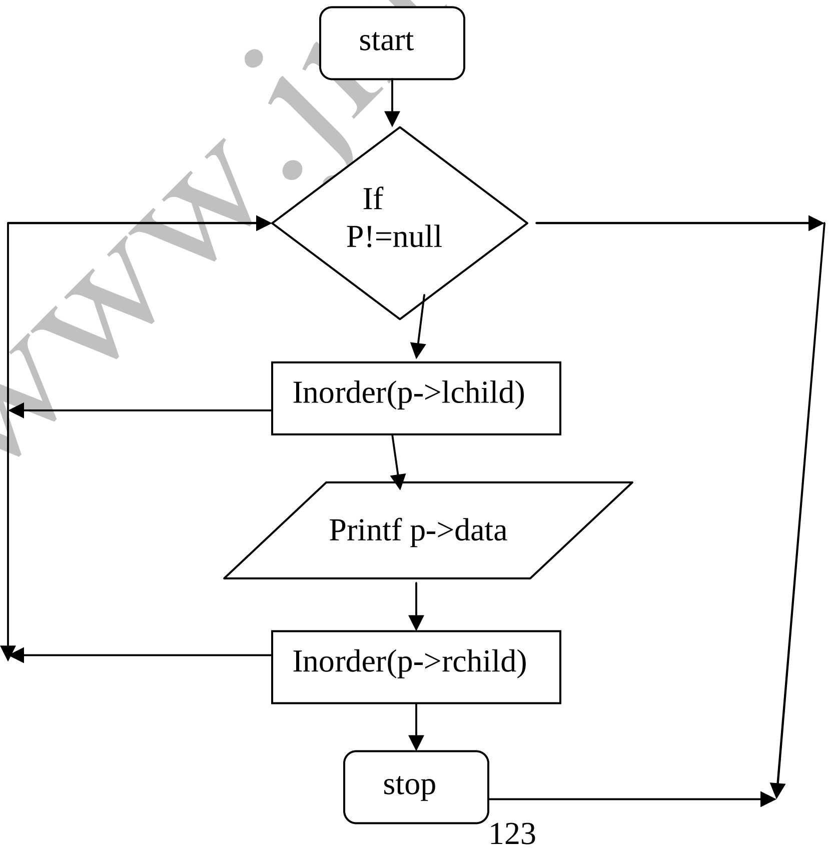
c)Algorithm: Algorithm for postorder traversal

```
Int function preorder(node *)
Step1:if pointer not equal to null then
        a)call function postorder(p->lchild)
        b)call function postotder(p->rchild)
        c)print p->data
```

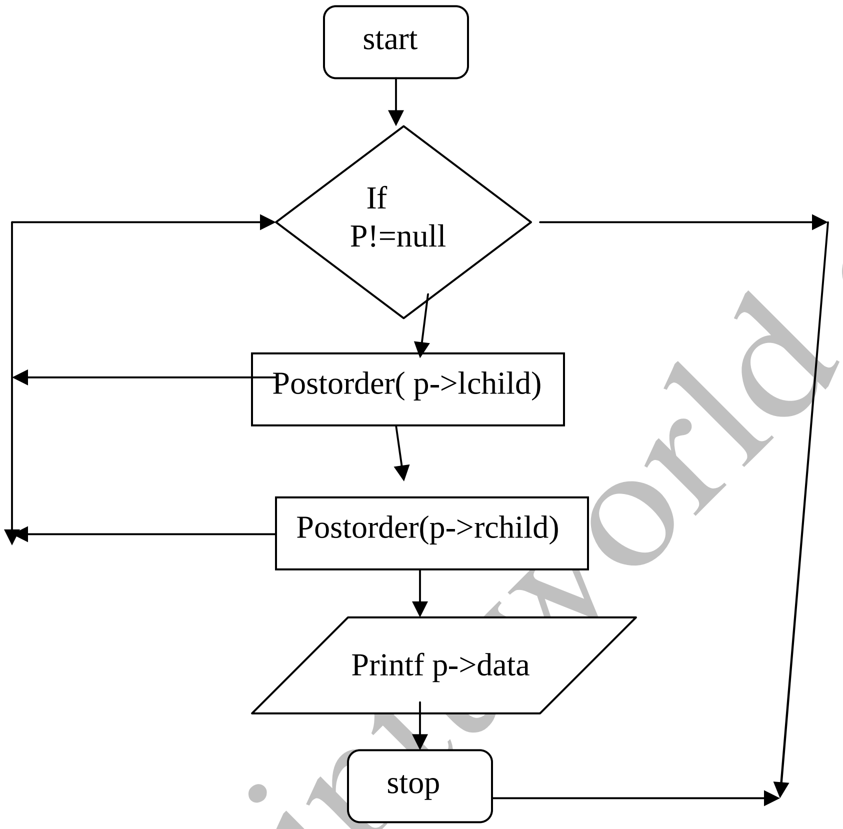
a)flowchart:flowchart for preorder traversal



a)flowchart:flowchart for inorder traversal



a)flowchart:flowchart for postorder traversal



Program6:

```
# include<iostream.h>

# include<conio.h>

class list

{

 public:

      int item;

      list *left;

      list *right;

};

void display(list *t,int i)

{

  int j;

    if (t!=NULL)

    {

    display(t->left,i+1);

    for(int j=1;j<i;j++)

     cout<<"  ";

    cout<<"  "<<t->item<<endl;

    display(t->right,i+1);

    }
 }
void preorder(list *t)
{
   if (t!=NULL)
   {


 cout<<" "<<t->item;
    preorder(t->left);
    preorder(t->right);
   }
}
void inorder(list *t)
{
   if (t!=NULL)
   {
    inorder(t->left);
    cout<<" "<<t->item;
    inorder(t->right);
   }
}
void postorder(list *t)
{
   if (t!=NULL)
   {
    postorder(t->left);
    postorder(t->right);
    cout<<" "<<t->item;
   }
}
list *create(list *t,int item)
{

  if (t==NULL)
      {
    t=new list;
    t->left=t->right=NULL;
    t->item=item;
    return t;
      }
   else
     if (t->item>item)
      t->left=create(t->left,item);
   else
     if (t->item<item)
     t->right=create(t->right,item);
   else
       cout<<endl<<"Duplicate Element";
 return t;
}
void main()
```

```
{
 list *start=NULL;
 int item;
 char wish;
 clrscr();
 do
  {


  cout<<"Enter Elment :";
    cin>>item;
    start=create(start,item);
    cout<<"Wish u continue(y/n):";
    cin>>wish;
  }
  while(wish=='y' || wish=='Y');
 cout<<endl<<"Given B Tree is "<<endl;
 display(start,1);
 cout<<endl<<"PreOrder ";
 preorder(start);
 cout<<endl<<"Inorder ";
 inorder(start);
 cout<<endl<<"PostOrder ";
 postorder(start);
getch();
}
```

```
> //program to implement inorder,preorder and postorder tree traversing.

#include<iostream.h>
#include<conio.h>
int a[6],i=0;
class list
{
 public:
 int item;
 list *left;
 list *right;
 };


/* void display(list *t,int i)
 {
  int j;
  if(t!=NULL)
  {
   display(t->left,i+1);
   cout<<" "<<t->item<<endl;
   display(t->right,i+1);
   }
   }
*/
    void preorder(list *t)
    {
    if(t!=NULL)
    {
    cout<<" "<<t->item;
    preorder(t->left);
    preorder(t->right);
    }
    }

    void inorder(list *t)
    {
    if(t!=NULL)
    {
    inorder(t->left);
    cout<<" "<<t->item;
    inorder(t->right);
    }
    }

    void postorder(list *t)
    {
        if(t!=NULL)
        {
        //cout<<" "<<t->item;
        postorder(t->left);
        postorder(t->right);
        cout<<" "<<t->item;
      }
    }

        list *create(list *t,int item)
        {
         if(t==NULL)
         {
          t=new list;
          t->left=t->right=NULL;
          t->item=item;
          a[i]=item;
          i++;
                return t;
          }
           else
           if(t->item>item)
           t->left=create(t->left,item);
           else
           if(t->item<item)
           t->right=create(t->right,item);
           else
           cout<<endl<<"duplicate elements";
           return t;
           }
           void dis()
           {
           for(int j=0;j<i;j++)
           cout<<a[j]<<endl;
                    }
           void main()
           {
                list *start=NULL;
                int item;
                char wish;
                clrscr();
                do
                {
                cout<<"enter element";
                cin>>item;
                start=create(start,item);
                cout<<"wish to continue(y/n):";
                cin>>wish;
                }
                while(wish=='Y'||wish=='y');
                cout<<endl<<"given btree is:"<<endl;
                dis();
                cout<<endl<<"preorder";
                preorder(start);
                cout<<endl<<"Inorder";
                inorder(start);
                cout<<endl<<"PostOrder";
                postorder(start);
                getch();
           }
```

Output:

```
        Enter element:9
        wish u continue(y/n):y
        Enter element:6
        wish u continue(y/n):y
        Enter element:5
        wish u continue(y/n):y
        Enter element:4
        wish u continue(y/n):y
        Enter element:3
        wish u continue(y/n):y
        Enter element:2
        wish u continue(y/n):y
        Enter element:1
        wish u continue(y/n):n
        given btree is:
                                1
                              2
                           3
                        4
                        5
                     6
                    7
        PreOrder:9 6 5 4 3 2 1
        InOrder:1 2 3 4 5 6 9
        PostOrder:1 2 3 4 5 6 9
Enter element:9
        wish u continue(y/n):y
        Enter element:7
        wish u continue(y/n):y
        Enter element:3
        wish u continue(y/n):y
        Enter element:6
        wish u continue(y/n):y
        Enter element:2
        wish u continue(y/n):y
        Enter element:1
        wish u continue(y/n):y
        Enter element:0
        wish u continue(y/n):n
        Enter element:4
        wish u continue(y/n):n
        given btree is:
                                0
                              1
                           2
                       3    4
                            6
                       7
        PreOrder:7 3 2 1 0 6 4
```

InOrder:0 1 2 3 4 6 7 PostOrder:0 1 2 4 6 3 7

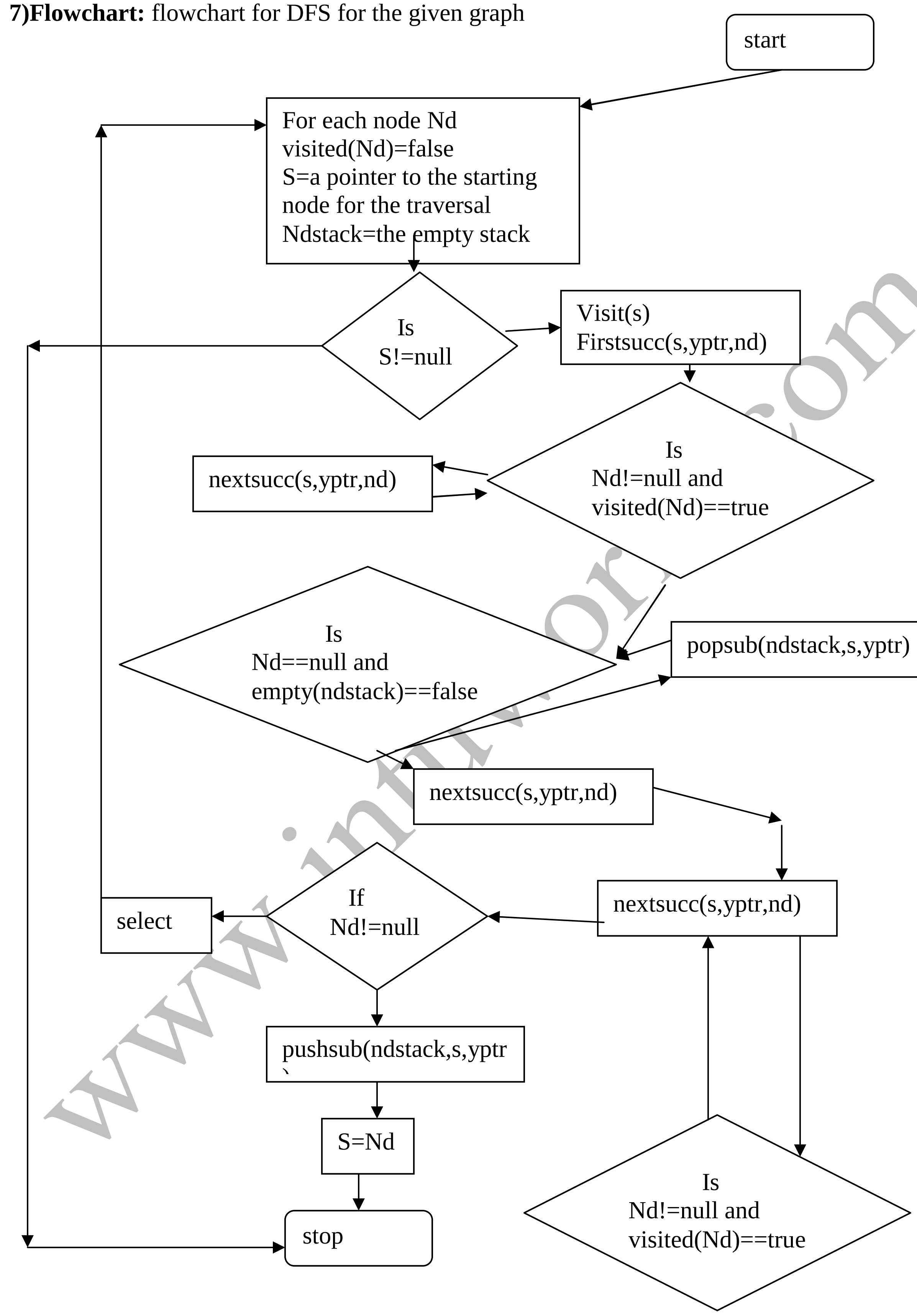
Program 7:

Implementation of BFS and DFS for the given graph.

Algorithm:algorithm for DFS for the given graph

```
Step1:for everynaode nd
Step2:visted(nd)=false
Step3:s= apointer to rhe starting node for the traversal
    ndstack=the empty stack
Step4:repeate until s !=null
    a)visit(s)
    b)firstsucc(s,yptr,nd)
    c)repeate until nd !=null and visited(nd)==true
        nextsucc(s,yptr,nd)
    d)repeate until nd==null and empty(ndstack)==false
        1)popsub(ndstack,s,yptr)
        nextsucc(s,yptr,nd)
        2)repeate until nd!=null and visited(nd)==true
        nextsucc(s,yptr,nd)
        3)if nd !=null
        Push(ndstack,s.yptr)
        S=nd
        Else
        S=select
```

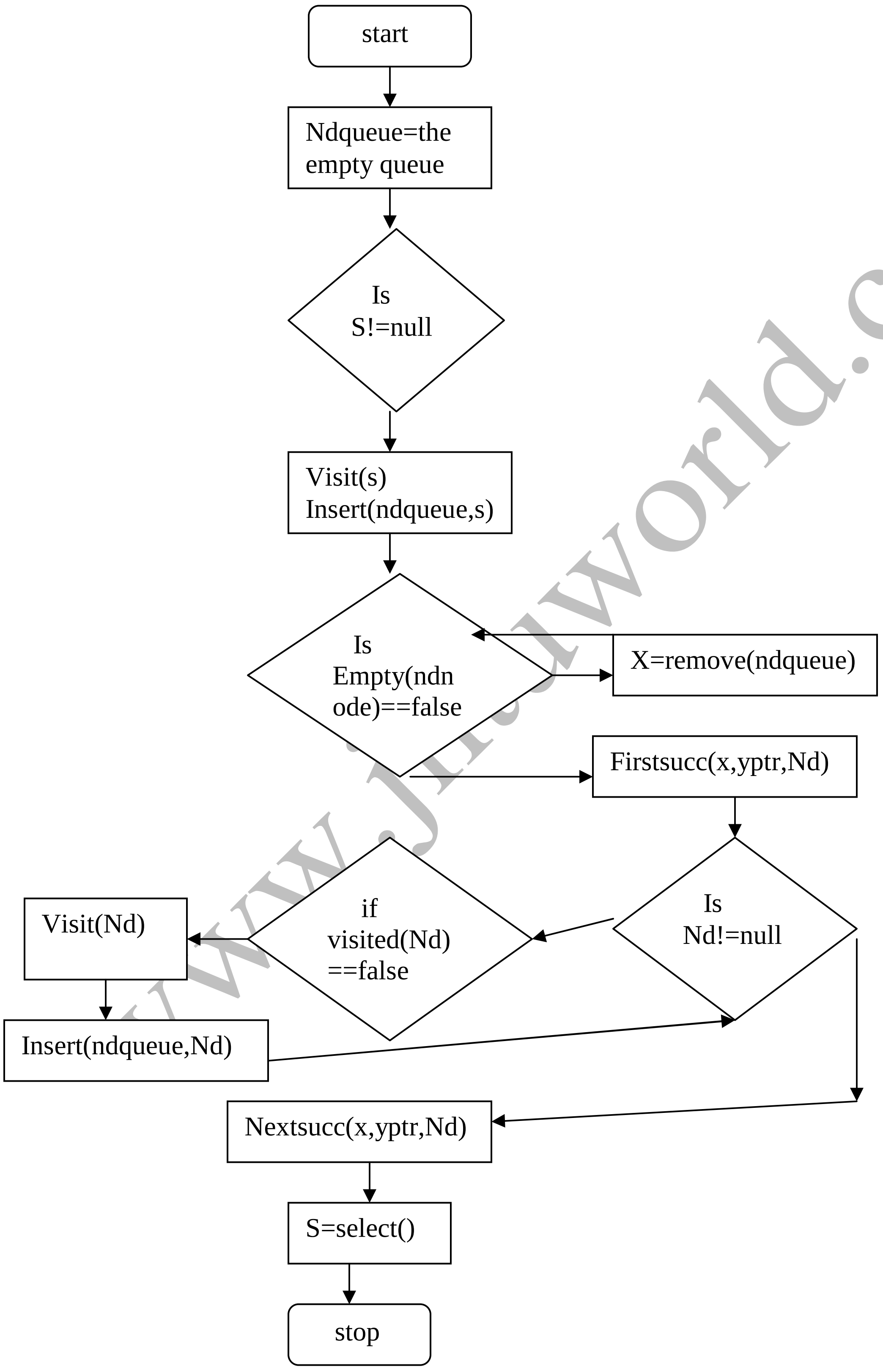
7)Flowchart: flowchart for DFS for the given graph



