1(a) Create a Java class called *Student*with the following details as variables within it.
(i) USN
(ii) Name
(iii) Branch
(iv) Phone
Write a Java program to create *nStudent* objects and print the USN, Name, Branch, and Phoneof these objects with suitable headings.

```
import java.util.*;
public class student
{
	String usn,name,branch,ph;
	static Scanner S1=new Scanner(System.in);
	public static void main(String[] args)
	{
		int i;
		student[] student=new student[10];
		System.out.println("enter number of students");
		int n=S1.nextInt();
		System.out.println("enter"+n+"student details");
		for(i=0;i<n;i++)
		{
			//St1 Student[i]=new St1();
			student[i]=new student();
			System.out.println("enter student"+(i+1)+"details");;
			System.out.println("enter students usn,branch,name and phno");
			student[i].usn=S1.next();
			student[i].branch=S1.next();
			student[i].name=S1.next();
			student[i].ph=S1.next();
		}
		System.out.println("USN\t\tName\t\tBranch\t\tphonenumber");
		for(i=0;i<n;i++)

	System.out.println("USN="+student[i].usn+"\tName="+student[i].name+"\tBranch="+stu
dent[i].branch+"\tPhno="+student[i].ph);
	}
}
```

Output:

```
enter number of students
2
enter2student details
enter student1details
enter students usn,branch,name and phno
4ai14is096
ISE
```

raju
8945638476
enter student2details
enter students usn,branch,name and phno
4ai13cs067
CSE
sanjay
9876878654
USN Name Branch phonenumber
USN=4ai14is096 Name=raju Branch=ISE Phno=8945638476
USN=4ai13cs067 Name=sanjay Branch=CSE Phno=9876878654

1(b) Write a Java program to implement the Stack using arrays. Write Push(), Pop(), and Display() methods to demonstrate its working.

```
import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

class Stack

{

        private int top;

        private int item[];

        Stack(int size)

        {

                top = -1;

                item = new int[size];

        }

        void pushItem(int data)

        {

                if (top == item.length - 1)

                                System.out.println("Stack is Full");

                else

                {

                        item[++top] = data;

                        System.out.println("Pushed Item :" + item[top]);

                }

        }

        void display()

        {
```

```
            if(top == -1)
                System.out.println("The stack is empty");
             else
            {
                System.out.println("The elements in the stack are : ");
                for(int i = top; i>=0; i--)
                    System.out.println(item[i]);
            }
        }
        int popItem()
        {
            if (top < 0)
            {
                System.out.println("Stack Underflow");
                return 0;
            }
            else
            {
                System.out.println("Pop Item : " + item[top]);
                return item[top--];
            }
        }
}
class StackExample
{
```

```
public static void main(String[] args) throws IOException
{
        Stack stk = new Stack(5);
        boolean yes=true;
        int choice;
        BufferedReader is = new BufferedReader(new InputStreamReader(System.in));
        do
        {
                System.out.println("1).Push\n2).Pop\n3).Display\n4).Exit\n\nEnter
Choice");
                choice = Integer.parseInt(is.readLine());
                switch(choice)
                {
                        case 1: System.out.println("Enter Push Item: ");
                                stk.pushItem(Integer.parseInt(is.readLine()));
                                break;
                        case 2: stk.popItem();
                                break;
                        case 3:stk.display();
                                break;
                        case 4: yes = false;
                                break;
                        default: System.out.println("Invalid Choice");
                }
        }while(yes==true);
}
```

```
}
```

Output:

```
1).Push
2).Pop
3).Display
4).Exit

Enter Choice
1
Enter Push Item:
2
Pushed Item :2
1).Push
2).Pop
3).Display
4).Exit

Enter Choice
1
Enter Push Item:
3
Pushed Item :3
1).Push
2).Pop
3).Display
4).Exit

Enter Choice
3
The elements in the stack are :
3
2
1).Push
2).Pop
3).Display
4).Exit

Enter Choice
2
Pop Item : 3
1).Push
2).Pop
3).Display
4).Exit

Enter Choice
```

3
The elements in the stack are :
2
1).Push
2).Pop
3).Display
4).Exit

2. Design a superclass called *Staff* with details as StaffId, Name, Phone, Salary. Extend this class by writing three subclasses namely *Teaching* (domain, publications), *Technical* (skills), and *Contract* (period). Write a Java program to read and display at least 3 *staff* objects of all three categories.
(a)Here you have to create 5 different class in the same folder….

(1)staff

```
public class staff
{
        int s_id;
        String s_name;
        long phone;
        double salary;
        String domain;
        String publication;
        String skill;
        int period;
}
```

(2) Teaching

```
import java.util.*;

public class teaching extends staff
{
        Scanner s1=new Scanner(System.in);
        public void inputteach()
        {
                s_id=s1.nextInt();
                s_name=s1.next();
                phone=s1.nextLong();
                salary=s1.nextDouble();
                domain=s1.next();
                publication=s1.next();
        }
        public void displayteach()
        {
                System.out.println("s_id is:"+s_id+"\n name is: "+s_name+"\n phone number
is:"+phone+"\nsalary is:"+salary+"\ndomain is:"+domain+"\npublication is:"+publication);
        }
}
```

(3)Technical

```
import java.util.*;
public class technical extends staff
{
        Scanner s1=new Scanner(System.in);
        public void inputtech()
```

```
        {
                s_id=s1.nextInt();
                s_name=s1.next();
                phone=s1.nextLong();
                salary=s1.nextDouble();
                skill=s1.next();
        }
        public void displaytech()
        {
                System.out.println("\ns_id is:"+s_id+"\nname is:"+s_name+"\nphone number
is:"+phone+"\nsalary is:"+salary+"\nskill is:"+skill);

        }

}
```