7)Algorithm:algorithm for BFS for the given graph

```
Step1:mdqueue=the empty queue
Step2:repeate until s !=null
    a)visit(s)
    b)insert(ndqueue,s)
    c)repeate empty(ndqueue)==false
        a)x=remove(ndqueue)
        b)firstsucc(x,yptr,nd
    d)repeate until nd!=null
        1)if visited(nd)==false then
            a)visit(nd)
            b)insert(ndqueue,nd)
        2)nextsucc(x,yptr,nd)
    e)s=select()
```

7)Algorithm:flowchart for BFS for the given graph



7)Program:

```
void network::bfs(int v, int reach[], int lable)
{//breadth first search.
          linkedqueue<int> q;
          initializepos();//init graph iterator array
          reach[v]=label;
          a.add(v);
          while(!q.isempty()){
          int w;
          q.deleter(w);
          int u = begin(w);
          while(u) {
          if (!reach[u]) {
          q.add(u);
          reach[u] = label;}
          u=nextvertex(w);
          }
}
void network::dfs(int v,int reach[], int label)
{
          initializepos();
          dfs(v, reach, label);
          deactivatepos();
}
void network::dfs(int v, int reach[], int label)
{
          reach[v]= label;
          int u=begin(v);
          while(u)
          {
          if(!reach[u]) dfs(u, reach, label);
          u= nextvertex(v);
          }
}
```

Or

8> BSTDS.

```
# include<iostream.h>
# include<conio.h>
template<class T>
class bst
{
 T info;
 bst *lptr;
 bst *rptr;
 public:
    bst *insert(bst *node,T item);
    void display(bst *node);
};
template<class T>
bst<T> * bst<T>::insert(bst<T> *node,T item)
{
  if (node==NULL)
     {
       node=new bst<T>;
       node->info=item;
       node->rptr=node->lptr=NULL;
       return node;
     }
  else
    if (node->info<item)
     node->rptr=insert(node->rptr,item);
  else
     node->lptr=insert(node->lptr,item);
  return node;
}
template<class T>
void bst<T>::display(bst<T> *node)
{
  if (node!=NULL)
      {
       cout<<node->info<<" ";
       display(node->lptr);
       display(node->rptr);
       }
}
void main()
{
 bst<int> *root;
 root=NULL;
 clrscr();
 root=root->insert(root,30);
 root=root->insert(root,20);
 root=root->insert(root,50);
 root=root->insert(root,25);
 cout<<"Elements in the bst are ";
 root->display(root);
```

Quick Sort

Algorithm Analysis:The quick sort is an in-place, divide-and-conquer, massively recursive sort. As a normal person would say, it's essentially a faster in-place version of the merge sort. The quick sort algorithm is simple in theory, but very difficult to put into code (computer scientists tied themselves into knots for years trying to write a practical implementation of the algorithm, and it still has that effect on university students).

The recursive algorithm consists of four steps (which closely resemble the merge sort):

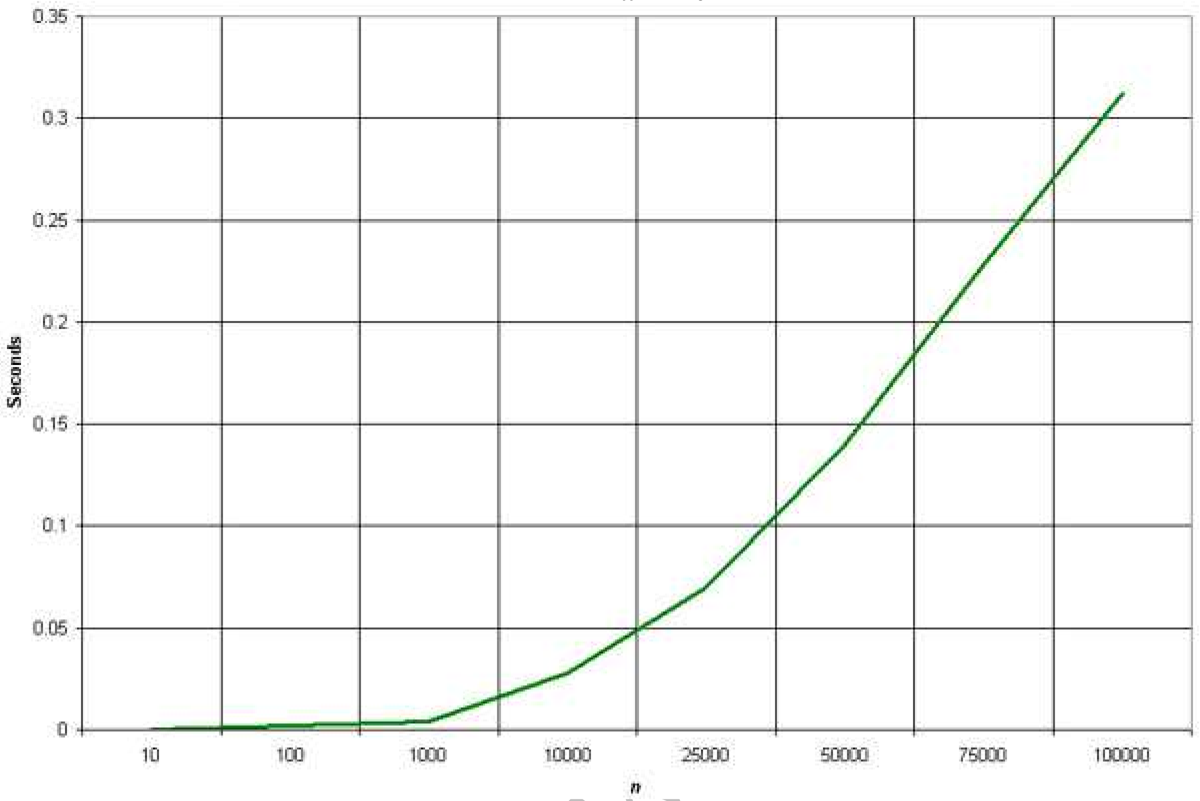
1. If there are one or less elements in the array to be sorted, return immediately.
2. Pick an element in the array to serve as a "pivot" point. (Usually the left-most element in the array is used.)
3. Split the array into two parts - one with elements larger than the pivot and the other with elements smaller than the pivot.
4. Recursively repeat the algorithm for both halves of the original array.

The efficiency of the algorithm is majorly impacted by which element is choosen as the pivot point. The worst-case efficiency of the quick sort, $O(n^2)$, occurs when the list is sorted and the left-most element is chosen. Randomly choosing a pivot point rather than using the left-most element is recommended if the data to be sorted isn't random. As long as the pivot point is chosen randomly, the quick sort has an algorithmic complexity of $O(n \log n)$.

Pros: Extremely fast.
Cons: Very complex algorithm, massively recursive.

Empirical Analysis

Quick Sort Efficiency



The quick sort is by far the fastest of the common sorting algorithms. It's possible to write a special-purpose sorting algorithm that can beat the quick sort for some data sets, but for general-case sorting there isn't anything faster.

As soon as students figure this out, their immediate implulse is to use the quick sort for everything - after all, faster is better, right? It's important to resist this urge - the quick sort isn't always the best choice. As mentioned earlier, it's massively recursive (which means that for very large sorts, you can run the system out of stack space pretty easily). It's also a complex algorithm - a little too complex to make it practical for a one-time sort of 25 items, for example.

With that said, in most cases the quick sort is the best choice if speed is important (and it almost always is). Use it for repetitive sorting, sorting of medium to large lists, and as a default choice when you're not really sure which sorting algorithm to use. Ironically, the quick sort has horrible efficiency when operating on lists that are mostly sorted in either forward or reverse order - avoid it in those situations.

Program 8:

Implement the following Sorting methods.
1) Quick sort 2) Merge sort 3) Heap sort.

8)1) write an algorithm for quick sort

procedure for quicksort(K,LB,UB)

K Array oa n elements

LB lower bound of the current sublist

UB upper bound of the current sublist

I Index variable

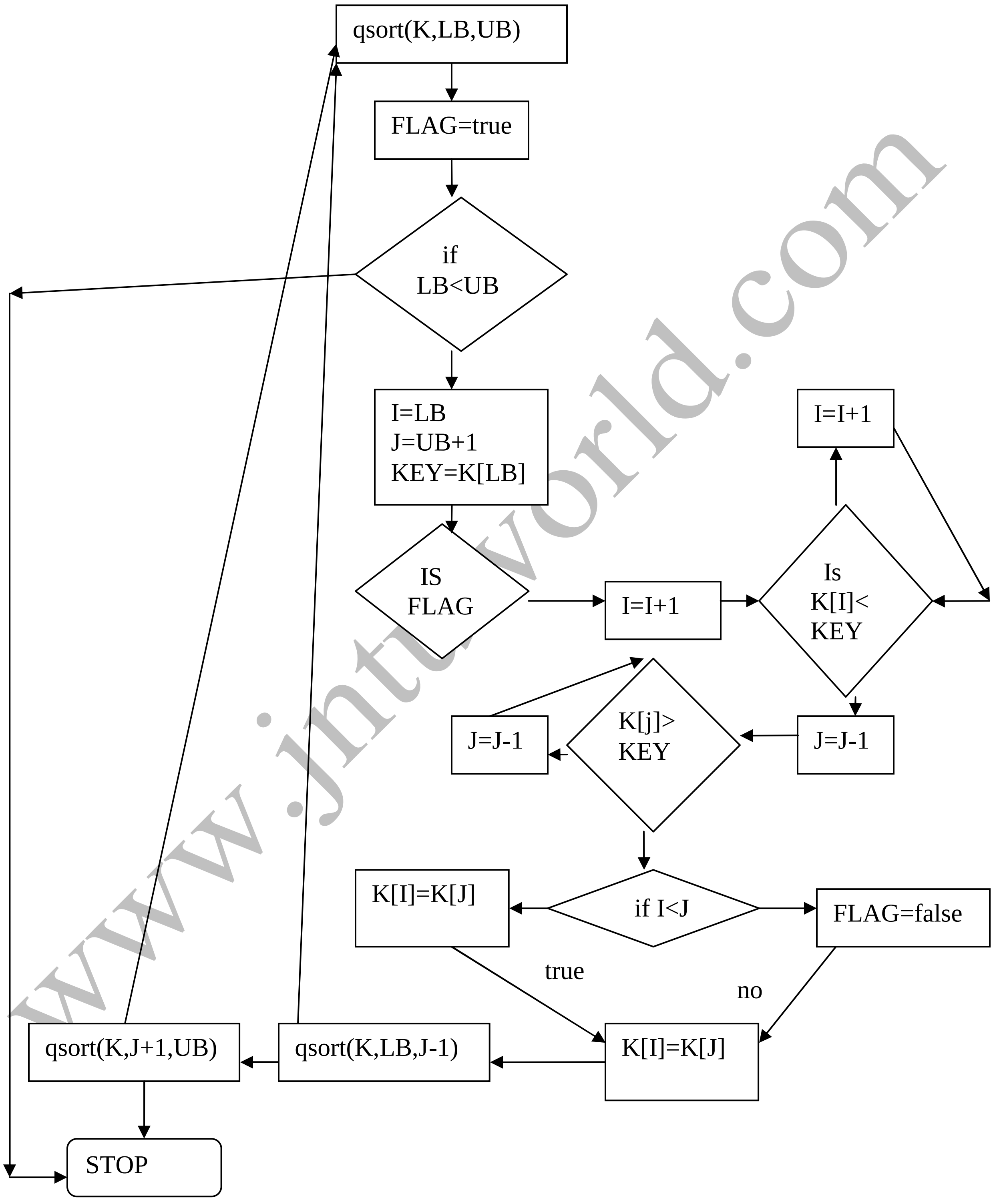
J Index variable

KEY key value

FLAG logical variable

```
step1:flag=true
step2: if LB<UB
        I=Lb
        J=Ub+1
        KEY=K{LB}
        repeate while(FLAG)
        I=I+1
                repeate while K[I]<KEY
                I=I+1
                J=J+1
                repeate while K[J]=>KEY
                J=J-1
                if I<J then
                        K[I]=K[J]
                else
                FLAG=false
        K[LB}=K[J]
        call QUICKSORT(K,LB,J-1)
        call QUICKSORT(K,J+1,UB)
step3:return
```

)1) write a flowchart for quick sort



8)1) write a program for quick sort

```
#include<iostream.h>

#include<conio.h>

void q_sort(int [],int,int);

void main()

{

        int n;

        int a[20];

        clrscr();

        cout<<"how many numbers do you want to Enter";

        cin>>n;

        cout<<"Enter numbers";

        for(int i=0;i<n;i++)

        cin>>a[i];

        q_sort(a,0,n-1);

        cout<<"display elements after quicksort";

        for(i=0;i<n;i++)

        cout<<a[i]<<"\t";

}


void q_sort(int numbers[], int left, int right)

{

  int pivot, l_hold, r_hold;



  l_hold = left;
```

```
r_hold = right;
pivot = numbers[left];
while (left < right)
{
  while ((numbers[right] >= pivot) && (left < right))
    right--;
  if (left != right)
  {
    numbers[left] = numbers[right];
    left++;
  }
  while ((numbers[left] <= pivot) && (left < right))
    left++;
  if (left != right)
  {
    numbers[right] = numbers[left];
    right--;
  }
}
numbers[left] = pivot;
pivot = left;
left = l_hold;
right = r_hold;
if (left < pivot)
  q_sort(numbers, left, pivot-1);
if (right > pivot)
  q_sort(numbers, pivot+1, right);
```

```
}
```

output:

```
how many numbers do you want to Enter:6

Enter numbers:9 8 7 5 3 4

elements after quicksort:3 4 5 7 8 9

how many numbers do you want to Enter:10

Enter numbers:9 8 7 5 3 4 1 -3 -2 0

elements after quicksort:-2 -3 0 1 3 4 5 7 8 9
```

)b)Mergesort algorithm:

Program:

```
#include<iostream.h>

#include<conio.h>

int l[20];

int l1[20];

int l2[20];

void sort();

void merge();

void main()

{

   int i,j,k;

   clrscr();

   cout<<"merge to already sorted lists";

   cout<<"Enter 10 integers for list1:";

   for(i=0;i<10;i++)

   cin>>l1[i];

   cout<<"Enter 10 numbers for list2:";

   for(i=0;i<10;i++)

   cin>>l2[i];

   sort();
```

```
    merge();
    cout<<"after merge list";
    for(i=0;i<20;i++)
    cout<<l[i];
    getch();
}
void sort()
{
    int i,j,t;
    for(i=0;i<9;i++)
    for(j=0;j<9;j++)
    {
    if(l1[j]>l1[j+1])
    {
    t=l1[j];
    l1[j]=l1[j+1];
    l1[j+1]=t;
    }
    if(l2[j]>l2[j+1])
    {
    t=l2[j];
    l2[j]=l2[j+1];
    l2[j+1]=t;
    }
    }
```

```
}

void merge()

{

    int i=0,j=0,k=0;

    while(k<20)

    {

    if(l1[i]<l2[j])

    l[k++]=l1[i++];

    else

    l[k++]=l2[j++];

    }

}
```

or

```
> Merge Sort.
#include<iostream.h>
#include<conio.h>
class msort
{
 public:
           int n,a[20];
           msort(int k)
           {
            n=k;
            }
            void read();
            void disp();
            void sort(int,int);
            void merge(int,int,int);
            };

            void msort::read()
            {
             int i;
             cout<<"\n enter the elements:";
             for(i=0;i<n;i++)
             {
```

```
cin>>a[i];
}
}

void msort::disp()
{
 int i;
 for(i=0;i<n;i++)
 {
  cout<<a[i]<<"\t";
  }
  }

  void msort::sort(int m,int n)
  {
   int h;
   if(m<n)
   {
    h=(m+n)/2;
    sort(m,h);
    sort(h+1,n);
    merge(m,h,n);
    }
    }
    void msort::merge(int m,int h,int n)
    {
     int i,j,k,b[20];

     i=m;
     j=h+1;
     k=m;
     while((i<=h)&&(j<=n))
     {
     if(a[i]<a[j])
     {
     b[k]=a[i];
     k=k+1;
     i=i+1;
     }
     else
     {
     b[k]=a[j];
     k=k+1;
     j=j+1;
     }
     }
     while(i<=h)
     {
     b[k]=a[i];
     k++;
```

```
i++;
}
while(j<=n)
{
b[k]=a[j];
k++;
j++;
}
    for(i=m;i<=n;i++)
    a[i]=b[i];
    }

    void main()
    {
    clrscr();
    int n;
    char ch='y';
    while(ch=='y'||ch=='Y')

    {
    cout<<"Enter the no:";
    cin>>n;
    msort b(n);
    b.read();
    b.sort(0,n);
    cout<<"\n The sorted array is :"<<endl;
    b.disp();
    cout<<"\n Do you want to continue ?";
    cin>>ch;
    }
    getch();

    }
```

Output:

```
Enter 10 numbers for list1:0 1 2 3 4 5 6 7 8 9

Enter 10 numbers for list1:0 1 2 3 4 5 6 7 8 9

After merge list:0 0 1 1 2 2 3 3 4 4 5 5 6 6 7 7  8 8 9 9
```

8)c)Heap sort algorithm:

step1:arrange elemnets of a list in correct form of a binary tree

step2:remove top mostelements of the heap

step3:rearrange the remaining elements from a heap this process is continued till we getsortedlist

Program: Heap Sort.

```
#include<iostream.h>

#include<conio.h>

#define max 20;

int heap[21];

void insert(int,int,int);

void makeheap(int);

void heapsort(int);

void main()

{

   int i,j,n;

   clrscr();

   cout<<"how many numbers are there for sorting?:";

   cin>>n;

   for(i=1;i<=n;i++)

   cin>>heap[i];

   makeheap(n);

   cout<<"after heapsort";

   heapsort(n);

   for(i=1;i<=n;i++)
   cout<<heap[i]<<" ";
   cout<<"\n";
   getch();
}
void makeheap(int size)
{
   int k,kmax;
   kmax=size/2;
   for(k=kmax;k>=1;k--)
   insert(heap[k],k,size);
}
void insert(int i,int n,int s)
{
   int c,t;
   c=n*2;
   while(c<=s)
   {
   if(c<s && heap[c]<heap[c+1])
   c++;
   if(i>=heap[c])
   break;
   else
   {
   t=heap[n];
   heap[n]=heap[c];
```

```
                heap[c]=t;
                n=c;
                c=n*2;
                }
                }
            }

            void heapsort(int s)
            {
                int i,j,t;
                for(i=s;i>1;i--)
                {
                        t=heap[i];
                        heap[i]=heap[1];
                        heap[1]=t;
                        insert(heap[1],1,i-1);
                }
            }
```

Output:

```
how many numbers do you want to Enter:5
Enter numbers:6 4 3 2 1
elements after mergesort:1 2 3 4 6
how many numbers do you want to Enter:6
Enter numbers:5 3 4 1 -3  0
```

```
elements after mergesort:-3 0 1 3 4 5
```

Program :9

Insertion into a B Tree and deletion from a B tree.