(4)Contract

```
import java.util.*;
public class contract extends staff
{
        Scanner s1=new Scanner(System.in);
        public void inputcon()
        {
                s_id=s1.nextInt();
                s_name=s1.next();
                phone=s1.nextLong();
                salary=s1.nextDouble();
                period=s1.nextInt();
        }
        public void displaycon()
        {
                System.out.println("\ns_id is:"+s_id+"\nname is:"+s_name+"\nphone number
is:"+phone+"\nsalary is:"+salary+"\nperiod is:"+period);

        }

}
```

(5) main

```
import java.util.*;
public class main
{
        public static void main(String[] args)
        {
```

```
                Scanner s1=new Scanner(System.in);
                int ch;
                System.out.println("Enter your choice");
                System.out.println("1.Teaching\n2.Technical\n3.Contract\n4.Exit\n");
                ch=s1.nextInt();
                switch(ch)
                {
                        case 1: System.out.println("Enter teaching staff details");
                                teaching ob1=new teaching();
                                ob1.inputteach();
                                ob1.displayteach();
                                break;
                        case 2: System.out.println("Enter technical staff detailse");
                                technical ob2=new technical();
                                ob2.inputtech();
                                ob2.displaytech();
                                break;
                        case 3: System.out.println("Enter contract staff details");
                                contract ob3=new contract();
                                ob3.inputcon();
                                ob3.displaycon();
                                break;
                }
        }
  }
```

Output:

```
Enter your choice
1.Teaching
2.Technical
3.Contract
4.Exit

1
Enter teaching staff details
201
raju
8765743567
15000
testing
pearson
s_id is:201
name is:raju
phone number is:8765743567
salary is:15000.0
domain is:testing
publication is:pearson
```

2(b) Write a Java class called *Customer* to store their name and date_of_birth. The date_of_birth format should be dd/mm/yyyy. Write methods to read customer data as <name, dd/mm/yyyy> and display as <name, dd, mm, yyyy> using StringTokenizer class considering the delimiter character as "/".

```
import java.util.*;
public class customer
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		 System.out.println("enter name and dob");
		String str=sc.nextLine();
		StringTokenizer st=new StringTokenizer(str,"/ ",true);
		while(st.hasMoreTokens())
			System.out.println(st.nextToken()+",");

		sc.close();
	}
}
```

space

Output:

```
enter name and dob
sanjay 12/03/1990
sanjay,
 ,
12,
/,
03,
/,
1990,
```

3(a) Write a Java program to read two integers *a* and*b*. Compute *a*/*b* and print, when *b* is not zero. Raise an exception when *b* is equal to zero.

```
import java.util.*;
public class division
{
	public static void main(String[] args)
	{
		int a,b;
		double A;
		Scanner s=new Scanner(System.in);
		System.out.println("enter a value");
		a=s.nextInt();
		System.out.println("enter b value");
		b=s.nextInt();
		try
		{
			A=a/b;
			System.out.println("A is "+A);
		}
		catch(ArithmeticException ae)
		{
			System.out.println(ae);
		}
	}
 }
```

Output:

```
enter a value
40
enter b value
10
A is 4.0

enter a value
40
enter b value
0
java.lang.ArithmeticException: / by zero
```

3(b) Write a Java program that implements a multi-thread application that has three threads. First thread generates a random integer for every 1 second; second thread computes the square of the number and prints; third thread will print the value of cube of the number.

Here you have to create 4 classes within the same folder….

(1)Cube

```
import java.util.*;
public class cube implements Runnable
{
	public int x;
	public cube(int x)
	{
		this.x=x;

	}
	public void run()
	{
		System.out.println("From third Thread cube of "+x+" is "+x*x*x);

	}

}
```

(2)F

```
import java.util.*;
public class F extends Thread
{
	public void run()
	{
		int num=0;
		Random r=new Random();
		try
		{
			for(int i=0;i<5;i++)
			{
				num=r.nextInt(100);
				System.out.println("Main thread started and Generated random
number is"+num);
				Thread t2=new Thread(new square(num));
				t2.start();
				Thread t3=new Thread(new cube(num));
				t3.start();
				Thread.sleep(1000);
```

```
                        System.out.println("_____");

                    }
                }
                catch(Exception E)
                {
                    System.out.println(E.getMessage());
                }
        }
}
```

(3)MultiThread

```
import java.util.*;
public class MultiThread
{
        public static void main(String[] args)
        {
                F firstThread=new F();
                Thread t1=new Thread(firstThread);
                t1.start();
        }
}
```

(4)square

```
import java.util.*;
public class square implements Runnable
{
        public int x;
        public square(int x)
        {
                this.x=x;
        }
        public void run()
        {
                System.out.println("From Second Thread square of "+x+" is "+x*x);

        }
}
```

Output:

Main thread started and Generated random number is84
From Second Thread square of 84 is 7056
From third Thread cube of 84 is 592704

Main thread started and Generated random number is84
From Second Thread square of 84 is 7056
From third Thread cube of 84 is 592704

Main thread started and Generated random number is85
From Second Thread square of 85 is 7225
From third Thread cube of 85 is 614125

Main thread started and Generated random number is69
From Second Thread square of 69 is 4761
From third Thread cube of 69 is 328509

Main thread started and Generated random number is96
From Second Thread square of 96 is 9216
From third Thread cube of 96 is 884736