B+ TREES

B Trees. B Trees are multi-way trees. That is each node contains a set of keys and pointers. A B Tree with four keys and five pointers represents the minimum size of a B Tree node. A B Tree contains only data pages.

B Trees are dynamic. That is, the height of the tree grows and contracts as records are added and deleted.

B+ Trees A B+ Tree combines features of ISAM and B Trees. It contains index pages and data pages. The data pages always appear as leaf nodes in the tree. The root node and intermediate nodes are always index pages. These features are similar to ISAM. Unlike ISAM, overflow pages are not used in B+ trees.

The index pages in a B+ tree are constructed through the process of inserting and deleting records. Thus, B+ trees grow and contract like their B Tree counterparts. The contents and the number of index pages reflects this growth and shrinkage.

B+ Trees and B Trees use a "fill factor" to control the growth and the shrinkage. A 50% fill factor would be the minimum for any B+ or B tree. As our example we use the smallest page structure. This means that our B+ tree conforms to the following guidelines.

Number of Keys/page	4
Number of Pointers/page	5
Fill Factor	50%
Minimum Keys in each page	2

As this table indicates each page must have a minimum of two keys. The root page may violate this rule.

The following table shows a B+ tree. As the example illustrates this tree does not have a full index page. (We have room for one more key and pointer in the root page.) In addition, one of the data pages contains empty slots.

B+ Tree with four keys

Adding Records to a B+ Tree

The key value determines a record's placement in a B+ tree. The leaf pages are maintained in sequential order AND a doubly linked list (not shown) connects each leaf page with its sibling page(s). This doubly linked list speeds data movement as the pages grow and contract.

We must consider three scenarios when we add a record to a B+ tree. Each scenario causes a different action in the insert algorithm. The scenarios are:

The insert algorithm for B+ Trees

Leaf Page Full	**Index Page FULL**	**Action**
NO	NO	Place the record in sorted position in the appropriate leaf page
YES	NO	1. Split the leaf page 2. Place Middle Key in the index page in sorted order. 3. Left leaf page contains records with keys below the middle key. 4. Right leaf page contains records with keys equal to or greater than the middle key.
YES	YES	1. Split the leaf page. 2. Records with keys < middle key go to the left leaf page. 3. Records with keys >= middle key go to the right leaf page. 4. Split the index page. 5. Keys < middle key go to the left index page. 6. Keys > middle key go to the right index page. 7. The middle key goes to the next (higher level) index. IF the next level index page is full, continue splitting the index pages.

Illustrations of the insert algorithm

The following examples illlustrate each of the **insert** scenarios. We begin with the simplest scenario: inserting a record into a leaf page that is not full. Since only the leaf node containing 25 and 30 contains expansion room, we're going to insert a record with a key value of 28 into the B+ tree. The following figures shows the result of this addition.

Add Record with Key 28

Adding a record when the leaf page is full but the index page is not

Next, we're going to insert a record with a key value of 70 into our B+ tree. This record should go in the leaf page containing 50, 55, 60, and 65. Unfortunately this page is full. This means that we must split the page as follows:

Left Leaf Page	Right Leaf Page
50 55	60 65 70

The middle key of 60 is placed in the index page between 50 and 75.

The following table shows the B+ tree after the addition of 70.

Add Record with Key 70

Adding a record when both the leaf page and the index page are full

As our last example, we're going to add a record containing a key value of 95 to our B+ tree. This record belongs in the page containing 75, 80, 85, and 90. Since this page is full we split it into two pages:

Left Leaf Page	Right Leaf Page
75 80	85 90 95

The middle key, 85, rises to the index page. Unfortunately, the index page is also full, so we split the index page:

Left Index Page	Right Index Page	New Index Page
25 50	75 85	60

The following table illustrates the addition of the record containing 95 to the B+ tree.

Add Record with Key 95

Rotation

B+ trees can incorporate rotation to reduce the number of page splits. A rotation occurs when a leaf page is full, but one of its sibling pages is not full. Rather than splitting the leaf page, we move a record to its sibling, adjusting the indices as necessary. Typically, the left sibling is checked first (if it exists) and then the right sibling.

As an example, consider the B+ tree before the addition of the record containing a key of 70. As previously stated this record belongs in the leaf node containing 50 55 60 65. Notice that this node is full, but its left sibling is not.

Add Record with Key 28

Using rotation we shift the record with the lowest key to its sibling. Since this key appeared in the index page we also modify the index page. The new B+ tree appears in the following table.

Illustration of Rotation

Deleting Keys from a B+ tree

We must consider three scenarios when we delete a record from a B+ tree. Each scenario causes a different action in the delete algorithm. The scenarios are:

The delete algorithm for B+ Trees

Leaf Page Below Fill Factor	**Index Page Below Fill Factor**	**Action**
NO	NO	Delete the record from the leaf page. Arrange keys in ascending order to fill void. If the key of the deleted record appears in the index page, use the next key to replace it.
YES	NO	Combine the leaf page and its sibling. Change the index page to reflect the change.
YES	YES	1. Combine the leaf page and its sibling. 2. Adjust the index page to reflect the change. 3. Combine the index page with its sibling. Continue combining index pages until you reach a page

		with the correct fill factor or you reach the root page.

As our example, we consider the B+ tree after we added 95 as a key. As a refresher this tree is printed in the following table.

Add Record with Key 95

Delete 70 from the B+ Tree

We begin by deleting the record with key 70 from the B+ tree. This record is in a leaf page containing 60, 65 and 70. This page will contain 2 records after the deletion. Since our fill factor is 50% or (2 records) we simply delete 70 from the leaf node. The following table shows the B+ tree after the deletion.

Delete Record with Key 70

Delete 25 from the B+ tree

Next, we delete the record containing 25 from the B+ tree. This record is found in the leaf node containing 25, 28, and 30. The fill factor will be 50% after the deletion; however, 25 appears in the index page. Thus, when we delete 25 we must replace it with 28 in the index page.

The following table shows the B+ tree after this deletion.

Delete Record with Key 25

Delete 60 from the B+ tree

As our last example, we're going to delete 60 from the B+ tree. This deletion is interesting for several resasons:

1. The leaf page containing 60 (60 65) will be below the fill factor after the deletion. Thus, we must combine leaf pages.
2. With recombined pages, the index page will be reduced by one key. Hence, it will also fall below the fill factor. Thus, we must combine index pages.
3. Sixty appears as the only key in the root index page. Obviously, it will be removed with the deletion.

The following table shows the B+ tree after the deletion of 60. Notice that the tree contains a single index page.

Delete Record with Key 60

The Structure of B-Trees

Unlike a binary-tree, each node of a b-tree may have a variable number of keys and children. The keys are stored in non-decreasing order. Each key has an associated child that is the root of a subtree containing all nodes with keys less than or equal to the key but greater than the preceeding key. A node also has an additional rightmost child that is the root for a subtree containing all keys greater than any keys in the node.

A b-tree has a minumum number of allowable children for each node known as the *minimization factor*. If *t* is this *minimization factor*, every node must have at least *t - 1* keys. Under certain circumstances, the root node is allowed to violate this property by having fewer than *t - 1* keys. Every node may have at most *2t - 1* keys or, equivalently, *2t* children.

Since each node tends to have a large branching factor (a large number of children), it is typically neccessary to traverse relatively few nodes before locating the desired key. If access to each node requires a disk access, then a b-tree will minimize the number of disk accesses required. The minimzation factor is usually chosen so that the total size of each node corresponds to a multiple of the block size of the underlying storage device. This choice simplifies and optimizes disk access. Consequently, a b-tree is an ideal data structure for situations where all data cannot reside in primary storage and accesses to secondary storage are comparatively expensive (or time consuming).

Height of B-Trees

For *n* greater than or equal to one, the height of an *n*-key b-tree T of height *h* with a minimum degree *t* greater than or equal to 2,

$$h \leq \log_t \frac{n+1}{2}$$

For a proof of the above inequality, refer to Cormen, Leiserson, and Rivest pages 383-384.

The worst case height is *O(log n)*. Since the "branchiness" of a b-tree can be large compared to many other balanced tree structures, the base of the logarithm tends to be large; therefore, the number of nodes visited during a search tends to be smaller than required by other tree structures. Although this does not affect the asymptotic worst case height, b-trees tend to have smaller heights than other trees with the same asymptotic height.

Operations on B-Trees

The algorithms for the *search*, *create*, and *insert* operations are shown below. Note that these algorithms are single pass; in other words, they do not traverse back up the tree. Since b-trees strive to minimize disk accesses and the nodes are usually stored on disk, this single-pass approach will reduce the number of node visits and thus the number of disk accesses. Simpler double-pass approaches that move back up the tree to fix violations are possible.

Since all nodes are assumed to be stored in secondary storage (disk) rather than primary storage (memory), all references to a given node be be preceeded by a read operation denoted by *Disk-Read*. Similarly, once a node is modified and it is no longer needed, it must be written out to secondary storage with a write operation denoted by *Disk-Write*. The algorithms below assume that all nodes referenced in parameters have already had a corresponding *Disk-Read* operation. New nodes are created and assigned storage with the *Allocate-Node* call. The implementation details of the *Disk-Read*, *Disk-Write*, and *Allocate-Node* functions are operating system and implementation dependent.

B-Tree-Search(x, k)

```
i <- 1
while i <= n[x] and k > $key_i[x]$
    do i <- i + 1
if i <= n[x] and k = $key_i[x]$
    then return (x, i)
if leaf[x]
    then return NIL
    else Disk-Read($c_i[x]$)
        return B-Tree-Search($c_i[x]$, k)
```

The search operation on a b-tree is analogous to a search on a binary tree. Instead of choosing between a left and a right child as in a binary tree, a b-tree search must make an n-way choice. The correct child is chosen by performing a linear search of the values in the node. After finding the value greater than or equal to the desired value, the child pointer to the immediate left of that value is followed. If all values are less than the desired value, the rightmost child pointer is followed. Of course, the search can be terminated as soon as the desired node is found. Since the running time of the search operation depends upon the height of the tree, *B-Tree-Search* is $O(\log_t n)$.

B-Tree-Create(T)

```
x <- Allocate-Node()
leaf[x] <- TRUE
n[x] <- 0
Disk-Write(x)
root[T] <- x
```

The *B-Tree-Create* operation creates an empty b-tree by allocating a new root node that has no keys and is a leaf node. Only the root node is permitted to have these properties; all other nodes must meet the criteria outlined previously. The *B-Tree-Create* operation runs in time $O(1)$.

B-Tree-Split-Child(x, i, y)

```
z <- Allocate-Node()
leaf[z] <- leaf[y]
n[z] <- t - 1
for j <- 1 to t - 1
    do $key_j[z]$ <- $key_{j+t}[y]$
if not leaf[y]
    then for j <- 1 to t
        do $c_j[z]$ <- $c_{j+t}[y]$
n[y] <- t - 1
for j <- n[x] + 1 downto i + 1
    do $c_{j+1}[x]$ <- $c_j[x]$
$c_{i+1}$ <- z
```

```
for j <- n[x] downto i
    do key_{j+1}[x] <- key_j[x]
key_i[x] <- key_t[y]
n[x] <- n[x] + 1
Disk-Write(y)
Disk-Write(z)
Disk-Write(x)
```

If is node becomes "too full," it is necessary to perform a split operation. The split operation moves the median key of node *x* into its parent *y* where *x* is the *i*th child of *y*. A new node, *z*, is allocated, and all keys in *x* right of the median key are moved to *z*. The keys left of the median key remain in the original node *x*. The new node, *z*, becomes the child immediately to the right of the median key that was moved to the parent *y*, and the original node, *x*, becomes the child immediately to the left of the median key that was moved into the parent *y*.

The split operation transforms a full node with *2t - 1* keys into two nodes with *t - 1* keys each. Note that one key is moved into the parent node. The *B-Tree-Split-Child* algorithm will run in time *O(t)* where *t* is constant.

B-Tree-Insert(T, k)

```
r <- root[T]
if n[r] = 2t - 1
    then s <- Allocate-Node()
        root[T] <- s
                leaf[s] <- FALSE
                n[s] <- 0
                $c_1$ <- r
                B-Tree-Split-Child(s, 1, r)
                B-Tree-Insert-Nonfull(s, k)
    else B-Tree-Insert-Nonfull(r, k)
```

B-Tree-Insert-Nonfull(x, k)

```
i <- n[x]
if leaf[x]
    then while i >= 1 and k < $key_i$[x]
          do $key_{i+1}$[x] <- $key_i$[x]
                i <- i - 1
        $key_{i+1}$[x] <- k
                n[x] <- n[x] + 1
                Disk-Write(x)
    else while i >= and k < $key_i$[x]
          do i <- i - 1
                i <- i + 1
                Disk-Read($c_i$[x])
                if n[$c_i$[x]] = 2t - 1
                    then B-Tree-Split-Child(x, i, $c_i$[x])
                        if k > $key_i$[x]
                                then i <- i + 1
        B-Tree-Insert-Nonfull($c_i$[x], k)
```

To perform an insertion on a b-tree, the appropriate node for the key must be located using an algorithm similiar to *B-Tree-Search*. Next, the key must be inserted into the node. If the node is not full prior to the insertion, no special action is required; however, if the node is full, the node must be split to make room for the new key. Since splitting the node results in moving one key to the parent node, the parent node must not be full or another split operation is required. This process may repeat all the way up to the root and may require splitting the root node. This approach requires two passes. The first pass locates the node where the key should be inserted; the second pass performs any required splits on the ancestor nodes.

Since each access to a node may correspond to a costly disk access, it is desirable to avoid the second pass by ensuring that the parent node is never full. To accomplish this, the presented algorithm splits any full nodes encountered while descending the tree. Although this approach may result in unecessary split operations, it guarantees that the parent never needs to be split and eliminates the need for a second pass up the tree. Since a split runs in linear time, it has little effect on the $O(t \log_t n)$ running time of *B-Tree-Insert*.

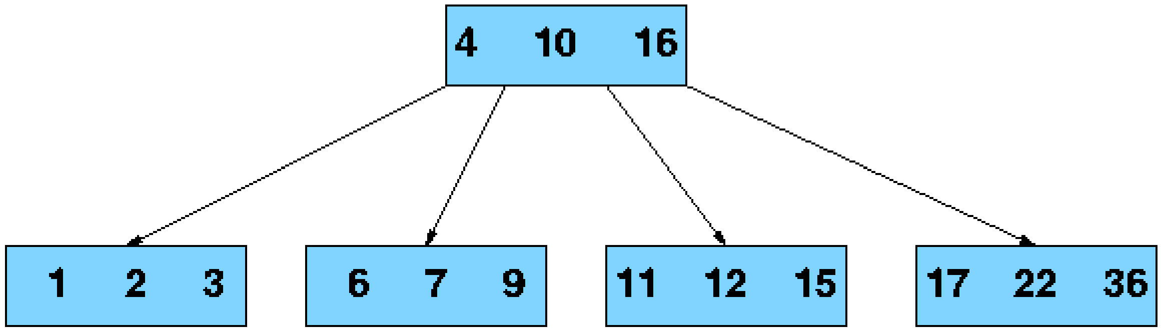
Splitting the root node is handled as a special case since a new root must be created to contain the median key of the old root. Observe that a b-tree will grow from the top.

B-Tree-Delete

Deletion of a key from a b-tree is possible; however, special care must be taken to ensure that the properties of a b-tree are maintained. Several cases must be considered. If the deletion reduces the number of keys in a node below the minimum degree of the tree, this violation must be corrected by combining several nodes and possibly reducing the height of the tree. If the key has children, the children must be rearranged. For a detailed discussion of deleting from a b-tree, refer to Section 19.3, pages 395-397, of Cormen, Leiserson, and Rivest or to another reference listed below.

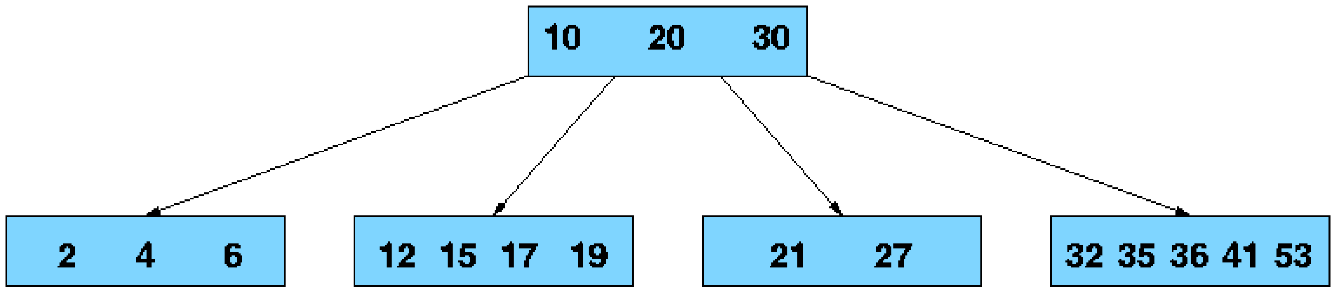
Examples

Sample B-Tree



Searching a B-Tree for Key 21

B-Tree: Minimization Factor t = 3, Minimum Degree = 2, Maximum Degree = 5



Search(21)

Inserting Key 33 into a B-Tree (w/ Split)

Applications

Databases

A database is a collection of data organized in a fashion that facilitates updating, retrieving, and managing the data. The data can consist of anything, including, but not limited to names, addresses, pictures, and numbers. Databases are commonplace and are used everyday. For example, an airline reservation system might maintain a database of available flights, customers, and tickets issued. A teacher might maintain a database of student names and grades.

Because computers excel at quickly and accurately manipulating, storing, and retrieving data, databases are often maintained electronically using a *database management system*. Database management systems are essential components of many everyday business operations. Database products like *Microsoft SQL Server*, *Sybase Adaptive Server*, *IBM DB2*, and *Oracle* serve as a foundation for accounting systems, inventory systems, medical recordkeeping sytems, airline reservation systems, and countless other important aspects of modern businesses.

It is not uncommon for a database to contain millions of records requiring many gigabytes of storage. For examples, TELSTRA, an Australian telecommunications company, maintains a customer billing database with 51 billion rows (yes, billion) and 4.2 terabytes of data. In order for a database to be useful and usable, it must support the desired operations, such as retrieval and storage, quickly. Because databases cannot typically be maintained entirely in memory, b-trees are often used to index the data and to provide fast access. For example, searching an unindexed and unsorted database containing n key values will have a worst case running time of $O(n)$; if the same data is indexed with a b-tree, the same search operation will run in $O(\log n)$. To perform a search for a single key on a set of one million keys (1,000,000), a linear search will require at most 1,000,000 comparisons. If the same data is indexed with a b-tree of minimum degree 10, 114 comparisons will be required in the worst case. Clearly, indexing large amounts of data can significantly improve search performance. Although other balanced tree structures can be used, a b-tree also optimizes costly disk accesses that are of concern when dealing with large data sets.

Concurrent Access to B-Trees

Databases typically run in multiuser environments where many users can concurrently perform operations on the database. Unfortunately, this common scenario introduces complications. For example, imagine a database storing bank account balances. Now assume that someone attempts to withdraw \$40 from an account containing \$60. First, the current balance is checked to ensure sufficent funds. After funds are disbursed, the balance of the account is reduced. This approach works flawlessly until concurrent transactions are considered. Suppose that another person simultaneously attempts to withdraw \$30 from the same account. At the same time the account balance is checked by the first person, the account balance is also retrieved for the second person. Since neither person is requesting more funds than are currently available, both requests are satisfied for a total of \$70. After the first person's transaction, \$20 should remain (\$60 - \$40), so the new balance is recorded as \$20. Next, the account balance after the second person's transaction, \$30 (\$60 - \$30), is recorded overwriting the \$20 balance. Unfortunately, \$70 have been disbursed, but the account balance has only been decreased by \$30. Clearly, this behavior is undesirable, and special precautions must be taken.

A b-tree suffers from similar problems in a multiuser environment. If two or more processes are manipulating the same tree, it is possible for the tree to become corrupt and result in data loss or errors.

The simplest solution is to serialize access to the data structure. In other words, if another process is using the tree, all other processes must wait. Although this is feasible in many cases, it can place an unecessary and costly limit on performance because many operations actually can be performed concurrently without risk. *Locking*, introduced by Gray and refined by many others, provides a mechanism for controlling concurrent operations on data structures in order to prevent undesirable side effects and to ensure consistency. For a detailed discussion of this and other concurrency control mechanisms, please refer to the references below.

Program 10:
Insertion into a AVL Tree and deletion from a AVL tree.

```
template<typename Key,typename Elements>
class AVLTree :public Binary Search Tree<Key,Element,AVL Item<Key,Element>>{protected:
          //local types
typedef AVLItem<Key,Element> Item;                                              // a tree node
item
typedef BinarysearchTree<Key,Element,Item>BST;                                  //base search
tree
typedef BST :: BTposition     BTposition;                                  //a tree position
public:                                                                    //public types
typedef BST::position                                                      //position
\\...(insert AVLItem here)
protected:                                                                 //local utilities
int height(const BTposition& p) const {                                         //get height of
p
  if(T.isExternal(p))return 0;
   else return p.elment().height();
}
void setHeight(BTPosition p)  {                                                 //set height of
p
int leftHeight  = height(T.leftChild(p));
int rightHeight=height(T.rightChild(p));
int maxHeight =max(leftHeight,rightHeight);
p.element().setHeight(1+maxHeight);
}
bool is Balanced(const BTPosition&  P) const {.                                 //is p
balanced?
if bf=height(T.leftchild(p))-height(T.rightchild(p));
return((-1<=bf)&&(bf<=1));
}
BTPosition tallGrandchild(const BTPosition& p)const;                                 //get
tallest grandchild
//...(insert rebalance() here)
public:
AVLTree()  : BST()  {  }                                                   //constructor
void insertItem(const key&  k,const Element & e)  {
          //insert(key,element)
BTposition  p = inserter(k ,e);                                            //insert in base tree
setHeight(p);                                                              //compute its height
rebalance(p);                                                              //rebalance if needed
}
 void removeElement(const Key&   k)                                             //remove using
key
  throw(NonexistentElementException)  {
BTposition p  = finder(k,T.root());                                        //find in base tree
if(p.isNull())                                                             //not found?
throw NonexistentElementException("Remove nonexistent element");
```

```
BTPosition  r =  remover(p);                                    //remove it
rebalance(r);                                               //rebalance if needed
}};

11> AVL
#include<iostream.h>
#include<conio.h>
#include<process.h>
class AVL
{
private:
        struct node
        {
        int data;
        int height;
        struct node *left;
        struct node *right;
        };
        struct node *p;
        public:
          AVL()
          {
          p=NULL;
          }
          void insert(int,struct node &);
          void del(int,struct node &);
          void find(int,struct node &);
         // void preorder(struct node);
          //void inorder(struct node);
          //void postorder(struct node);
          //void count(struct node);
          int AVLheight(struct node);
          int max(int,int);
          struct node single_left_rotation(struct node &);
          struct node single_right_rotation(struct node &);
          struct node double_left_rotation(struct node &);
          struct node double_ right_rotation(struct node &);
          };
          int AVL::max(int a,int b)
          {
          if(a>b)
          return a;
          else
          return b;
          }
          int AVL::AVLheight(struct node p)
          {
          int t;
          if(p==NULL)
          {
          return-1;
          }
          else
          {
          t=p->height;
          return t;
          }
          }
          void AVL::insert(int e,struct node &)
          {
          if(p==NULL)
          {
          p=new node;
          p->data=e;
          p->height=0;
          p->left=NULL;
          p->right=NULL;
          }
          else
          {
          if(p->data>e)
          insert(e,p->left);
          if((AVLheight(p->left)-AVLheight(p->right)==2||-2))
          {
           if(p->left->data>e)
           p=singlerotationleft(p);
           else
           p=doublerotationleft(p);
           }
           else if(p->data<e)
           insert(e,p->right);
           if((AVLheight(p->left)-(AVLheight(p->right)==2||-2))
           {
            if(p->right->data<e)
          p=singlerotationright(p);
           else
              p=doublerotationright(p);
         }
         else
        {
        cout<<"\n duplicate ele";
}
}

 int m,n,d;
 m=AVLheight(p->left);
 n=AVLheight(p->right);
d=max(m,n);
p->height=d+1;
}
```

```
 void AVL::find(int e,struct node &p)
{
 if(p==NULL)
cout<<"\n ele is not found";
else if(p->data>e)
find(e,p->left);
else if(p->data<e)
find(e,p->right);
else
cout<<"\n ele is found";
}

 void AVL::del(int e,struct node &p)
{
 if(p==NULL)
{
 cout<<"\n ele not found";
}
else if(p->data>e)
del(e,p->left);
else if(p->data<e)
find(e,p->right);
else if((p->left)==NULL&&(p->right)==NULL)
{
temp=p;
p=NULL;
delete temp;
cout<<"\n ele is deleted";
}
 else if(p->left==NULL)
{
 temp=p;
p=p->right;
delete temp;
cout<<"\n ele deleted";
}
else if(r->right==NULL)
{
 temp=p;
p=p->left;
delete temp;
cout<<"\n ele delete";
}
else
 p->data=minright(p->right);
}

struct node AVL::single leftrotation(structnode &p1)
{
 struct node *p2;
p2=p1->left;
p1->left=p2->right;
p2->right=p1;
p1->height=max(AVLheight(p1->left),AVLheight(p->right))+1;
p2->height=max(AVLheight(p2->left),p2->height))+1;
return p2;
}

struct node AVL::single right rotation(struct node &p1)
{
 struct node *p2;
p2=p1->right;
p1->right=p2->left;
p2->left=p1;
p1->height=max(AVLheight(p1->left),AVLheight(p1->right))+1;
p2->height=max(p2->height,AVLheight(p2->right))+1;
return p2;
}

struct node AVL::double rotationleft(struct node &p1)
{
 p1->left=single rotation right(p1->left);
 return single rotation right left(p1);
}

struct node AVL::double rotationright(struct node &p1)
{
 p1->left=single rotation left(p1->right);
 return single rotation right right(p1);
}


 void main()
{
 AVL a1;
int ch,ele;
 do
 {
 cout<<"\n1.insert:";
 cout<<"\n2.del:";
cout<<"\n3.find:";
cout<<"\n4.AVLheight:";
cout<<"\n5.max:";
cout<<"\n6.single left rotation:";
cout<<"\n7.single right rotation:";
cout<<"\n8.double left rotation:";
cout<<"\n9.double right rotation:";
cout<<"\n10.exit:";
```

```
 cout<<"\n enter choice:";
 cin>>ch;
switch(ch)
{
 case 1: cout<<"\n element to insert:";
         cin>>ele;
             a1.insert(ele,p);
            break:
case 2: cout<<"\n element to delete:";
         cin>>ele;
             a1.delsert(ele,p);
            break:
case 3: cout<<"\n element to find:";
         cin>>ele;
             a1.find(ele,p);
            break:
case 4:  a1.AVLheight(p);
            break:
case 5:  a1.max(a,b);
            break:
case 6:  a1.single left rotation(p);
            break:
case 7:  a1.single right rotation(p);
            break:
case 8:  a1.double left rotation(p);
            break:
case 9: a1.double right rotation(p);
            break:
case 10: a1.exit(0);
            break:
}
}while(ch<=10);
getch();
}
```

Program 11:

Implement Kruskals algorithm to generate a minimum spanning tree.

More Graph Problems

Cycle Detection Algorithms

Many algorithms rely on detecting cycles in graphs. Many cycle detection algorithms are "brute force" algorithms and are quite inefficient. However, there are several algorithms which are quite efficient. One such algorithm is based upon a depth-first traversal of the graph. For undirected graphs, a single line needs to be added to the algorithm we presented earlier for depth-first traversal. This algorithm as well as the "modified" version are shown below:

Original Algorithm:

```
DFS(v)
    num(v) = i++;
    for all vertices u adjacent to v
        if num(u) is 0
            attach edge (uv) to edges;
            DFS(u);

depthFirstSearch( )
    for all vertices v
        num(v) = 0;
    edges = null;
    i = 1;
    while there is a vertex v such that num(v) is 0
        DFS(v);
    output edges;
```

Modified Algorithm:

```
cycleDetection (v)
    num(v) = i++;
    for all vertices u adjacent to v
        if num(u) is 0
            attach edge (uv) to edges;
            cycleDetection(u);
        else cycle detected;
```

For digraphs, the situation is a bit more complicated, since there may be edges between different spanning subtrees, called *side edges*. An edge (a back edge) indicates a cycle if it joins two vertices already included in the same spanning subtree. To consider only this case, a number higher than any number generated in subsequent searches is assigned to a vertex being currently visited after all its descendants have also been visited. In this way, if a vertex is about to be joined by an edge with a vertex having a lower number, then a cycle has been detected. The algorithm for cycle detection using this technique in a digraph is shown below:

```
digraphCycleDetection (v)
    num(v) = i++;
    for all vertices u adjacent to v
        if num(u) is 0
            attach edge(uv) to edges;
            digraphCycleDetection(u);
        else if num(u) is not ∞
            cycle is detected;
    num(v) = ∞;
```

Kruskal’s Algorithm to Generate a Minimum Spanning Tree

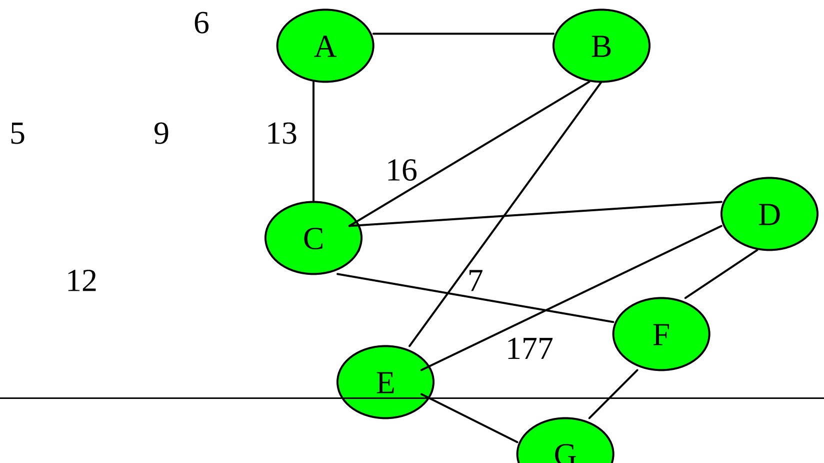
We have already seen Prim’s algorithm for generating a minimum spanning tree. Prim’s technique, although we presented it in tabular form, basically creates a single tree and expands the tree from the root as edges are considered. Kruskal’s algorithm takes a different approach in which a set of trees (a forest) is condensed to a single tree.

In Kruskal’s method, all edges are ordered by weight, and then each edge in this ordered sequence is checked to see whether it can be considered as part of the tree which is under construction. The edge is added to the tree only if no cycle arises after its inclusion. Kruskal’s algorithm is quite simple and is shown below:

```
KruskalAlgorithm (weighted connected undirected graph)
    tree = null;
    edges = sequence of all edges of graph sorted by weight;
    for (i = 1; i ≤ |E| and |tree| < |V|-1; i++)
        if e_i from edges does not form a cycle with edges in tree
            add e_i to tree;
```

The complexity of this algorithm is determined by the complexity of the sorting algorithm which is applied, which for an efficient sorting algorithm is $O(|E|\log_2 |E|)$. It also depends on the complexity of the algorithm used for the cycle detection.

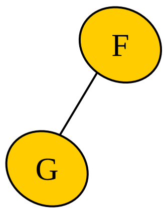
To illustrate the technique of Kruskal’s algorithm, consider the following example:



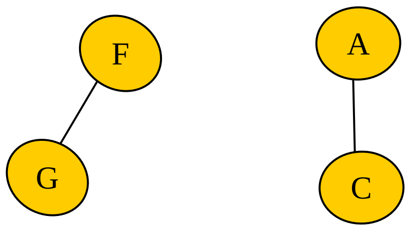
8 3

The ordering of the weighted edges is:
(g,f) = 3, (a,c) = 5, (a,b) = 6, (d,f) = 7, (e,g) = 8, (c,b) = 9, (c,f) = 12, (b,e)= 13, (d,e) = 15, and (c,d) = 16

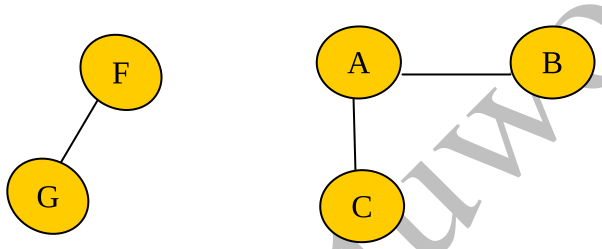
Iteration 1: A tree is formed from the minimum weight edge (g,f)



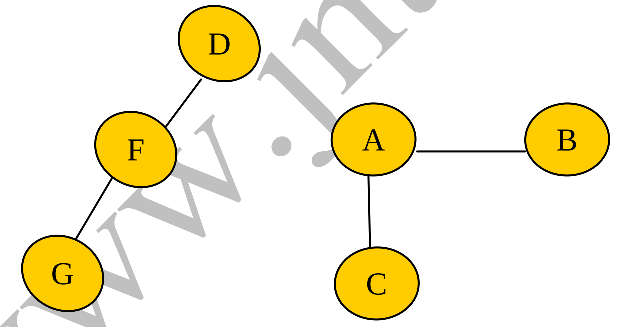
Iteration 2: A second tree is formed from the minimum weight edge (a,c). Notice that this tree is not connected to the first tree since they have no vertex in common.