---

4. Sort a given set of *n* integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of *n*> 5000 and record the time taken to sort. Plot a graph of the time taken versus *n*on graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divide and- conquer method works along with its time complexity analysis: worst case, average case and best case.

```
import java.util.*;

class Quick
{
	static int a[]=new int[20000];
	static int j;
	static Scanner s=new Scanner(System.in);
	static void quicksort(int low,int high)
	{
		if(low>high)
			return;
		partition(low,high);
		quicksort(low,j-1);
		quicksort(j+1,high);
	}

	static void partition(int low,int high)
	{
		int l=low,h=high,temp,pivot;
		pivot=a[low];
		while(l<h)
		{
			while(a[l]<=pivot&&l<h)
				l++;
			while(a[h]>pivot)
				h--;

			if(l<h)
			{
				temp=a[l];
				a[l]=a[h];
				a[h]=temp;
			}
		}/*while*/

		a[low]=a[h];
		a[h]=pivot;
		j=h;
	}
```

```
        public static void main(String args[])
        {

                int i,n;
                Random t = new Random();

                System.out.println("Enter the size of the array\n");
                n=s.nextInt();

                System.out.println("The array elements are:\n");
                for(i=0;i<n;i++)
                {
                        a[i]=t.nextInt(10000);
                        System.out.print(a[i]+"\t");

                }
                System.out.println();
                long startTime = System.currentTimeMillis();
                quicksort(0,n-1);
                long endTime = System.currentTimeMillis();

                System.out.println("Elements in sorted order\n");
                for(i=0;i<n;i++)
                        System.out.print(a[i]+"\t");
                System.out.println();
                System.out.println("It took " + (endTime - startTime) + " Milliseconds");
        }/*End of main()*/
}
```

Output:

```
Enter the size of the array
5
The array elements are:
8835    4168    4565    6868    2008
Elements in sorted order
2008    4168    4565    6868    8835
It took 0 Milliseconds
```

5. Sort a given set of *n* integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of *n*> 5000, and record the time taken to sort. Plot a graph of the time taken versus *n*on graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divide and- conquer method works along with its time complexity analysis: worst case, average case and best case.

```
import java.util.*;
public class Merge
{
	static int a[]=new int[20000];
	static int c[]=new int[20000];

	static int j;
	static Scanner s=new Scanner(System.in);
	static void merge(int l,int m,int h)
	{
		int i,j,k;
		i=l;
		j=m+1;
		k=l;

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

		while(i<=m)
			c[k++]=a[i++];
		while(j<=h)
			c[k++]=a[j++];
		for(i=l;i<=h;i++)
			a[i]=c[i];
	}

	static void mergesort(int low,int high)
	{
		int m;
		if(low<high)
		{
			m=(low+high)/2;
			mergesort(low,m);
			mergesort(m+1,high);
			merge(low,m,high);
```

```
		}
	}

	public static void main(String []args)
	{
		int i,n;
		Random t = new Random();

		System.out.println("Enter the size of the array\n");
		n=s.nextInt();

		System.out.println("The array elements are:\n");
		for(i=0;i<n;i++)
		{
			a[i]=t.nextInt(10000);
			System.out.print(a[i]+"\t");

		}
		System.out.println();
		long startTime = System.currentTimeMillis();

		mergesort(0,n-1);
		long endTime = System.currentTimeMillis();

		System.out.println("Elements in sorted order\n");
		for(i=0;i<n;i++)
			System.out.print(a[i]+"\t");
		System.out.println();

		System.out.println("It took " + (endTime - startTime) + " Milliseconds");
	}
}
```

Output:

Enter the size of the array

5
The array elements are:
9905 9092 1017 6209 777
Elements in sorted order
777 1017 6209 9092 9905
It took 0 Milliseconds

6. Implement in Java, the 0/1 Knapsack problem using (a) Dynamic Programming method (b) Greedy method.

```
import java.util.*;
public class knapsack
{
	static float[] weight=new float[20];
	static float value[]=new float[20];
	static float ratio[]=new float[20];
	static float x[]=new float[20];
	static float[][] v=new float[50][50];
	static float w;
	static int i,j,n;
	static Scanner s=new Scanner(System.in);

	static void Gknapsack()
	{
		float tp=0,u;
		u=w;
		for(i=1;i<n;i++)
			x[i]=0;
		for(i=1;i<n;i++)
		{
			if(weight[i]>u)
				break;
			else
			{
				x[i]=1;
				tp=tp+value[i];
				u=u-weight[i];
			 }
		}
		if(i<=n)
			x[i]=u/weight[i];
		tp=tp+(x[i]*value[i]);
		System.out.println("weight\tvalue\tx");
		for(i=1;i<=n;i++)
			System.out.println(weight[i]+"\t"+value[i]+"\t"+x[i]);
		System.out.println("Maximum profit="+tp);
	}
	static void Dknapsack()
	{
		for(i=0;i<=n;i++)
			for(j=0;j<=w;j++)
			{
				if(i==0||j==0)
					v[i][j]=0;
```

```
                        else if(weight[i]>j)
                                v[i][j]=v[i-1][j];
                        else
                                v[i][j]=maximum(v[i-1][j],v[i-1][j-(int) weight[i]]+value[i]);
        }
        System.out.println("solution Matrix");
        for(i=0;i<=n;i++)
        {
                for(j=0;j<=w;j++)
                        System.out.print(v[i][j]+"\t");
                System.out.println();
        }
        System.out.println("Maximum profit="+v[n][(int) w]);
}
static float maximum(float a,float b)
{
        if(a>b)
        return a;
        return b;
}

static void printvector()
{
        for(i=1;i<=n;i++)
                x[i]=0;
        i=n;
        j=(int) w;
        while(i!=0&&j!=0)
        {
                 if(v[i][j]!=v[i-1][j])
                {
                        x[i]=1;
                        j=j-(int) weight[i];
                }
                i=i-1;
        }
        System.out.println("item included");
        for(i=1;i<=n;i++)
                System.out.println("x["+i+"]="+x[i]);

}

public static void main(String args[])
{
        float temp;
        System.out.println("Enter the number of objects");
        n=s.nextInt();
```

```
                System.out.println("Enter the weight of each object");
                for(i=1;i<=n;i++)
                        weight[i]=s.nextFloat();
                System.out.println("Enter the value of each object");
                for(i=1;i<=n;i++)
                        value[i]=s.nextFloat();
                System.out.println("Enter the capacity of the knapsack");
                w=s.nextFloat();
                for(i=1;i<=n;i++)
                        ratio[i]=value[i]/weight[i];
                for(i=1;i<=n;i++)
                   for(j=1;j<=n;j++)
                        if(ratio[i]>ratio[j])
                        {
                                temp=ratio[j];
                                ratio[j]=ratio[i];
                                ratio[i]=temp;

                                temp=weight[j];
                                weight[j]=weight[i];
                                weight[i]=temp;

                                temp=value[j];
                                value[j]=value[i];
                                value[i]=temp;
                        }
                System.out.println("greedy knapsack solution");
                Gknapsack();
                System.out.println("dynamic knapsack solution");
                Dknapsack();
                printvector();

        }
}
```

Output:

```
Enter the number of objects
4
Enter the weight of each object
2 1 3 2
Enter the value of each object
12 10 20 15
Enter the capacity of the knapsack
5
greedy knapsack solution
weight  value   x
1.0     10.0    1.0
2.0     15.0    1.0
3.0     20.0    0.6666667
2.0     12.0    0.0
Maximum profit=38.333336
dynamic knapsack solution
solution Matrix
0.0     0.0     0.0     0.0     0.0     0.0
0.0     10.0    10.0    10.0    10.0    10.0
0.0     10.0    15.0    25.0    25.0    25.0
0.0     10.0    15.0    25.0    30.0    35.0
0.0     10.0    15.0    25.0    30.0    37.0
Maximum profit=37.0
item included
x[0]=0.0
x[1]=1.0
x[2]=1.0
x[3]=0.0
x[4]=1.0
```

7. From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm. Write the program in Java.

```
import java.util.*;
public class Dijkstra
{
	static int MAX=20, n,i,j,source,inf=999;
	static Scanner s=new Scanner(System.in);
	static int c[][]=new int[MAX][MAX];
	static int dist[]=new int[MAX];
	static boolean v[]=new boolean[MAX];


	static void shortest_path()
	{
		int i,j,w,num,u;
		for(i=1;i<=n;i++)
		{
			v[i]=false;
			dist[i]=c[source][i];
		}
		v[source]=true;
		num=2;
		while(num<=n)
		{
			u=choose();
			v[u]=true;
			num++;
			for(w=1;w<=n;w++)
				if( dist[u]+c[u][w]<dist[w] && !v[w])
					dist[w]=dist[u]+c[u][w];
		}
	}

	static int choose()
	{
		int min,w,j=1;
		min=inf;
		for(w=1;w<=n;w++)
			if( dist[w]<min && v[w]==false )
			{
				min=dist[w];
				j=w;
			}
		return j;
	}
	public static void main(String args[])
```

```
        {
                int i,j;

                System.out.println("Enter the no. of vertices\n");
                n=s.nextInt();
                System.out.println("Enter the cost adjacency matrix\n");
                for(i=1;i<=n;i++)
                        for(j=1;j<=n;j++)
                                c[i][j]=s.nextInt();
                System.out.println("Enter the starting vertex\n");
                source=s.nextInt();
                shortest_path();
                for(j=1;j<=n;j++)
                        if(j!=source)
                                System.out.println("Shortest path vertex "+source+" to vertex "+j+" is "+ dist[j]);

        }
}
```

Output:

```
Enter the no. of vertices

5
Enter the cost adjacency matrix

0 3 999 7 999
3 0 4 2 999
999 4 0 5 6
7 2 5 0 4
999 999 6 4 0
Enter the starting vertex

1
Shortest path vertex 1 to vertex 2 is 3
Shortest path vertex 1 to vertex 3 is 7
Shortest path vertex 1 to vertex 4 is 5
Shortest path vertex 1 to vertex 5 is 9
```

8. Find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm. Use Union-Find algorithms in your program.

```
import java.util.*;
 class edge
 {
        int u,v,cost;
        edge(int i,int j,int k)
        {
           u=i;
           v=j;
           cost=k;
        }
 }
public class Kruskal
{
        static int n,m,v,inf=999;
        static int t[][]=new int[20][20];
        static edge e[]=new edge[20];
        static int parent[]=new int[20];
        static Scanner s=new Scanner(System.in);

        static void kruskal()
        {
                int count=0,sum=0,pos,u,v;
                int i,j,k=0;
                for(i=0;i<n;i++)
                        parent[i]=i;

                while(count!=n-1)
                {
                        pos=minimum();
                        if(pos==-1)
                                break;
                        u=e[pos].u;
                        v=e[pos].v;
                        i=find(u);
                        j=find(v);
                        if(i!=j)
                        {
                                t[k][0]=u;
                                t[k][1]=v;
                                k++;
                                count++;
                                sum+=e[pos].cost;
                                union_if(i,j);
                        }
```

```
                e[pos].cost=inf;
        }
        if(count==n-1)
        {
                System.out.println("Spanning tree exists\n");
                System.out.println("Spanning tree is shown below\n");
                for(i=0;i<n-1;i++)
                        System.out.println("<"+t[i][0]+","+t[i][1]+">");
                System.out.println("cost of spanning tree="+sum);
        }
        else
                System.out.println("Spanning tree doesn't exists\n");
}

static int minimum()
{
        int small,i,pos=-1;
        small=inf;
        for(i=0;i<m;i++)
                if(e[i].cost<small)
                {
                        small=e[i].cost;
                        pos=i;
                }
        return pos;
}

static int find(int v)
{
        while(parent[v]<v)
                v=parent[v];
        return v;
}
static void union_if(int i,int j)
{
        if(i<j)
                parent[j]=i;
        else
                parent[i]=j;
}
public static void main(String args[])
{
        int k,i,j,cost;
        System.out.println("Enter the no. of nodes\n");
        n=s.nextInt();
        System.out.println("Enter the no. of edges\n");
        m=s.nextInt();
```

```
                System.out.println("Enter the edge list\n");
                for(k=0;k<m;k++)
                {
                        System.out.println("Enter the edge & cost in form of u,v,w\n");

                        i=s.nextInt();
                        j=s.nextInt();
                        cost=s.nextInt();
                        e[k]=new edge(i,j,cost);

                }
                kruskal();

        }
}
```