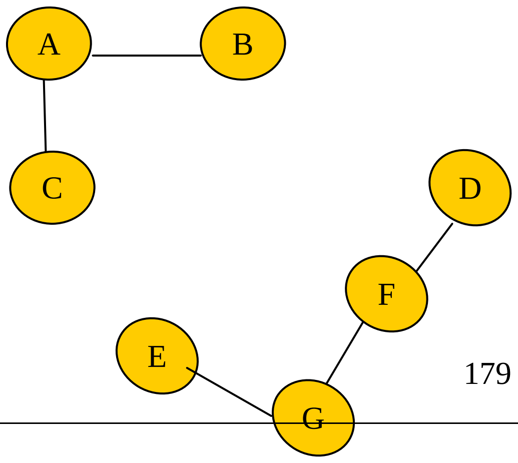
Iteration 3: The next minimum edge (a,b) is added to the forest, this time to an existing tree since there is a common vertex in *a*.



Iteration 4: The next edge added to the tree (forest) is, (d,f).

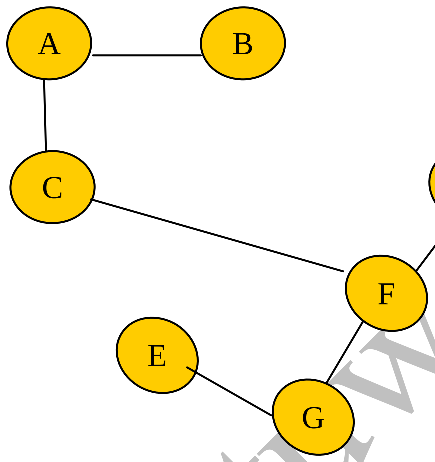


teration 5: The next edge added is, (e,g).



Iteration 6: This step will attempt to add the edge (c,b) but this would induce a cycle so the edge is not added to the tree.

Iteration 7: This step will add the edge (c,f).



Iteration 8: This step will attempt to add the edge (b,e) but this would induce a cycle so the edge is not added to the tree.

Iteration 9: This step will attempt to add the edge (d,e) but this would induce a cycle so the edge is not added to the tree.

Iteration 10: This step will attempt to add the edge (c,d) but this would induce a cycle so the edge is not added to the tree.

Thus the final minimum spanning tree is shown after iteration #7 has completed.

For practice, you should run Prim's algorithm on the initial graph for this example. Does Prim's algorithm produce the same minimum spanning tree? {Answer is on the last page of this set of notes.}

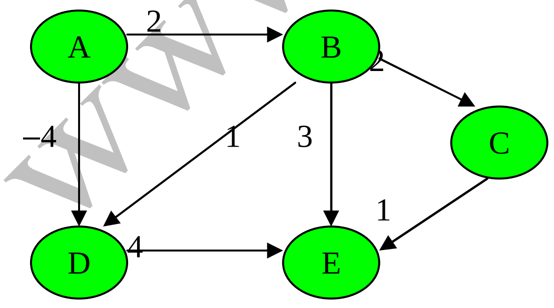
All-to-All Shortest Path Problem

Dijkstra's and Ford's algorithms solve the shortest path problem from one specified vertex to all other vertices in the graph. This type of problem is often called the One-to-All Shortest Path problem. The problem of finding all shortest paths from any vertex to any other vertex (the All-to-All Shortest Path problem) seems to be a more complicated problem. However, an algorithm developed by Stephen Warshall and implemented by Robert Floyd and P.Z. Ingerman solves this problem in a surprisingly simple way provided that the adjacency matrix indicates the weight of each edge in the graph. The technique works whether the graph is undirected or directed and the graph may include negative weights. The algorithm is shown below:

```
WFIalgorithm (matrix weights)
        for i = 1  to  |V|
            for j = 1 to |V|
                for k = 1 to |V|
                    if weight[j][k] > weight[j][i] + weight[i][k]
                        weight[j][k] = weight[j][i] + weight[i][k];
```

The outermost loop handles the vertices which may be on a path between the vertex with index *j* and the vertex with index *k*. For example, in the first iteration, when *i* = 1, all paths $v_i \ldots v_l \ldots v_k$ are considered, and if there is currently no path from v_j to v_k and v_k is reachable from v_j, the path is established, with its weight equal to $p = weight(path(v_j,\ldots,v_l)) + weight(path(v_l,\ldots v_k))$, or the current weight of this path, $weight(path(v_j,\ldots v_k))$ is changed to *p* if *p* is less than $weight(path(v_j,\ldots,v_k))$. To illustrate the WFI algorithm consider the following example:

	A(1)	B(2)	C(3)	D(4)	E(5)
A(1)	0	2	∞	–4	∞
B(2)	∞	0	–2	1	3
C(3)	∞	∞	0	∞	1
D(4)	∞	∞	∞	0	4
E(5)	∞	∞	∞	∞	0

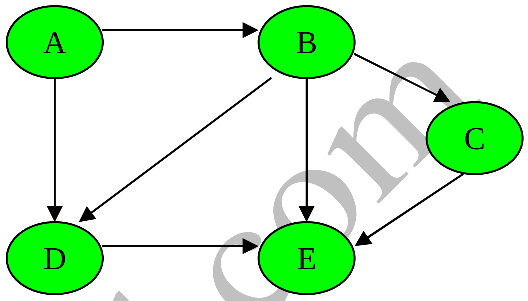


Since the graph in the example is a directed graph, notice that the matrix is a diagonal matrix. In this case only the cells in the upper right side of the main diagonal contain data which describes the graph. The cells in the lower left side of the diagonal all contain infinity. The cells along the main diagonal are initialized to 0. After examining how the WFI algorithm operates, we'll come back to explore the adjacency matrix a bit more as there turns out to be a very useful purpose to representing the graph in this fashion.

Iteration 1 (variable *i* refers to vertex *A*)

The test which is performed in the algorithm is: weight[j][k] > weight[j][i] + weight[i][k]. Vertex *A* has no incident edges (for all *j* and *k* values there are no values for weight[j][1] or weight[1][k]) so no changes will occur to the matrix during the first iteration of the algorithm. Since *A* has no incident edges, it cannot be along the path between any two vertices *j* and *k*.

	A(1)	B(2)	C(3)	D(4)	E(5)
A(1)	0	2	∞	–4	∞
B(2)	∞	0	–2	1	3
C(3)	∞	∞	0	∞	1
D(4)	∞	∞	∞	0	4
E(5)	∞	∞	∞	∞	0

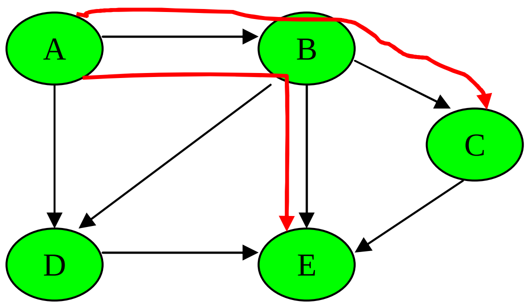


Iteration 2 (variable *i* refers to vertex *B*)

During this iteration, *i* is 2 so the test becomes: weight[j][k] > weight[j][2] + weight[2][k]. Vertex *B* has incident 1 edge; the following tests will be performed: weight[j][k] > weight[j][2] + weight[2][k]. Since *B* has only one incident edge (from *A*) it can only be along a path which begins at *A* (since *A* has no incident edges). Shorter paths found in this iteration are shown in red.

weight[1][1] > weight[1][2] + weight[2][1] - no, no changes
weight[1][2] > weight[1][2] + weight[2][2] – no, no changes
weight[1][3] > weight[1][2] + weight[2][3] – yes, ∞ > 0, set weight[1][3] to 0
weight[1][4] > weight[1][2] + weight[2][4] – no, no changes
weight[1][5] > weight[1][2] + weight[2][5] – yes, ∞ > 5, set weight[1][5] to 5

	A	B	C	D	E
A	0	2	0	–4	5
B	∞	0	–2	1	3
C	∞	∞	0	∞	1
D	∞	∞	∞	0	4
E	∞	∞	∞	∞	0

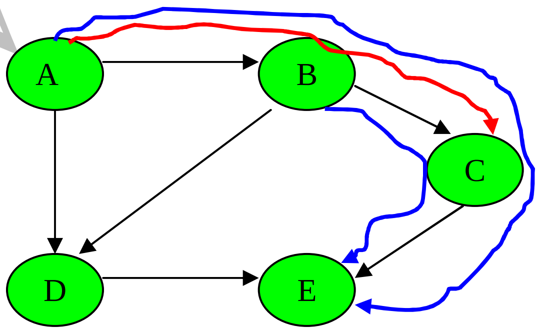


<u>Iteration 3 (variable *i* refers to vertex *C*)</u>

During this iteration, *i* is 3 so the test becomes: weight[j][k] > weight[j][3] + weight[3][k]. Vertex *C* has 1 incident edge; the following tests will be performed: weight[j][k] > weight[j][3] + weight[3][k]. Shorter paths found in this iteration are shown in blue.

weight[1][1] > weight[1][3] + weight[3][1] – no, no changes
weight[1][2] > weight[1][3] + weight[3][2] – no, no changes
weight[1][3] > weight[1][3] + weight[3][3] – no, no changes
weight[1][4] > weight[1][3] + weight[3][4] – no, no changes
weight[1][5] > weight[1][3] + weight[3][5] – yes, 5 > 1, set weight[1,5] to 1
weight[2][1] > weight[2][3] + weight[3][1] – no, no changes
weight[2][2] > weight[2][3] + weight[3][2] – no, no changes
weight[2][3] > weight[2][3] + weight[3][3] – no, no changes
weight[2][4] > weight[2][3] + weight[3][4] – no, no changes
weight[2][5] > weight[2][3] + weight[3][5] – yes, 3 > –1, set weight[3][5] to –1

	A	B	C	D	E
A	0	2	0	–4	1
B	∞	0	–2	1	–1
C	∞	∞	0	∞	1
D	∞	∞	∞	0	4
E	∞	∞	∞	∞	0



<u>Iteration 4 (variable *i* refers to vertex *D*)</u>

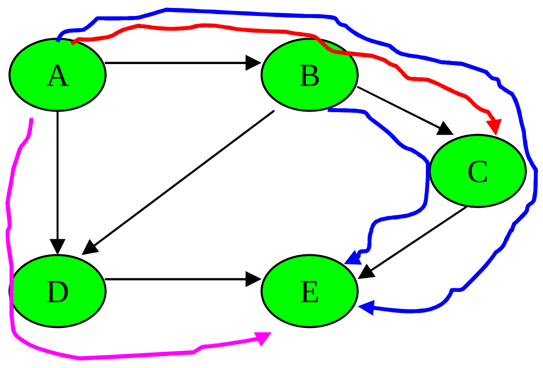
During this iteration, *i* is 4 so the test becomes: weight[j][k] > weight[j][4] + weight[4][k]. Vertex *D* has 2 incident edges; the following tests will be performed: weight[j][k] > weight[j][4] + weight[4][k]. Shorter paths found in this iteration are shown in purple.

weight[1][1] > weight[1][4] + weight[4][1] - , no, no changes
weight[1][2] > weight[1][4] + weight[4][2] – no, nochanges
weight[1][3] > weight[1][4] + weight[4][3] – no, no changes

weight[1][4] > weight[1][4] + weight[4][4] – no, no changes
weight[1][5] > weight[1][4] + weight[4][5] – yes, 1>0, set weight[1][5] to 0
weight[2][1] > weight[2][4] + weight[4][1] - , no, no changes
weight[2][2] > weight[2][4] + weight[4][2] – no, nochanges
weight[2][3] > weight[2][4] + weight[4][3] – no, no changes
weight[2][4] > weight[2][4] + weight[4][4] – no, no changes
weight[2][5] > weight[2][4] + weight[4][5] – no, no changes
weight[3][1] > weight[3][4] + weight[4][1] - , no, no changes
weight[3][2] > weight[3][4] + weight[4][2] – no, nochanges
weight[3][3] > weight[3][4] + weight[4][3] – no, no changes
weight[3][4] > weight[3][4] + weight[4][4] – no, no changes
weight[3][5] > weight[3][4] + weight[4][5] – no, no changes
weight[4][1] > weight[4][4] + weight[4][1] - , no, no changes
weight[4][2] > weight[4][4] + weight[4][2] – no, nochanges
weight[4][3] > weight[4][4] + weight[4][3] – no, no changes
weight[4][4] > weight[4][4] + weight[4][4] – no, no changes
weight[4][5] > weight[4][4] + weight[4][5] – no, no changes
weight[5][1] > weight[5][4] + weight[4][1] - , no, no changes
weight[5][2] > weight[5][4] + weight[4][2] – no, nochanges
weight[5][3] > weight[5][4] + weight[4][3] – no, no changes

weight[5][4] > weight[5][4] + weight[4][4] – no, no changes
weight[5][5] > weight[5][4] + weight[4][5] – no, no changes

	A	B	C	D	E
A	0	2	0	–4	0
B	∞	0	–2	1	–1
C	∞	∞	0	∞	1
D	∞	∞	∞	0	4
E	∞	∞	∞	∞	0

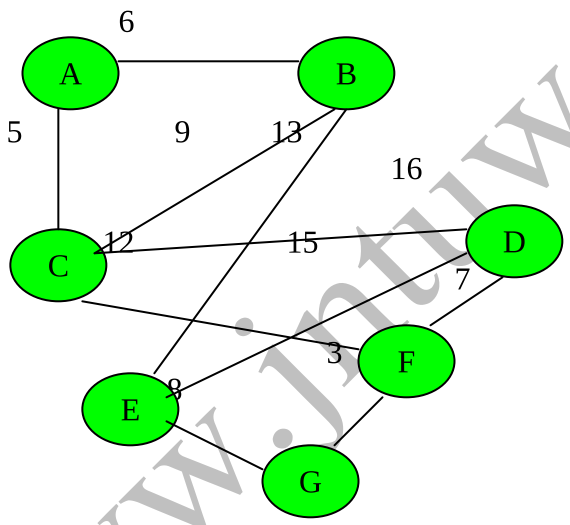


Iteration 5 (variable *i* refers to vertex *E*)

As with the first iteration of the algorithm, the last iteration will cause no changes to the adjacency matrix because vertex *E* has no edges which emanate from it. Therefore, it cannot be along the path between any other two vertices in the graph. So our work is done and the adjacency matrix contains the values of the shortest paths between any two arbitrary vertices in the graph.

The WFIalgorithm for solving the all-to-all shortest path problems also allows for the detection of cycles in the graph. To achieve this additional functionality for the algorithm, the weights along the main diagonal must be initialized to ∞ rather than 0. Through the course of execution of the algorithm on a particular graph, if any of the values along the main diagonal are changed, the graph will contain a cycle. Futher, if one of the initial values of ∞ between two vertices in the adjacency matrix is not changed to a finite value during the execution of the algorithm this is an indication that a vertex is unreachable from another.