Output:

Enter the no. of nodes
4
Enter the no. of edges
4
Enter the edge list

Enter the edge & cost in form of u,v,w

1 2 1

Enter the edge & cost in form of u,v,w

1 3 5

Enter the edge & cost in form of u,v,w
1 4 2

Enter the edge & cost in form of u,v,w

3 4 3

Spanning tree exists

Spanning tree is shown below

<1,2>
<1,4>
<3,4>
cost of spanning tree=6

9. Find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.

```
import java.util.*;
class Prim
{

	static int MAX=50,n,i,j,INF=999,flag=1;
	static boolean v[]=new boolean[MAX];
	static int mincost=0;
	static Scanner s =new Scanner(System.in);

	static int c[][]=new int[MAX][MAX];
	static int t[]=new int[2*(MAX-1)];
	static void prims()
	{

		int i,j,k,s,v1=0,v2=0,min;
		for(i=0;i<n;i++)
			v[i]=false;
		v[0]=true;
		k=0;
		s=0;
		k++;
		while(k<n)
		{
			min=INF;
			for(i=0;i<n;i++)
				for(j=0;j<n;j++)
					if(v[i]==true && v[j]==false && c[i][j]<min)
					{
						min=c[i][j];
						v1=i;
						v2=j;
					}
			mincost+=min;
			if(min==INF)
			{
				System.out.println("Spanning tree doesn't exists\n");
				flag=0;
				break;
			}
			v[v2]=true;
			k++;
			t[s++]=v1;
			t[s++]=v2;
```

```
            }
        }


        public static void main(String args[])
        {
            int i,j;

            System.out.println("Enter the no. of vertices\n");
            n=s.nextInt();
            System.out.println("Enter the cost adjacency matrix\n");
            for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                    c[i][j]=s.nextInt();
            prims();
            if(flag==1)
            {
                System.out.println("Shortest path spanning tree\n");
                for(i=0;i<2*(n-1);i+=2)
                    System.out.println("<"+(t[i]+1)+","+(t[i+1]+1)+">");
                System.out.println("Minimum cost="+mincost);
            }

        }
}
```

Output:

Enter the no. of vertices

4

Enter the cost adjacency matrix

0 1 5 2
1 0 999 999
5 999 0 3
2 999 3 0

Shortest path spanning tree

<1,2>
<1,4>
<4,3>

Minimum cost=6

10. Write Java programs to
(a) Implement All-Pairs Shortest Paths problem using Floyd's algorithm.
(b) Implement Travelling Sales Person problem using Dynamic programming.

```
import java.util.*;
public class Floyd
{

	static Scanner s=new Scanner(System.in);
	static int c[][]=new int[10][10];
	static int n;
	static int min(int a,int b)
	{
		if(a<b)
			return a;
		return b;
	}
	static void floyds()
	{
		int i,j,k;
		for(k=1;k<=n;k++)
			for(i=1;i<=n;i++)
				for(j=1;j<=n;j++)
					c[i][j]=min(c[i][j],c[i][k]+c[k][j]);
	}
	public static void main(String args[])
	{
		int i,j;
		System.out.println("Enter the no of vertices\n");
		n=s.nextInt();
		System.out.println("Enter the cost adjacency matrix,0-self loop and 999-no
edge\n");
		for(i=1;i<=n;i++)
			for(j=1;j<=n;j++)
				c[i][j]=s.nextInt();
		floyds();
		System.out.println("Shortest path matrix\n");
		for(i=1;i<=n;i++)
		{
			for(j=1;j<=n;j++)
				System.out.print("\t"+c[i][j]);
			System.out.println();
		}
	}
}
```

Output:

Enter the no of vertices

4

Enter the cost adjacency matrix,0-self loop and 999-no edge

0 999 3 999
2 0 999 999
999 7 0 1
6 999 999 0

Shortest path matrix

0	10	3	4
2	0	5	6
7	7	0	1
6	16	9	0

10 b

```
import java.util.*;
public class Travel
{
	static Scanner s=new Scanner(System.in) ;
	static int a[][]=new int[10][10];
	static int visited[]=new int[10];
	static int n,cost=0;

	static void tsp(int city)
	{
		int i,ncity;
		visited[city]=1;
		System.out.print((city)+" -->");
		ncity=least(city);
		if(ncity==999)
		{
			ncity=1;
			System.out.print(ncity);
			cost+=a[city][ncity];
			return;
		}
		tsp(ncity);
	}

	static int least(int c)
	{
		int i,nc=999;
		int min=999,kmin=1;
		for(i=1;i <= n;i++)
		{
			if((a[c][i]!=0)&&(visited[i]==0))
				if(a[c][i] < min)
				{
					min=a[i][0]+a[c][i];
					kmin=a[c][i];
					nc=i;
				}
		}
		if(min!=999)
			cost+=kmin;
		return nc;
	}

	public static void main(String args[])
	{
```

```
                int i,j;
                System.out.println("Enter No. of Cities: ");
                n=s.nextInt();
                System.out.println("Enter Cost Matrix\n");
                for(i=1;i <= n;i++)
                        for( j=1;j <= n;j++)
                                a[i][j]=s.nextInt();
                for(i=1;i <= n;i++)
                        visited[i]=0;
                System.out.println("\nThe Path is:");
                tsp(1);
                System.out.println("\n\nMinimum cost:"+cost);

        }

}
```

Output:

```
Enter No. of Cities:
4
Enter Cost Matrix

0 2 5 7
2 0 8 3
5 8 0 1
7 3 1 0

The Path is:
1 -->2 -->4 -->3 -->1
Minimum cost:11
```

11. Design and implement in Java to find a subset of a given set S = {Sl, S2,.....,Sn} of *n* positive integers whose SUM is equal to a given positive integer *d*. For example, if S ={1, 2, 5, 6, 8} and *d*= 9, there are two solutions {1,2,6}and {1,8}. Display a suitable message, if the given problem instance doesn't have a solution.

```
import java.util.*;
public class Subset
{
	static Scanner s=new Scanner(System.in) ;
	static boolean inc[]=new boolean[50];
	static int w[]=new int[50];
	static int sum,n,flag=0;
	static boolean promising(int i,int wt,int total)
	{
		return(( (wt+total)>=sum)&&((wt==sum)||((wt+w[i+1])<=sum)));
	 }
	static void sumset(int i,int wt,int total)
	{
		int j;
		if(promising(i,wt,total))
		{
			if(wt==sum)
			{
				flag=1;
				System.out.print("\n{\t");
				for (j=0;j<=i;j++)
					if(inc[j])
						System.out.print(w[j]+"\t");
				System.out.println("}");
			}
			else
			{
				inc[i+1]=true;
				sumset(i+1,wt+w[i+1],total-w[i+1]);
				inc[i+1]=false;
				sumset(i+1,wt,total-w[i+1]);
			}
		}
	}
	public static void main(String args[])
	{
	    int i,j,n,temp,total=0;
	    System.out.println("Enter the number of elements in set");
	    n=s.nextInt();
	    System.out.println("Enter "+ n +"numbers to the set");
	    for (i=0;i<n;i++)
	    {
```

```
                w[i]=s.nextInt();
                total+=w[i];
            }
            System.out.println(" Input the sum value to create sub set");
            sum=s.nextInt();
            for (i=0;i<=n;i++)
                for (j=0;j<n-1;j++)
                    if(w[j]>w[j+1])
                    {
                        temp=w[j];
                        w[j]=w[j+1];
                        w[j+1]=temp;
                    }
            System.out.println("\n The given "+n+" numbers in ascending order");
            for (i=0;i<n;i++)
                System.out.print("\t"+w[i]);
            System.out.println();

            if((total<sum)||sum<w[0])
                System.out.println("\n Subset construction is not possible");
            else
            {
                for (i=0;i<n;i++)
                    inc[i]=false;
                System.out.println("The solution using backtracking is");
                sumset(-1,0,total);
            }
            if(flag == 0)
                System.out.println("\n Subset construction is not possible");
        }
}
```

Output:

```
Enter the number of elements in set
5
Enter 5numbers to the set
1 2 5 6 8
 Input the sum value to create sub set
9

 The given 5 numbers in ascending order
      1      2      5      6      8
The solution using backtracking is

{     1     2     6     }

{     1     8     }
```

12. Design and implement in Java to find all Hamiltonian Cycles in a connected undirected Graph G of *n* vertices using backtracking principle.

```
import java.util.*;
public class Ham
{
	static Scanner s=new Scanner(System.in);
	static int graph[][]=new int[10][10];
	static int path[]=new int[10];
	static int n;
	static void printSolution()
	{
		System.out.println("Solution Exists:");
		System.out.println(" Following is one Hamiltonian Cycle ");
		for (int i = 0; i < n; i++)
			System.out.print((path[i]+1)+"  ");
		System.out.println(path[0]+1);;
	}
	static boolean isSafe(int v, int pos)
	{
		if (graph [path[pos-1]][v] == 0)
			return false;
		for (int i = 0; i < pos; i++)
			if (path[i] == v)
				return false;
		return true;
	}
	/* solve hamiltonian cycle problem */
	static boolean hamCycleUtil( int pos)
	{
		if (pos == n)
		{
			if (graph[ path[pos-1] ][ path[0] ] == 1)
				return true;
			else
				return false;
		}

		for (int v = 1; v < n; v++)
		{
			if (isSafe(v,  pos))
			{
				path[pos] = v;
				if (hamCycleUtil (pos+1) == true)
					return true;
				path[pos] = -1;
			}
```

```
            }
            return false;
        }
        /* solves the Hamiltonian Cycle problem using Backtracking.*/
        static boolean hamCycle()
        {
            for (int i = 0; i < n; i++)
                path[i] = -1;
            path[0] = 0;
            if (hamCycleUtil( 1) == false)
            {
                System.out.println("\nSolution does not exist");
                return false;
            }
            printSolution();
            return true;
        }
        /* Main */

        public static void main(String args[])
        {
            int i,j;
            System.out.println("Enter the number of vertices");
            n=s.nextInt();
            System.out.println("Enter the cost adjacency matrix");
            for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                    graph[i][j]=s.nextInt();
            hamCycle();

            return ;
        }

}
```

Output:

```
Enter the number of vertices
6
Enter the cost adjacency matrix
0 1 1 1 0 0
1 0 1 0 0 1
1 1 0 1 1 0
1 0 1 0 1 0
0 0 1 1 0 1
0 1 0 0  1 0
Solution Exists:
 Following is one Hamiltonian Cycle
1  2  6  5  3  4  1
```

Viva Questions

1. **What is an Algorithm?**
   An algorithm is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.