Answer for Prim's algorithm using the example for Kruskal's algorithm



vertex	visited	minimum weight	vertex causing change to min weight
A	F	0	0
B	F	∞	0
C	F	∞	0
D	F	∞	0
E	F	∞	0
F	F	∞	0
G	F	∞	0

Initial table

vertex	visited	minimum weight	vertex causing change to min weight
A	T	0	
B	F	6	A
C	F	5	A
D	F	∞	0
E	F	∞	0
F	F	∞	0
G	F	∞	0

After first iteration – active vertex was A

vertex	visited	minimum weight	vertex causing change to min weight
A	T	0	0
B	F	6	A
C	T	5	A
D	F	16	C
E	F	∞	0
F	F	12	C
G	F	∞	0

After second iteration – active vertex was C

vertex	visited	minimum weight	vertex causing change to min weight
A	T	0	0
B	T	6	A
C	T	5	A
D	F	16	C
E	F	13	B
F	F	12	C
G	F	∞	0

After third iteration – active vertex was B

vertex	visited	minimum weight	vertex causing change to min weight
A	T	0	0
B	T	6	A
C	T	5	A
D	F	7	F
E	F	13	B
F	T	12	C
G	F	3	F

After fourth iteration – active vertex was F

vertex	visited	minimum weight	vertex causing change to min weight
A	T	0	0
B	T	6	A
C	T	5	A
D	F	7	F
E	F	8	G
F	T	12	C
G	T	3	F

After fifth iteration – active vertex was G

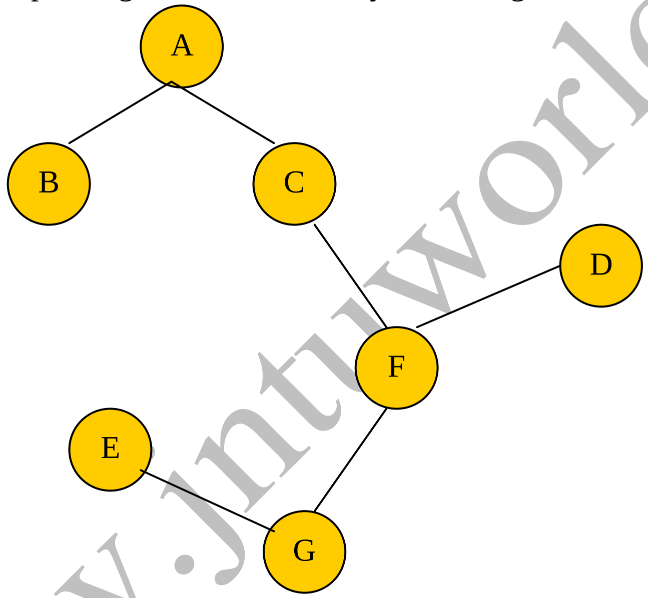
vertex	visited	minimum weight	vertex causing change to min weight
A	T	0	0
B	T	6	A
C	T	5	A
D	T	7	F
E	F	8	G
F	T	12	C
G	T	3	F

After sixth iteration – active vertex was D

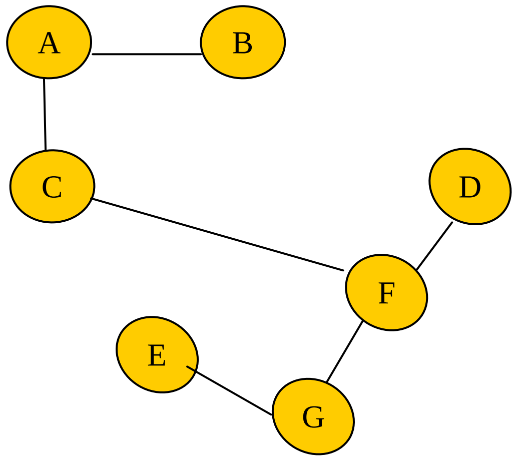
vertex	**visited**	**minimum weight**	**vertex causing change to min weight**
A	T	0	0
B	T	6	A
C	T	5	A
D	T	7	F
E	F	8	G
F	T	12	C
G	T	3	F

After seventh and final iteration – active vertex was E

The minimum spanning tree constructed by Prim's algorithm is shown below:



The minimum spanning tree constructed by Kruskal's algorithm:



Yes, both algorithms generate the same minimum spanning tree. Reason… this tree has only one minimum spanning tree!

```
        void  graph : : kruskal( )

             {

                       int edgesAccepted;

                       disjSet s (NUM_VERTICES );

                       priorityQueue h(NUM_EDGES );

                       vertex u, v;

                       settype uset, vset;

                       Edge e;

             /*1*/              h= readGraphintoHeapArray( );

             /*2*/              h.buildHeap( );

             /*3*/              edgesAccepted =0;

             /*4*/              while(edgesAccepted < NUM_VERTICES -1)

                       {

             /*5*/                       h.deleteMin(e);               //edge e = (u,v)

             /*6*/                       uset = s.find( u);

             /*7*/                       vst = s.find( v);

             /*8*/                       if(uset !=vset)

                                {

                                //Accept the edge

             /*9*/                       edgesAccepted++;

             /*10*/                      s.unionSets( uset, vset);

                                }}}
```

12. Program to implement Hashing.(Linear Probing).

```
#include<iostream.h>
#include<conio.h>
#include<process.h>

class hash
{
private:
            int *a;
            int div;
public:
            hash();
            void insert(int);
            void delet(int);
            void disp();
            int find(int);
            int full();
};


hash::hash()
{
            int i;
            clrscr();
            cout<<"\n enter the devisor:";
            cin>>div;
            a=new int[div];
            for(i=0;i<div;i++)
            a[i]=-1;
}


int hash::full()
{
  for(int i=0;i<div;i++)
  {
  if(a[i]==-1)return 0;
  }
return 1;
}

void hash::disp()
{
            int i;
            for(i=0;i<div;i++)
            {
             if(a[i]==-1)
             {

             }
             else
             {
             cout<<a[i]<<endl;
             }
       }
}

void hash::insert(int x)
{
            int i,j,k;
            i=find(x);
            if(this->full())
            {
            cout<<"Hash table is full,we cannot insert";
            return;
            }

            else if(i>=0)
            {
            cout<<"\n It is a duplicate element\n";
            return;
            }

            else
             {
             k=x%div;
               for(j=0;j<div;j++)
               {
             if(a[k]==-1)
             {
             a[k]=x;
             cout<<"Element inserted"<<endl;
             return;
             }
             else
             {

             if(k==(div-1))
             k=0;
             else
             k++;
             }
               }
             }

}
void hash::delet(int x)
{
            int i;
            i=find(x);
            if(i==-1)
            {
            cout<<"\nElement not found\n";
            return;
            }
            else
            {
            a[i]=-1;
            cout<<"Element deleted";
            }

}


int hash::find(int x)
{
int i,j;
i=x%div;
for(j=0;j<div;j++)
{
if(a[i]==x)
return i;
  else
  {
  if(i==(div-1))i=0;
  else i++;
  }
}
return -1;


}


void main()
{
       hash h;
```

```
int ch,ele,y;
clrscr();
do
{
cout<<"\n1.insert:";
cout<<"\n2.del:";
cout<<"\n3.find:";
cout<<"\n4.disp:";
cout<<"\n5.exit:";
cout<<"\nEnter the choice:";
cin>>ch;
switch(ch)
{
    case 1:cout<<"\nEnter the element:";
             cin>>ele;
         h.insert(ele);
             break;
    case 2:cout<<"\nEnter the element to delete:";
             cin>>ele;
         h.delet(ele);
             break;
    case 3:cout<<"\nEnter the element to find:";
             cin>>ele;
         y=h.find(ele);
         if(y==-1)
         {
         cout<<"Element not found"<<endl;
         }
         else
         {
         cout<<"Element found at"<<y+1<<"Position"<<endl;
         }
         break;

    case 4: cout<<"\nThe elements are:";
             h.disp();
             break;
    case 5:exit(0);
             break;
}
}while(ch<=5);
getch();
}
```

VIVA QUESTIONS

What is C++?

Released in 1985, C++ is an object-oriented programming language created by Bjarne Stroustrup. C++ maintains almost all aspects of the C language, while simplifying memory management and adding several features - including a new datatype known as a class (you will learn more about these later) - to allow object-oriented programming. C++ maintains the features of C which allowed for low-level memory access but also gives the programmer new tools to simplify memory management.

C++ used for:

C++ is a powerful general-purpose programming language. It can be used to create small programs or large applications. It can be used to make CGI scripts or console-only DOS programs. C++ allows you to create programs to do almost anything you need to do. The creator of C++, Bjarne Stroustrup, has put together a partial list of applications written in C++.

How do you find out if a linked-list has an end? (i.e. the list is not a cycle)

You can find out by using 2 pointers. One of them goes 2 nodes each time. The second one goes at 1 nodes each time. If there is a cycle, the one that goes 2 nodes each time will eventually meet the one that goes slower. If that is the case, then you will know the linked-list is a cycle.

What is the difference between realloc() and free()?

The free subroutine frees a block of memory previously allocated by the malloc subroutine. Undefined results occur if the Pointer parameter is not a valid pointer. If the Pointer parameter is a null value, no action will occur. The realloc subroutine changes the size of the block of memory pointed to by the Pointer parameter to the number of bytes specified by the Size parameter and returns a new pointer to the block. The pointer specified by the Pointer parameter must have been created with the malloc, calloc, or realloc subroutines and not been deallocated with the free or realloc subroutines. Undefined results occur if the Pointer parameter is not a valid pointer.

What is function overloading and operator overloading?

Function overloading: C++ enables several functions of the same name to be defined, as long as these functions have different sets of parameters (at least as far as their types are concerned). This capability is called function overloading. When an overloaded function is called, the C++ compiler selects the proper function by examining the number, types and order of the arguments in the call. Function overloading is commonly used to create several functions of the same name that perform similar tasks but on different data types.
Operator overloading allows existing C++ operators to be redefined so that they work on objects of user-defined classes. Overloaded operators are syntactic sugar for equivalent function calls. They form a pleasant facade that doesn't add anything fundamental to the language (but they can improve understandability

and reduce maintenance costs).

What is data structure?
A data structure is a way of organizing data that considers not only the items stored, but also their relationship to each other. Advance knowledge about the relationship between data items allows designing of efficient algorithms for the manipulation of data.

List out the areas in which data structures are applied extensively?
Compiler Design, Operating System, Database Management System, Statistical analysis package, Numerical Analysis, Graphics, Artificial Intelligence, Simulation

If you are using C language to implement the heterogeneous linked list, what pointer type will you use?
The heterogeneous linked list contains different data types in its nodes and we need a link, pointer to connect them. It is not possible to use ordinary pointers for this. So we go for void pointer. Void pointer is capable of storing pointer to any type as it is a generic pointer type.

What is the data structures used to perform recursion?
Stack. Because of its LIFO (Last In First Out) property it remembers its caller, so knows whom to return when the function has to return. Recursion makes use of system stack for storing the return addresses of the function calls. Every recursive function has its equivalent iterative (non-recursive) function. Even when such equivalent iterative procedures are written, explicit stack is to be used.

What are the methods available in storing sequential files ?
Straight merging, Natural merging, Polyphase sort, Distribution of Initial runs.

List out few of the Application of tree data-structure?
The manipulation of Arithmetic expression, Symbol Table construction, Syntax analysis.

In RDBMS, what is the efficient data structure used in the internal storage representation?
B+ tree. Because in B+ tree, all the data is stored only in leaf nodes, that makes searching easier. This corresponds to the records that shall be stored in leaf nodes.

What is a spanning Tree?
A spanning tree is a tree associated with a network. All the nodes of the graph appear on the tree once. A minimum spanning tree is a spanning tree organized so that the total edge weight between nodes is minimized.

what is the difference b/w abstract and interface?
Abstract   can,t support multiple inheritence.
Interface can suppport the multiple inheritence.

Abstract   have accesbulity modifiers.

Interface have no accesbulity modifiers

how memory store byte

byte mean binary digit for 8 digits .
 it meant 1 byte store 8 bits .

1bit =1024kb

HOW TO SWAP TWO NOS IN ONE STEP?

```
main()
{
int a,b,c;
printf("enter two no's :");
scanf("%d%d",&a,&b);
c=a^=b^=a^=b;
printf("%d",c);
}
```

What is the Advantage of Interface over the Inheritance in OOPS?

Provides flexibility in implementing the operations for the derived classes.

2. Avoid conflicts when more than one interfaces are derived in a class.

write a c++ program to find maximum of two numbers using

inline functions

```
#include<iostream>
using namespace std;
int main()
{
   int c;
   c=max(5,4);        //will display 5
   cout<<c<<endl;
   return 0;
}
inline int max(int a, int b)
{
   return (a>b)? a:b;
}
```

What is the difference between declaration and definition?

The declaration tells the compiler that at some later point we plan to present the definition of this declaration.
E.g.: void stars () //function declaration
The definition contains the actual implementation.
E.g.: void stars () // declarator
{
for(int j=10; j > =0; j--) //function body
cout << *;
cout << endl; }

What are the advantages of inheritance?

It permits code reusability. Reusability saves time in program development. It encourages the reuse of proven and debugged high-quality software, thus reducing problem after a system becomes functional.

How do you write a function that can reverse a linked-list?

```
void reverselist(void)
{
if(head==0)
return;
if(head->next==0)
return;
if(head->next==tail)
{
head->next = 0;
tail->next = head;
}
else
{
node* pre = head;
node* cur = head->next;
node* curnext = cur->next;
head->next = 0;
cur-> next = head;

for(; curnext!=0; )
{
cur->next = pre;
pre = cur;
cur = curnext;
curnext = curnext->next;
}

curnext->next = cur;
}
}
```

What do you mean by inline function?
The idea behind inline functions is to insert the code of a called function at the point where the function is called. If done carefully, this can improve the application's performance in exchange for increased compile time and possibly (but not always) an increase in the size of the generated binary executables.

Write a program that ask for user input from 5 to 9 then calculate the average

```
#include "iostream.h"
int main() {
int MAX = 4;
int total = 0;
int average;
int numb;
for (int i=0; i<MAX; i++) {
cout << "Please enter your input between 5 and 9: ";
cin >> numb;
while ( numb<5 || numb>9) {
cout << "Invalid input, please re-enter: ";
cin >> numb;
}
total = total + numb;
}
average = total/MAX;
cout << "The average number is: " << average << "\n";
return 0;
}
```

Write a short code using C++ to print out all odd number from 1 to 100 using a for loop

```
for( unsigned int i = 1; i < = 100; i++ )
if( i & 0x00000001 )
cout << i << \",\";
```

What is public, protected, private?
Public, protected and private are three access specifier in C++.
Public data members and member functions are accessible outside the class.
Protected data members and member functions are only available to derived classes.
Private data members and member functions can’t be accessed outside the class. However there is an exception can be using friend classes.
Write a function that swaps the values of two integers, using int* as the argument type.

```
void swap(int* a, int*b) {
int t;
t = *a;
*a = *b;
*b = t;
}
```

Tell how to check whether a linked list is circular.
Create two pointers, each set to the start of the list. Update each as follows:

```
while (pointer1) {
pointer1 = pointer1->next;
pointer2 = pointer2->next; if (pointer2) pointer2=pointer2->next;
if (pointer1 == pointer2) {
print (\"circular\n\");
}
}
```

OK, why does this work?
If a list is circular, at some point pointer2 will wrap around and be either at the item just before pointer1, or the item before that. Either way, it’s either 1 or 2 jumps until they meet.

What is virtual constructors/destructors?
Answer1
Virtual destructors:
If an object (with a non-virtual destructor) is destroyed explicitly by applying the delete operator to a base-class pointer to the object, the base-class destructor function (matching the pointer type) is called on the object.
There is a simple solution to this problem declare a virtual base-class destructor. This makes all derived-class destructors virtual even though they don’t have the same name as the base-class destructor. Now, if the object in the hierarchy is destroyed explicitly by applying the delete operator to a base-class pointer to a derived-class object, the destructor for the appropriate class is called. Virtual constructor: Constructors cannot be virtual. Declaring a constructor as a virtual function is a syntax error.

Answer2
Virtual destructors: If an object (with a non-virtual destructor) is destroyed explicitly by applying the delete operator to a base-class pointer to the object, the base-class destructor function (matching the pointer type) is called on the object.
There is a simple solution to this problem – declare a virtual base-class destructor. This makes all derived-class destructors virtual even though they don’t have the same name as the base-class destructor. Now, if the object in the hierarchy is destroyed explicitly by applying the delete operator to a base-class pointer to a derived-class object, the destructor for the appropriate class is called.

Virtual constructor: Constructors cannot be virtual. Declaring a constructor as a virtual function is a syntax error. Does c++ support multilevel and multiple inheritance?
Yes.

What are the advantages of inheritance?
• It permits code reusability.
• Reusability saves time in program development.
• It encourages the reuse of proven and debugged high-quality software, thus reducing problem after a system becomes functional.

What is the difference between declaration and definition?
The declaration tells the compiler that at some later point we plan to present the

definition of this declaration.
E.g.: void stars () //function declaration
The definition contains the actual implementation.
E.g.: void stars () // declarator
{
for(int j=10; j>=0; j--) //function body
cout<<"*";
cout<<endl; }

What is RTTI?

Runtime type identification (RTTI) lets you find the dynamic type of an object when you have only a pointer or a reference to the base type. RTTI is the official way in standard C++ to discover the type of an object and to convert the type of a pointer or reference (that is, dynamic typing). The need came from practical experience with C++. RTTI replaces many Interview Questions - Homegrown versions with a solid, consistent approach.

What is encapsulation?
Packaging an object's variables within its methods is called encapsulation.

Explain term POLIMORPHISM and give an example using eg. SHAPE object: If I have a base class SHAPE, how would I define DRAW methods for two objects CIRCLE and SQUARE

Answer1
POLYMORPHISM : A phenomenon which enables an object to react differently to the same function call.
in C++ it is attained by using a keyword virtual

Example
public class SHAPE
{
public virtual void SHAPE::DRAW()=0;
}
Note here the function DRAW() is pure virtual which means the sub classes must implement the DRAW() method and SHAPE cannot be instatiated

public class CIRCLE::public SHAPE
{
public void CIRCLE::DRAW()
{
// TODO drawing circle
}
}
public class SQUARE::public SHAPE
{
public void SQUARE::DRAW()
{
// TODO drawing square
}
}

```
now from the user class the calls would be like
globally
SHAPE *newShape;

When user action is to draw
public void MENU::OnClickDrawCircle(){
newShape = new CIRCLE();
}

public void MENU::OnClickDrawCircle(){
newShape = new SQUARE();

}

the when user actually draws
public void CANVAS::OnMouseOperations(){
newShape->DRAW();
}
```

Answer2

```
class SHAPE{
public virtual Draw() = 0; //abstract class with a pure virtual method
};

class CIRCLE{
public int r;
public virtual Draw() { this->drawCircle(0,0,r); }
};

class SQURE
public int a;
public virtual Draw() { this->drawRectangular(0,0,a,a); }
};
```

Each object is driven down from SHAPE implementing Draw() function in its own way.

What is an object?
Object is a software bundle of variables and related methods. Objects have state and behavior.

How can you tell what shell you are running on UNIX system?
You can do the Echo $RANDOM. It will return a undefined variable if you are from the C-Shell, just a return prompt if you are from the Bourne shell, and a 5 digit random numbers if you are from the Korn shell. You could also do a ps -l and look for the shell with the highest PID.

What do you mean by inheritance?
Inheritance is the process of creating new classes, called derived classes, from existing classes or base classes. The derived class inherits all the capabilities of the base class, but can add embellishments and refinements of its own.