2. **What is Exact and Approximation algorithm?**
   The principal decision to choose solving the problem exactly is called exact algorithm. The principal decision to choose solving the problem approximately is called Approximation algorithm.
3. **What is Algorithm Design Technique?**
   An algorithm design technique is a general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing.
4. **Define Pseudo code.**
   A Pseudo code is a mixture of a natural language and programming language like constructs. A pseudo code is usually more precise than a natural language, and its usage often yields more succinct algorithm descriptions.
5. **Explain Algorithm's Correctness**
   To prove that the algorithm yields a required result for every legitimate input in a finite amount of time. Example: Correctness of Euclid's algorithm for computing the greatest common divisor stems from correctness of the equality gcd (m, n) = gcd (n, m mod n).
6. **What is Efficiency of algorithm?**
   Efficiency of an algorithm can be precisely defined and investigated with mathematical rigor. There are two kinds of algorithm efficiency
   i. Time Efficiency – Indicates how fast the algorithm runs
   ii. Space Efficiency – Indicates how much extra memory the algorithm needs.
7. **What is generality of an algorithm?**
   It is a desirable characteristic of an algorithm. Generality of the problem the algorithm solves is sometimes easier to design an algorithm for a problem posed in more general terms.
8. **What is algorithm's Optimality?**
   Optimality is about the complexity of the problem that algorithm solves. What is the minimum amount of effort any algorithm will need to exert to solve the problem in question is called algorithm's Optimality.
9. **What do you mean by ″Sorting" problem?**
   The sorting problem asks us to rearrange the items of a given list in ascending order (or descending order)
10. **What do you mean by ″Searching" problem?**
    The searching problem deals with finding a given value, called a search key, in a given set.
11. **What do you mean by ″Worst case-Efficiency" of an algorithm?**
    The ″Worst case-Efficiency" of an algorithm is its efficiency for the worst-case input of size n, which is an input (or inputs) of size n for which the algorithm runs the longest among all possible inputs of that size. Ex: if you want to sort a list of numbers in

ascending order when the numbers are given in descending order. In this running time will be the longest.

12. What do you mean by ″Best case-Efficiency" of an algorithm?

The ″Best case-Efficiency" of an algorithm is its efficiency for the Best-case input of size n, which is an input (or inputs) of size n for which the algorithm runs the fastest among all possible inputs of that size. Ex: if you want to sort a list of numbers in ascending order when the numbers are given in ascending order. In this running time will be the smallest.

13. Define the ″Average-case efficiency" of an algorithm?

The ″Average-case efficiency" of an algorithm is its efficiency for the input of size n, for which the algorithm runs between the best case and the worst case among all possible inputs of that size.

14. What do you mean by "Amortized efficiency"?

The "Amortized efficiency" applies not only a single run of an algorithm but rather to a sequence of operations performed on the same data structure. It turns out that in some situations a single operation can be expensive ,but the total time for an entire sequence of n such operations is always significantly better than the worst case efficiency of that single operation multiplied by n. This is known as "Amortized efficiency".

15. How to measure the algorithm's efficiency?

It is logical to investigate the algorithm's efficiency as a function of some parameter n indicating the algorithm's input size. Example: It will be the size of the list for problems of sorting, searching, finding the list's smallest element, and most other problems dealing with lists.

16. What is called the basic operation of an algorithm?

The most important operation of the algorithm is the operation contributing the most to the total running time is called basic operation of an algorithm.

17. How to measure an algorithm's running time?

Let Cop be the time of execution of an algorithm's basic iteration on a particular computer and let C (n) be the number of times this operation needs to be executed for this algorithm. Then we can estimate the running time T(n) of a program implementing this algorithm on that computer by the formula $T(n) \approx Cop\ C(n)$.

18. Define order of growth.

The efficiency analysis framework concentrates on the order of growth of an algorithm's basic operation count as the principal indicator of the algorithm's efficiency. To compare and rank such orders of growth we use three notations

i. O (Big oh) notation ii.Ω (Big Omega) notation & iii. Θ (Big Theta) notation

19. Define Big oh notation

A function t(n) is said to be in O(g(n)) denoted $t(n) \varepsilon O(g(n))$, if t(n) is bounded above by some constant multiple of g(n) for all large n, i.e., if there exist some positive constant c and some non negative integer n0 such that $T(n) < c\ g(n)$ for $n > n0$

20. Define Ω notation

A function t(n) is said to be in Ω (g(n)), denoted $t(n) \in \Omega(g(n))$, if t(n) is bounded below by some positive constant multiple of g(n) for all large n, i.e., if there exist some positive constant c and some non negative integer n0 such that $T(n) < c\ g(n)$ for $n > n0$

21. Define Θ – notation

A function t(n) is said to be in Θ(g(n)), denoted $t(n) \in \Theta(g(n))$, if t(n) is bounded both above and below by some positive constant multiples of g(n) for all large n, i.e., if

there exist some positive constant c1 and c2 and some non negative integer n0 such that c2 g (n) < t (n) < c1 g(n) for n > n0

22. What is the use of Asymptotic Notations?

The notations O, Ω and Θ and are used to indicate and compare the asymptotic orders of growth of functions expressing algorithm efficiencies.

23. Explain the greedy method.

Greedy method is the most important design technique, which makes a choice that looks best at that moment. A given 'n' inputs are required us to obtain a subset that satisfies some constraints that is the feasible solution. A greedy method suggests that one can device an algorithm that works in stages considering one input at a time.

24. Define feasible and optimal solution.

Given n inputs and we are required to form a subset such that it satisfies some given constraints then such a subset is called feasible solution. A feasible solution either maximizes or minimizes the given objective function is called as optimal solution.

25. What is a minimum cost spanning tree?

A spanning tree of a connected graph is its connected acyclic sub-graph that contains all vertices of a graph. A minimum spanning tree of a weighted connected graph is its spanning tree of the smallest weight where weight of the tree is the sum of weights on all its edges.