Describe PRIVATE, PROTECTED and PUBLIC – the differences and give examples.

```
class Point2D{
int x; int y;

public int color;
protected bool pinned;
public Point2D() : x(0) , y(0) {} //default (no argument) constructor
};

Point2D MyPoint;
```

You cannot directly access private data members when they are declared (implicitly) private:

```
MyPoint.x = 5; // Compiler will issue a compile ERROR
//Nor yoy can see them:
int x_dim = MyPoint.x; // Compiler will issue a compile ERROR
```

On the other hand, you can assign and read the public data members:

```
MyPoint.color = 255; // no problem
int col = MyPoint.color; // no problem
```

With protected data members you can read them but not write them:

```
MyPoint.pinned = true; // Compiler will issue a compile ERROR

bool isPinned = MyPoint.pinned; // no problem
```

What is namespace?

Namespaces allow us to group a set of global classes, objects and/or functions under a name. To say it somehow, they serve to split the global scope in sub-scopes known as namespaces.
The form to use namespaces is:
namespace identifier { namespace-body }
Where identifier is any valid identifier and namespace-body is the set of classes, objects and functions that are included within the namespace. For example:
namespace general { int a, b; } In this case, a and b are normal variables integrated within the general namespace. In order to access to these variables from outside the namespace we have to use the scope operator ::. For example, to access the previous variables we would have to put:
general::a general::b
The functionality of namespaces is specially useful in case that there is a possibility that a global object or function can have the same name than another one, causing a redefinition error.

What is a COPY CONSTRUCTOR and when is it called?

A copy constructor is a method that accepts an object of the same class and copies it's data members to the object on the left part of assignement:

```
class Point2D{
int x; int y;
```

```
public int color;
protected bool pinned;
public Point2D() : x(0) , y(0) {} //default (no argument) constructor
public Point2D( const Point2D & ) ;
};

Point2D::Point2D( const Point2D & p )
{
this->x = p.x;
this->y = p.y;
this->color = p.color;
this->pinned = p.pinned;
}

main(){
Point2D MyPoint;
MyPoint.color = 345;
Point2D AnotherPoint = Point2D( MyPoint ); // now AnotherPoint has color =
345
```

What is Boyce Codd Normal form?
A relation schema R is in BCNF with respect to a set F of functional dependencies if for all functional dependencies in F+ of the form a-> , where a and b is a subset of R, at least one of the following holds:
* a- > b is a trivial functional dependency (b is a subset of a)
* a is a superkey for schema R

What is virtual class and friend class?
Friend classes are used when two or more classes are designed to work together and need access to each other's implementation in ways that the rest of the world shouldn't be allowed to have. In other words, they help keep private things private. For instance, it may be desirable for class DatabaseCursor to have more privilege to the internals of class Database than main() has.

What is the word you will use when defining a function in base class to allow this function to be a polimorphic function?
virtual

What do you mean by binding of data and functions?
Encapsulation.

What are 2 ways of exporting a function from a DLL?
1.Taking a reference to the function from the DLL instance.
2. Using the DLL 's Type Library

What is the difference between an object and a class?
Classes and objects are separate but related concepts. Every object belongs to a class and every class contains one or more related objects.
- A Class is static. All of the attributes of a class are fixed before, during, and after the execution of a program. The attributes of a class don't change.

- The class to which an object belongs is also (usually) static. If a particular object belongs to a certain class at the time that it is created then it almost certainly will still belong to that class right up until the time that it is destroyed.
- An Object on the other hand has a limited lifespan. Objects are created and eventually destroyed. Also during that lifetime, the attributes of the object may undergo significant change.

Suppose that data is an array of 1000 integers. Write a single function call that will sort the 100 elements data [222] through data [321].
quicksort ((data + 222), 100);

What is a class?
Class is a user-defined data type in C++. It can be created to solve a particular kind of problem. After creation the user need not know the specifics of the working of a class.

What is friend function?
As the name suggests, the function acts as a friend to a class. As a friend of a class, it can access its private and protected members. A friend function is not a member of the class. But it must be listed in the class definition.

Which recursive sorting technique always makes recursive calls to sort subarrays that are about half size of the original array?
Mergesort always makes recursive calls to sort subarrays that are about half size of the original array, resulting in O(n log n) time.

What is abstraction?
Abstraction is of the process of hiding unwanted details from the user.

What are virtual functions?
A virtual function allows derived classes to replace the implementation provided by the base class. The compiler makes sure the replacement is always called whenever the object in question is actually of the derived class, even if the object is accessed by a base pointer rather than a derived pointer. This allows algorithms in the base class to be replaced in the derived class, even if users don't know about the derived class.

What is the difference between an external iterator and an internal iterator? Describe an advantage of an external iterator.
An internal iterator is implemented with member functions of the class that has items to step through. .An external iterator is implemented as a separate class that can be "attach" to the object that has items to step through. .An external iterator has the advantage that many difference iterators can be active simultaneously on the same object.

What is a scope resolution operator?
A scope resolution operator (::), can be used to define the member functions of a class outside the class.

What do you mean by pure virtual functions?
A pure virtual member function is a member function that the base class forces derived classes to provide. Normally these member functions have no

implementation. Pure virtual functions are equated to zero.
class Shape { public: virtual void draw() = 0; };

What is polymorphism? Explain with an example?
"Poly" means "many" and "morph" means "form". Polymorphism is the ability of an object (or reference) to assume (be replaced by) or become many different forms of object.
Example: function overloading, function overriding, virtual functions. Another example can be a plus '+' sign, used for adding two integers or for using it to concatenate two strings.

What's the output of the following program? Why?

```
#include <stdio.h>
main()
{
typedef union
{
int a;
char b[10];
float c;
}
Union;

Union x,y = {100};
x.a = 50;
strcpy(x.b,\"hello\");
x.c = 21.50;

printf(\"Union x : %d %s %f \n\",x.a,x.b,x.c );
printf(\"Union y :%d %s%f \n\",y.a,y.b,y.c);
}
```

Given inputs X, Y, Z and operations | and & (meaning bitwise OR and AND, respectively)
What is output equal to in
output = (X & Y) | (X & Z) | (Y & Z)

Why are arrays usually processed with for loop?

The real power of arrays comes from their facility of using an index variable to traverse the array, accessing each element with the same expression a[i]. All the is needed to make this work is a iterated statement in which the variable i serves as a counter, incrementing from 0 to a.length -1. That is exactly what a loop does.

What is an HTML tag?
Answer: An HTML tag is a syntactical construct in the HTML language that abbreviates specific instructions to be executed when the HTML script is loaded into a Web browser. It is like a method in Java, a function in C++, a procedure in Pascal, or a subroutine in FORTRAN.

Explain which of the following declarations will compile and what will be constant - a pointer or the value pointed at: * const char *
*** char const ***
*** char * const**

Note: Ask the candidate whether the first declaration is pointing to a string or a single character. Both explanations are correct, but if he says that it's a single character pointer, ask why a whole string is initialized as char* in C++. If he says this is a string declaration, ask him to declare a pointer to a single character. Competent candidates should not have problems pointing out why const char* can be both a character and a string declaration, incompetent ones will come up with invalid reasons.

You're given a simple code for the class Bank Customer. Write the following functions:
*** Copy constructor**
*** = operator overload**
*** == operator overload**
*** + operator overload (customers' balances should be added up, as an example of joint account between husband and wife)**

Note:Anyone confusing assignment and equality operators should be dismissed from the interview. The applicant might make a mistake of passing by value, not by reference. The candidate might also want to return a pointer, not a new object, from the addition operator. Slightly hint that you'd like the value to be changed outside the function, too, in the first case. Ask him whether the statement customer3 = customer1 + customer2 would work in the second case.

What problems might the following macro bring to the application?
#define sq(x) x*x

Anything wrong with this code?
T *p = new T[10];
delete p;

Everything is correct, Only the first element of the array will be deleted", The entire array will be deleted, but only the first element destructor will be called.

Anything wrong with this code?
T *p = 0;
delete p;

Yes, the program will crash in an attempt to delete a null pointer.

How do you decide which integer type to use?
It depends on our requirement. When we are required an integer to be stored in 1 byte (means less than or equal to 255) we use short int, for 2 bytes we use int, for 8 bytes we use long int.

A char is for 1-byte integers, a short is for 2-byte integers, an int is generally a 2-

byte or 4-byte integer (though not necessarily), a long is a 4-byte integer, and a long long is a 8-byte integer.

What's the output of the following program? Why?

```
#include <stdio.h>
main()
{
typedef union
{
int a;
char b[10];
float c;
}
Union;

Union x,y = {100};
x.a = 50;
strcpy(x.b,\"hello\");
x.c = 21.50;

printf(\"Union x : %d %s %f \n\",x.a,x.b,x.c );
printf(\"Union y :%d %s%f \n\",y.a,y.b,y.c);
}
```

Given inputs X, Y, Z and operations | and & (meaning bitwise OR and AND, respectively)
What is output equal to in
output = (X & Y) | (X & Z) | (Y & Z)

Why are arrays usually processed with for loop?

The real power of arrays comes from their facility of using an index variable to traverse the array, accessing each element with the same expression a[i]. All the is needed to make this work is a iterated statement in which the variable i serves as a counter, incrementing from 0 to a.length -1. That is exactly what a loop does.

What is an HTML tag?
Answer: An HTML tag is a syntactical construct in the HTML language that abbreviates specific instructions to be executed when the HTML script is loaded into a Web browser. It is like a method in Java, a function in C++, a procedure in Pascal, or a subroutine in FORTRAN.

Explain which of the following declarations will compile and what will be constant - a pointer or the value pointed at: * const char *
*** char const ***
*** char * const**

Note: Ask the candidate whether the first declaration is pointing to a string or a single character. Both explanations are correct, but if he says that it's a single character pointer, ask why a whole string is initialized as char* in C++. If he says this is a string declaration, ask him to declare a pointer to a single character. Competent candidates should not have problems pointing out why const char* can be both a character and a string declaration, incompetent ones will come up with invalid reasons.

You're given a simple code for the class Bank Customer. Write the following functions:

*** Copy constructor**
*** = operator overload**
*** == operator overload**
*** + operator overload (customers' balances should be added up, as an example of joint account between husband and wife)**

Note:Anyone confusing assignment and equality operators should be dismissed from the interview. The applicant might make a mistake of passing by value, not by reference. The candidate might also want to return a pointer, not a new object, from the addition operator. Slightly hint that you'd like the value to be changed outside the function, too, in the first case. Ask him whether the statement customer3 = customer1 + customer2 would work in the second case.

What problems might the following macro bring to the application?
#define sq(x) x*x

Anything wrong with this code?
T *p = new T[10];
delete p;

Everything is correct, Only the first element of the array will be deleted", The entire array will be deleted, but only the first element destructor will be called.

Anything wrong with this code?
T *p = 0;
delete p;

Yes, the program will crash in an attempt to delete a null pointer.

How do you decide which integer type to use?
It depends on our requirement. When we are required an integer to be stored in 1 byte (means less than or equal to 255) we use short int, for 2 bytes we use int, for 8 bytes we use long int.

A char is for 1-byte integers, a short is for 2-byte integers, an int is generally a 2-byte or 4-byte integer (though not necessarily), a long is a 4-byte integer, and a long long is a 8-byte integer.

How do I initialize a pointer to a function?
This is the way to initialize a pointer to a function
void fun(int a)
{

```
}

void main()
{
void (*fp)(int);
fp=fun;
fp(1);

}
```

How do you link a C++ program to C functions?
By using the extern "C" linkage specification around the C function declarations.

Explain the scope resolution operator.
It permits a program to reference an identifier in the global scope that has been hidden by another identifier with the same name in the local scope.

What are the differences between a C++ struct and C++ class?
The default member and base-class access specifier are different.

How many ways are there to initialize an int with a constant?
Two.
There are two formats for initializers in C++ as shown in the example that follows. The first format uses the traditional C notation. The second format uses constructor notation.

```
int foo = 123;
int bar (123);
```

How does throwing and catching exceptions differ from using setjmp and longjmp?
The throw operation calls the destructors for automatic objects instantiated since entry to the try block.

What is a default constructor?
Default constructor WITH arguments class B { public: B (int m = 0) : n (m) {} int n; }; int main(int argc, char *argv[]) { B b; return 0; }

What is a conversion constructor?
A constructor that accepts one argument of a different type.

What is the difference between a copy constructor and an overloaded assignment operator?
A copy constructor constructs a new object by using the content of the argument object. An overloaded assignment operator assigns the contents of an existing object to another existing object of the same class.

When should you use multiple inheritance?
There are three acceptable answers: "Never," "Rarely," and "When the problem domain cannot be accurately modeled any other way."

Explain the ISA and HASA class relationships. How would you implement each in a class design?

A specialized class "is" a specialization of another class and, therefore, has the ISA relationship with the other class. An Employee ISA Person. This relationship is best implemented with inheritance. Employee is derived from Person. A class may have an instance of another class. For example, an employee "has" a salary, therefore the Employee class has the HASA relationship with the Salary class. This relationship is best implemented by embedding an object of the Salary class in the Employee class.

When is a template a better solution than a base class?
When you are designing a generic class to contain or otherwise manage objects of other types, when the format and behavior of those other types are unimportant to their containment or management, and particularly when those other types are unknown (thus, the generosity) to the designer of the container or manager class.

What is a mutable member?
One that can be modified by the class even when the object of the class or the member function doing the modification is const.

What is an explicit constructor?
A conversion constructor declared with the explicit keyword. The compiler does not use an explicit constructor to implement an implied conversion of types. It's purpose is reserved explicitly for construction.

What is the Standard Template Library (STL)?
A library of container templates approved by the ANSI committee for inclusion in the standard C++ specification.
A programmer who then launches into a discussion of the generic programming model, iterators, allocators, algorithms, and such, has a higher than average understanding of the new technology that STL brings to C++ programming.

Describe run-time type identification.
The ability to determine at run time the type of an object by using the typeid operator or the dynamic_cast operator.

What problem does the namespace feature solve?
Multiple providers of libraries might use common global identifiers causing a name collision when an application tries to link with two or more such libraries. The namespace feature surrounds a library's external declarations with a unique namespace that eliminates the potential for those collisions.
This solution assumes that two library vendors don't use the same namespace identifier, of course.

Are there any new intrinsic (built-in) data types?
Yes. The ANSI committee added the bool intrinsic type and its true and false value keywords.

Will the following program execute?
```
void main()
{
void *vptr = (void *) malloc(sizeof(void));
vptr++; }
```

Answer1
It will throw an error, as arithmetic operations cannot be performed on void pointers.

Answer2
It will not build as sizeof cannot be applied to void* (error "Unknown size")

Answer3
How can it execute if it won't even compile? It needs to be int main, not void main. Also, cannot increment a void *.

Answer4
According to gcc compiler it won't show any error, simply it executes. but in general we can't do arthematic operation on void, and gives size of void as 1

Answer5
The program compiles in GNU C while giving a warning for "void main". The program runs without a crash. sizeof(void) is "1? hence when vptr++, the address is incremented by 1.

Answer6
Regarding arguments about GCC, be aware that this is a C++ question, not C. So gcc will compile and execute, g++ cannot. g++ complains that the return type cannot be void and the argument of sizeof() cannot be void. It also reports that ISO C++ forbids incrementing a pointer of type 'void*'.

Answer7
in C++
voidp.c: In function `int main()':
voidp.c:4: error: invalid application of `sizeof' to a void type
voidp.c:4: error: `malloc' undeclared (first use this function)
voidp.c:4: error: (Each undeclared identifier is reported only once for each function it appears in.)
voidp.c:6: error: ISO C++ forbids incrementing a pointer of type `void*'

But in c, it work without problems

```
void main()
{
char *cptr = 0?2000;
long *lptr = 0?2000;
cptr++;
lptr++;
printf(" %x %x", cptr, lptr);
}
```

Will it execute or not?
Answer1
For Q2: As above, won't compile because main must return int. Also, 0×2000 cannot be implicitly converted to a pointer (I assume you meant 0×2000 and not 0?2000.)

Answer2

```
Not Excute.
Compile with VC7 results following errors:
error C2440: ‘initializing’ : cannot convert from ‘int’ to ‘char *’
error C2440: ‘initializing’ : cannot convert from ‘int’ to ‘long *’
```

```
Not Excute if it is C++, but Excute in C.
The printout:
2001 2004
```

Answer3
In C++

```
[$]> g++ point.c
point.c: In function `int main()’:
point.c:4: error: invalid conversion from `int’ to `char*’
point.c:5: error: invalid conversion from `int’ to `long int*’
```

in C

```
[$] etc > gcc point.c
point.c: In function `main’:
point.c:4: warning: initialization makes pointer from integer without a cast
point.c:5: warning: initialization makes pointer from integer without a cast
[$] etc > ./a.exe
2001 2004
```

What is the difference between Mutex and Binary semaphore?

semaphore is used to synchronize processes. where as mutex is used to provide synchronization between threads running in the same process.

In C++, what is the difference between method overloading and method overriding?

Overloading a method (or function) in C++ is the ability for functions of the same name to be defined as long as these methods have different signatures (different set of parameters). Method overriding is the ability of the inherited class rewriting the virtual method of the base class.

What methods can be overridden in Java?

In C++ terminology, all public methods in Java are virtual. Therefore, all Java methods can be overwritten in subclasses except those that are declared final, static, and private.

What are the defining traits of an object-oriented language?

The defining traits of an object-oriented langauge are:
* encapsulation
* inheritance
* polymorphism

Write a program that ask for user input from 5 to 9 then calculate the average

```
int main()
{
int MAX=4;
int total =0;
int average=0;
int numb;
cout<<"Please enter your input from 5 to 9";
cin>>numb;
if((numb <5)&&(numb>9))
cout<<"please re type your input";
else
for(i=0;i<=MAX; i++)
{
total = total + numb;
average= total /MAX;
}
cout<<"The average number is"<<average<<endl;

return 0;
}
```

Assignment Operator - What is the diffrence between a "assignment operator" and a "copy constructor"?

Answer1.

In assignment operator, you are assigning a value to an existing object. But in copy constructor, you are creating a new object and then assigning a value to that object. For example:

```
complex c1,c2;
c1=c2; //this is assignment
complex c3=c2; //copy constructor
```

Answer2.

A copy constructor is used to initialize a newly declared variable from an existing variable. This makes a deep copy like assignment, but it is somewhat simpler:

There is no need to test to see if it is being initialized from itself.
There is no need to clean up (eg, delete) an existing value (there is none).
A reference to itself is not returned.

RTTI - What is RTTI?

Answer1.

RTTI stands for "Run Time Type Identification". In an inheritance hierarchy, we can find out the exact type of the objet of which it is member. It can be done by using:

1) dynamic id operator
2) typecast operator

Answer2.

RTTI is defined as follows: Run Time Type Information, a facility that allows an object to be queried at runtime to determine its type. One of the fundamental principles of object technology is polymorphism, which is the ability of an object to dynamically change at runtime.

STL Containers - What are the types of STL containers?
There are 3 types of STL containers:

1. Adaptive containers like queue, stack
2. Associative containers like set, map
3. Sequence containers like vector, deque

What is the need for a Virtual Destructor ?
Destructors are declared as virtual because if do not declare it as virtual the base class destructor will be called before the derived class destructor and that will lead to memory leak because derived classâ€™s objects will not get freed.Destructors are declared virtual so as to bind objects to the methods at runtime so that appropriate destructor is called.

What is "mutable"?

Answer1.
"mutable" is a C++ keyword. When we declare const, none of its data members can change. When we want one of its members to change, we declare it as mutable.

Answer2.
A "mutable" keyword is useful when we want to force a "logical const" data member to have its value modified. A logical const can happen when we declare a data member as non-const, but we have a const member function attempting to modify that data member. For example:

```
class Dummy {
public:
bool isValid() const;
private:
mutable int size_ = 0;
mutable bool validStatus_ = FALSE;
// logical const issue resolved
};

bool Dummy::isValid() const
// data members become bitwise const
{
if (size > 10) {
validStatus_ = TRUE; // fine to assign
size = 0; // fine to assign
}
}
```

Answer2.
"mutable" keyword in C++ is used to specify that the member may be updated or modified even if it is member of constant object. Example:

```
class Animal {
private:
string name;
string food;
mutable int age;
public:
void set_age(int a);
};

void main() {
const Animal Tiger(â€™Fulffyâ€™,'antelopeâ€™,1);
Tiger.set_age(2);
// the age can be changed since its mutable
}
```

Differences of C and C++

Could you write a small program that will compile in C but not in C++ ?

In C, if you can a const variable e.g.
const int i = 2;
you can use this variable in other module as follows
extern const int i;
C compiler will not complain.

But for C++ compiler u must write
extern const int i = 2;
else error would be generated.

Bitwise Operations - Given inputs X, Y, Z and operations | and & (meaning bitwise OR and AND, respectively), what is output equal to in?
output = (X & Y) | (X & Z) | (Y & Z);

What is a modifier?
A modifier, also called a modifying function is a member function that changes the value of at least one data member. In other words, an operation that modifies the state of an object. Modifiers are also known as ‘mutators’. Example: The function mod is a modifier in the following code snippet:

```
class test
{
int x,y;
public:
test()
{
x=0; y=0;
}
void mod()
{
```

```
x=10;
y=15;
}
};
```

What is an accessor?
An accessor is a class operation that does not modify the state of an object. The accessor functions need to be declared as const operations

Differentiate between a template class and class template.

Template class: A generic definition or a parameterized class not instantiated until the client provides the needed information. It's jargon for plain templates.
Class template: A class template specifies how individual classes can be constructed much like the way a class specifies how individual objects can be constructed. It's jargon for plain classes.

When does a name clash occur?
A name clash occurs when a name is defined in more than one place. For example., two different class libraries could give two different classes the same name. If you try to use many class libraries at the same time, there is a fair chance that you will be unable to compile or link the program because of name clashes.

Define namespace.
It is a feature in C++ to minimize name collisions in the global name space. This namespace keyword assigns a distinct name to a library that allows other libraries to use the same identifier names without creating any name collisions. Furthermore, the compiler uses the namespace signature for differentiating the definitions.

What is the use of 'using' declaration. ?
A using declaration makes it possible to use a name from a namespace without the scope operator.

What is an Iterator class ?
A class that is used to traverse through the objects maintained by a container class. There are five categories of iterators: input iterators, output iterators, forward iterators, bidirectional iterators, random access. An iterator is an entity that gives access to the contents of a container object without violating encapsulation constraints. Access to the contents is granted on a one-at-a-time basis in order. The order can be storage order (as in lists and queues) or some arbitrary order (as in array indices) or according to some ordering relation (as in an ordered binary tree). The iterator is a construct, which provides an interface that, when called, yields either the next element in the container, or some value denoting the fact that there are no more elements to examine. Iterators hide the details of access to and update of the elements of a container class.
The simplest and safest iterators are those that permit read-only access to the contents of a container class.

What is an incomplete type?
Incomplete types refers to pointers in which there is non availability of the implementation of the referenced location or it points to some location whose value is not available for modification.

```
int *i=0x400 // i points to address 400
*i=0; //set the value of memory location pointed by i.
```

Incomplete types are otherwise called uninitialized pointers.

What is a dangling pointer?
A dangling pointer arises when you use the address of an object after its lifetime is over. This may occur in situations like returning addresses of the automatic variables from a function or using the address of the memory block after it is freed. The following code snippet shows this:

```
class Sample
{
public:
int *ptr;
Sample(int i)
{
ptr = new int(i);
}

~Sample()
{
delete ptr;
}
void PrintVal()
{
cout << "The value is " << *ptr;
}
};

void SomeFunc(Sample x)
{
cout << "Say i am in someFunc " << endl;
}

int main()
{
Sample s1 = 10;
SomeFunc(s1);
s1.PrintVal();
}
```

In the above example when PrintVal() function is called it is called by the pointer that has been freed by the destructor in SomeFunc.

Differentiate between the message and method.
Message:
* Objects communicate by sending messages to each other.
* A message is sent to invoke a method.

Method
* Provides response to a message.
* It is an implementation of an operation.

What is an adaptor class or Wrapper class?
A class that has no functionality of its own. Its member functions hide the use of a third party software component or an object with the non-compatible interface or a non-object-oriented implementation.

What is a Null object?
It is an object of some class whose purpose is to indicate that a real object of that class does not exist. One common use for a null object is a return value from a member function that is supposed to return an object with some specified properties but cannot find such an object.

What is class invariant?
A class invariant is a condition that defines all valid states for an object. It is a logical condition to ensure the correct working of a class. Class invariants must hold when an object is created, and they must be preserved under all operations of the class. In particular all class invariants are both preconditions and post-conditions for all operations or member functions of the class.

What do you mean by Stack unwinding?
It is a process during exception handling when the destructor is called for all local objects between the place where the exception was thrown and where it is caught.

Define precondition and post-condition to a member function.
Precondition: A precondition is a condition that must be true on entry to a member function. A class is used correctly if preconditions are never false. An operation is not responsible for doing anything sensible if its precondition fails to hold. For example, the interface invariants of stack class say nothing about pushing yet another element on a stack that is already full. We say that isful() is a precondition of the push operation. Post-condition: A post-condition is a condition that must be true on exit from a member function if the precondition was valid on entry to that function. A class is implemented correctly if post-conditions are never false. For example, after pushing an element on the stack, we know that isempty() must necessarily hold. This is a post-condition of the push operation.

What are the conditions that have to be met for a condition to be an invariant of the class?
* The condition should hold at the end of every constructor.
* The condition should hold at the end of every mutator (non-const) operation.

What are proxy objects?
Objects that stand for other objects are called proxy objects or surrogates.

```
template <class t="">
class Array2D
{
public:
class Array1D
{
public:
T& operator[] (int index);
const T& operator[] (int index)const;
};

Array1D operator[] (int index);
const Array1D operator[] (int index) const;
};
```

The following then becomes legal:

```
Array2D<float>data(10,20);
cout<<data[3][6]; // fine
```

Here data[3] yields an Array1D object and the operator [] invocation on that object yields the float in position(3,6) of the original two dimensional array. Clients of the Array2D class need not be aware of the presence of the Array1D class. Objects of this latter class stand for one-dimensional array objects that, conceptually, do not exist for clients of Array2D. Such clients program as if they were using real, live, two-dimensional arrays. Each Array1D object stands for a one-dimensional array that is absent from a conceptual model used by the clients of Array2D. In the above example, Array1D is a proxy class. Its instances stand for one-dimensional arrays that, conceptually, do not exist.

Name some pure object oriented languages.
Smalltalk, Java, Eiffel, Sather.

What is an orthogonal base class?
If two base classes have no overlapping methods or data they are said to be independent of, or orthogonal to each other. Orthogonal in the sense means that two classes operate in different dimensions and do not interfere with each other in any way. The same derived class

What is a node class?
A node class is a class that,
* relies on the base class for services and implementation,
* provides a wider interface to the users than its base class,
* relies primarily on virtual functions in its public interface
* depends on all its direct and indirect base class
* can be understood only in the context of the base class
* can be used as base for further derivation
* can be used to create objects.

A node class is a class that has added new services or functionality beyond the services inherited from its base class.

What is a container class? What are the types of container classes?
A container class is a class that is used to hold objects in memory or external storage. A container class acts as a generic holder. A container class has a predefined behavior and a well-known interface. A container class is a supporting class whose purpose is to hide the topology used for maintaining the list of objects in memory. When a container class contains a group of mixed objects, the container is called a heterogeneous container; when the container is holding a group of objects that are all the same, the container is called a homogeneous container.

How do you write a function that can reverse a linked-list?
Answer1:

```
void reverselist(void)
{
if(head==0)
return;
if(head-<next==0)
return;
if(head-<next==tail)
{
head-<next = 0;
tail-<next = head;
}
else
{
node* pre = head;
node* cur = head-<next;
node* curnext = cur-<next;
head-<next = 0;
cur-<next = head;

for(; curnext!=0; )
{
cur-<next = pre;
pre = cur;
cur = curnext;
curnext = curnext-<next;
}

curnext-<next = cur;
}
}
```

Answer2:

```
node* reverselist(node* head)
{
if(0==head || 0==head->next)
//if head->next ==0 should return head instead of 0;
return 0;

{
node* prev = head;
node* curr = head->next;
node* next = curr->next;

for(; next!=0; )
{
curr->next = prev;
prev = curr;
curr = next;
next = next->next;
}
curr->next = prev;

head->next = 0;
head = curr;
}

return head;
}
```

What is polymorphism?
Polymorphism is the idea that a base class can be inherited by several classes. A base class pointer can point to its child class and a base class array can store different child class objects.

How do you find out if a linked-list has an end? (i.e. the list is not a cycle)
You can find out by using 2 pointers. One of them goes 2 nodes each time. The second one goes at 1 nodes each time. If there is a cycle, the one that goes 2 nodes each time will eventually meet the one that goes slower. If that is the case, then you will know the linked-list is a cycle.

may inherit such classes with no difficulty

How can you tell what shell you are running on UNIX system?
You can do the Echo $RANDOM. It will return a undefined variable if you are from the C-Shell, just a return prompt if you are from the Bourne shell, and a 5 digit random numbers if you are from the Korn shell. You could also do a ps -l and look for the shell with the highest PID.

What is Boyce Codd Normal form?
A relation schema R is in BCNF with respect to a set F of functional dependencies if for all functional dependencies in F+ of the form a->b, where a and b is a subset of R, at least one of the following holds:

* a->b is a trivial functional dependency (b is a subset of a)
* a is a superkey for schema R

What is pure virtual function?
A class is made abstract by declaring one or more of its virtual functions to be pure. A pure virtual function is one with an initializer of = 0 in its declaration

Write a Struct Time where integer m, h, s are its members

```
struct Time
{
int m;
int h;
int s;
};
```

How do you traverse a Btree in Backward in-order?
Process the node in the right subtree
Process the root
Process the node in the left subtree

What is the two main roles of Operating System?
As a resource manager
As a virtual machine

In the derived class, which data member of the base class are visible?
In the public and protected sections.

Could you tell something about the Unix System Kernel?
The kernel is the heart of the UNIX openrating system, it's reponsible for controlling the computer's resouces and scheduling user jobs so that each one gets its fair share of resources.

What are each of the standard files and what are they normally associated with?
They are the standard input file, the standard output file and the standard error file. The first is usually associated with the keyboard, the second and third are usually associated with the terminal screen.

Detemine the code below, tell me exectly how many times is the operation sum++ performed ?

```
for ( i = 0; i < 100; i++ )
for ( j = 100; j > 100 - i; j–)
sum++;
```

(99 * 100)/2 = 4950
The sum++ is performed 4950 times.

Give 4 examples which belongs application layer in TCP/IP architecture?
FTP, TELNET, HTTP and TFTP

What's the meaning of ARP in TCP/IP?
The "ARP" stands for Address Resolution Protocol. The ARP standard defines two basic message types: a request and a response. a request message contains an IP address and requests the corresponding hardware address; a replay contains both the IP address, sent in the request, and the hardware address.

What is a Makefile?
Makefile is a utility in Unix to help compile large programs. It helps by only compiling the portion of the program that has been changed.
A Makefile is the file and make uses to determine what rules to apply. make is useful for far more than compiling programs.

What is deadlock?
Deadlock is a situation when two or more processes prevent each other from running. Example: if T1 is holding x and waiting for y to be free and T2 holding y and waiting for x to be free deadlock happens.

What is semaphore?
Semaphore is a special variable, it has two methods: up and down. Semaphore performs atomic operations, which means ones a semaphore is called it can not be inturrupted.

The internal counter (= #ups - #downs) can never be negative. If you execute the "down" method when the internal counter is zero, it will block until some other thread calls the "up" method. Semaphores are use for thread synchronization.

Is C an object-oriented language?
C is not an object-oriented language, but limited object-oriented programming can be done in C.

Name some major differences between C++ and Java.
C++ has pointers; Java does not. Java is platform-independent; C++ is not. Java has garbage collection; C++ does not. Java does have pointers. In fact all variables in Java are pointers. The difference is that Java does not allow you to manipulate the addresses of the pointer

What is the difference between Stack and Queue?
Stack is a Last In First Out (LIFO) data structure.
Queue is a First In First Out (FIFO) data structure

Write a fucntion that will reverse a string.
```
char *strrev(char *s)
{
int i = 0, len = strlen(s);
char *str;
if ((str = (char *)malloc(len+1)) == NULL)
/*cannot allocate memory */
err_num = 2;
return (str);
}
while(len)
str[i++]=s[–len];
str[i] = NULL;
return (str);
}
```

What is the software Life-Cycle?
The software Life-Cycle are
1) Analysis and specification of the task
2) Design of the algorithms and data structures
3) Implementation (coding)
4) Testing
5) Maintenance and evolution of the system
6) Obsolescence

What is the difference between a Java application and a Java applet?
The difference between a Java application and a Java applet is that a Java application is a program that can be executed using the Java interpeter, and a JAVA applet can be transfered to different networks and executed by using a web browser (transferable to the WWW).

Name 7 layers of the OSI Reference Model?
-Application layer
-Presentation layer
-Session layer
-Transport layer
-Network layer
-Data Link layer
-Physical layer

What are the advantages and disadvantages of B-star trees over Binary trees?

Answer1
B-star trees have better data structure and are faster in search than Binary trees, but it's harder to write codes for B-start trees.

Answer2
The major difference between B-tree and binary tres is that B-tree is a external data structure and binary tree is a main memory data structure. The computational complexity of binary tree is counted by the number of comparison operations at each node, while the computational complexity of B-tree is determined by the disk I/O, that is, the number of node that will be loaded from disk to main memory. The comparision of the different values in one node is not counted.

Write the psuedo code for the Depth first Search.

```
dfs(G, v) //OUTLINE
Mark v as "discovered"
For each vertex w such that edge vw is in G:
If w is undiscovered:
dfs(G, w); that is, explore vw, visit w, explore from there as much as possible,
and backtrack from w to v. Otherwise:
"Check" vw without visiting w. Mark v as "finished".
```

Describe one simple rehashing policy.

The simplest rehashing policy is linear probing. Suppose a key K hashes to location i. Suppose other key occupies H[i]. The following function is used to generate alternative locations:
rehash(j) = (j + 1) mod h
where j is the location most recently probed. Initially j = i, the hash code for K. Notice that this version of rehash does not depend on K.

Describe Stacks and name a couple of places where stacks are useful.

A Stack is a linear structure in which insertions and deletions are always made at one end, called the top. This updating policy is called last in, first out (LIFO). It is useful when we need to check some syntex errors, such as missing parentheses.

Suppose a 3-bit sequence number is used in the selective-reject ARQ, what is the maximum number of frames that could be transmitted at a time?

If a 3-bit sequence number is used, then it could distinguish 8 different frames. Since the number of frames that could be transmitted at a time is no greater half the numner of frames that could be distinguished by the sequence number, so at most 4 frames can be transmitted at a time.

REFERENCES:

1)DATA STRUCTURES USING C AND C++ -Yedidyah Langsam & M.Tenenbaum

2)OBJECT-ORIENTED PROGRAMMING IN C++ -Nabajyoti Barkakati

3) OBJECT-ORIENTED PROGRAMMING IN C++ -E Balagurusamy

4)C AND DATA STRUCTURES –A.S.R.MURTHY