26. Specify the algorithms used for constructing Minimum cost spanning tree.

a) Prim's Algorithm b) Kruskal's Algorithm

27. State single source shortest path algorithm (Dijkstra's algorithm).

For a given vertex called the source in a weigted connected graph,find shortest paths to all its other vertices. Dijikstra's algorithm applies to graph with non-negative weights only.

28. What is Knapsack problem?

A bag or sack is given capacity and n objects are given. Each object has weight wi and profit pi .Fraction of object is considered as xi (i.e) 0<=xi<=1 .If fraction is 1 then entire object is put into sack. When we place this fraction into the sack we get wixi and pixi.

29. Write the difference between the Greedy method and Dynamic programming.

Greedy method	Dynamic programming
Only one sequence of decision is generated.	Many number of decisions are generated.
It does not guarantee to give an optimal solution always.	It definitely gives an optimal solution always.

30. **state efficiency of prim's algorithm**.

O(|v|2) (WEIGHT MATRIX AND PRIORITY QUEUE AS UNORDERED ARRAY)
O(|E| LOG|V|) (ADJACENCY LIST AND PRIORITY QUEUE AS MIN-HEAP)

31. State Kruskal Algorithm.

The algorithm looks at a MST for a weighted connected graph as an acyclic sub-graph with |v|-1 edges for which the sum of edge weights is the smallest.

32. state efficiency of Dijkstra's algorithm.

O(|v|2) (WEIGHT MATRIX AND PRIORITY QUEUE AS UNORDERED ARRAY)
O(|E| LOG|V|) (ADJACENCY LIST AND PRIORITY QUEUE AS MIN-HEAP)

33. Define dynamic programming.

Dynamic programming is an algorithm design method that can be used when a solution to the problem is viewed as the result of sequence of decisions. It is technique for solving problems with overlapping sub-problems.

34. What are the drawbacks of dynamic programming?

- Time and space requirements are high, since storage is needed for all level.
- Optimality should be checked at all levels.

35. Define multistage graph

A multistage graph G =(V,E) is a directed graph in which the vertices are partitioned in to K>=2 disjoint sets Vi,1<=i<=k.The multi stage graph problem is to find a minimum cost paths from s(source) to t(sink) Two approach(forward and backward)

36. Define All pair shortest path problem

Given a weighted connected graph, all pair shortest path problem asks to find the lengths of shortest paths from each vertex to all other vertices.

37. Define Distance matrix

Recording the lengths of shortest path in n x n matrix is called Distance matrix(D)

38. Define floyd's algorithm

To find all pair shortest path. The algorithm computes the distance matrix of a weighted graph with n vertices through series of n by n matrices :D(0)...D(k-1),D(k).....D(n)

39. State the time efficiency of floyd's algorithm

O(n3) It is cubic

40. Define OBST

If probabilities of searching for elements of a set are known then finding optimal BST for which the average number of comparisons in a search is smallest possible.

41. Define catalan number

The total number of binary search trees with n keys is equal to nth catalan number C(n)=(2n to n) 1/(n+1) for n>0,c(0)=1

42. State time and space efficiency of OBST

SPACE EFFICIENCY :QUADRATIC TIME EFFICIENCY : CUBIC.

43. What are the requirements that are needed for performing Backtracking?

To solve any problem using backtracking, it requires that all the solutions satisfy a complex set of constraints. They are: i. Explicit constraints. ii. Implicit constraints.

44. Define explicit constraint.

They are rules that restrict each x to take on values only from a give set. They depend on the particular instance I of the problem being solved. All tuples that satisfy the explicit constraints define a possible solution space.

45. Define implicit constraint.

They are rules that determine which of the tuples in the solution space of I satisfy the criteria function. It describes the way in which the x must relate to each other.

46. Define state space tree.

The tree organization of the solution space is referred to as state space tree.

47. Define Branch-and-Bound method.

The term Branch-and-Bound refers to all the state space methods in which all children of the E-node are generated before any other live node can become the E- node.

48. State 8 – Queens problem.

The problem is to place eight queens on a 8 x 8 chessboard so that no two queen "attack" that is, so that no two of them are on the same row, column or on the diagonal.

49. **State Sum of Subsets problem**.

Given n distinct positive numbers usually called as weights , the problem calls for finding all the combinations of these numbers whose sums are m.

50. What is promising and non-promising nodes?

A node in a state space tree is said to be promising if it corresponds to a partially constructed solution from which a complete solution can be obtained. The nodes which are not promising for solution in a state space tree are called non-promising nodes.

51. Write formula for bounding function in Knapsack problem

In knapsack problem upper bound value is computed by the formula

UB = v + (W-w) * (vi+1/wi+1)

52. Write about traveling salesperson problem.

Let g = (V, E) be a directed. The tour of G is a directed simple cycle that includes every vertex in V. The cost of a tour is the sum of the cost of the edges on the tour. The traveling salesperson problem is to find a tour of minimum cost.

53. Write some applications of traveling salesperson problem.

- ➔ Routing a postal van to pick up mail from boxes located at n different sites.
- ➔ Using a robot arm to tighten the nuts on some piece of machinery on an assembly line.
- ➔ Production environment in which several commodities are manufactured on the same set of machines.

54. Give the time complexity and space complexity of traveling salesperson problem.

Time complexity is O ($n^2 2^n$). Space complexity is O ($n 2^n$).

55. What is class P and NP?

P is set of all decision problems solvable by deterministic algorithms in polynomial time. NP is set of all decision problems solvable by non deterministic algorithms in polynomial time.

56. Define NP-Hard and NP-Complete problems

Problem L is NP-Hard if and only if satisfiability reduces to L. A Problem L is NP-Complete if and only if L is NP-Hard and L belongs to NP.