SRINIVAS UNIVERSITY

INSTITUTE OF ENGINEERING & TECHNOLOGY

MUKKA, MANGALURU

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

A LAB MANUAL ON

DESIGN AND ANALYSIS OF ALGORITHMS

2023-2024

Subject Code: 22SCS044

Prepared by:

Ms. Swarna H.R.
Asst. Professor

NAME: ______________________________

USN: ______________________________

SEMESTER: ______________________________

SECTION/ BRANCH: ______________________________

INSTITUTE VISION AND MISSION

Vision

To be a trendsetter among universities and build students who emerge as leaders with competence, conscience and compassion by empowering them with sound education and high standards of ethical and professional behaviour enabling them to build and promote a more humane, just and sustainable world for future generations.

Mission

Our mission is to provide an exceptional learning environment where students can develop and enhance their leadership and teamwork skills, creative and intellectual powers and passion for learning by providing an uncompromising standard of excellence in teaching; embodying the spirit of excellence to educate the citizen-leaders of society with distinction.

DEPARTMENT VISION AND MISSION

Vision

To emerge as a center of excellence with global reputation with adaption of rapid advancements in the field of computer specialization.

Mission

1. To provide a strong theoretical and practical background in area of computer science with an emphasize on software development.
2. To inculcate Professional behavior, strong ethical values, leadership qualities, research capabilities and lifelong learning.
3. To educate students to become effective problem solvers, apply knowledge with social sensitivity for the betterment of the society and humanity as a whole.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

Programme educational objectives are broad statements that describe the career and professional accomplishments that the programme is preparing graduates to achieve within 3 to 5 years after graduation.

The **Programme Educational Objectives** of the B. Tech CSE programme are:

- **PEO1:** To apply the knowledge of mathematics, basic science and engineering solving the real world computing problems to succeed higher education and professional careers.
- **PEO2:** To develop the skills required to comprehend, analyze, design and create innovative computing products and solutions for real life problems.
- **PEO3:** To inculcate professional and ethical attitude, communication and teamwork skills, multi-disciplinary approach and an ability to relate computer engineering issues with social awareness.

Course Objectives:

This course will enable students to

- Design and implement various algorithms in JAVA
- Employ various design strategies for problem solving.
- Measure and compare the performance of different algorithms.

Course outcomes:

- Design algorithms using appropriate design techniques (brute-force, greedy, dynamic programming, etc.)
- Implement a variety of algorithms such assorting, graph related, combinatorial, etc., in a high level language.
- Analyze and compare the performance of algorithms using language features.
- Apply and implement learned algorithm design techniques and data structuresto solve realworld problems.

Note:

The design, development, and implementation of the specified algorithms for the following problems can be done using Java language under LINUX /Windows environment. Netbeans/Eclipse IDE tool can be used for development and demonstration

DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY

Sub Code :	**22SCS044**	**IA Marks :**	**25**

Laboratory Experiments:

1. 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 non graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divideand-conquer method works along with its time complexity analysis: worst case, average caseand best case

2. 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 non 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.

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

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

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

6. Implement in Java, the 0/1 Knapsack problem using Greedy method.

7. Implement in Java, the 0/1 Knapsack problem using Dynamic Programming.

8. Write Java programs to

(a) Compute the Transitive Closure of a given directed graph using Warshall's Algorithm.

(b) Implement All-Pairs Shortest Paths problem using Floyd's algorithm.

9. Implement Travelling Sales Person problem using Dynamic programming.

10. 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.

CONTENTS

1. 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 divideand- conquer method works along with its time complexity analysis: worst case, average case and best case.

```
import java.util.Random;
import java.util.Scanner;

public class quicksort {
static int max=2000;
        int partition (int[] a, int low,int high)
        {
                int p,i,j,temp;
                p=a[low];
                i=low+1;
                j=high;

                while(low<high)
                {
                        while(a[i]<=p && i<high)
                                i++;
                        while(a[j]>p)
                                j--;
                        if(i<j)
                        {
                                temp=a[i];
                                a[i]=a[j];
                                a[j]=temp;
                        }
                        else
                        {
                                temp=a[low];
                                a[low]=a[j];
                                a[j]=temp;
                                return j;
                        }
                }
        return j;
        }

        void sort(int[] a,int low,int high)
```

```
    {
            if(low<high)
            {
            int s=partition(a,low,high);
                  sort(a,low,s-1);
                  sort(a,s+1,high);
            }
    }

    public static void main(String[] args)
{
           int[] a;
           int i;
           System.out.println("Enter the array size");
           Scanner sc =new Scanner(System.in);
           int n=sc.nextInt();
           a= new int[max];
           Random generator=new Random();
           for( i=0;i<n;i++)
           a[i]=generator.nextInt(100);
           System.out.println("Array before sorting");
           for( i=0;i<n;i++)
                   System.out.println(a[i]+" ");
           long startTime=System.nanoTime();
           quicksort m=new quicksort();
           m.sort(a,0,n-1);
           long stopTime=System.nanoTime();
           long elapseTime=(stopTime-startTime);
           System.out.println("Time taken to sort array is:"+elapseTime+" nanoseconds");
           System.out.println("Sorted array is");
           for(i=0;i<n;i++)
                   System.out.println(a[i]);
   }
}
```

Output:

2. 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 divideand- conquer method works along with its time complexity analysis: worst case, average case and best case.

```
import java.util.Random;
import java.util.Scanner;

public class mergesort {
        static int max=10000;
         void merge( int[] array,int low, int mid,int high)
        {
                int i=low;
                int j=mid+1;
                int k=low;
                int[]resarray;
                resarray=new int[max];
                while(i<=mid && j<=high)
                {
                        if(array[i]<array[j])
                        {
                                resarray[k]=array[i];
                                i++;
                                k++;
                        }
                        else
                        {
                                resarray[k]=array[j];
                                j++;
                                k++;
                        }
                }
                while(i<=mid)
                        resarray[k++]=array[i++];

                while(j<=high)
                        resarray[k++]=array[j++];

                for(int m=low;m<=high;m++)
                        array[m]=resarray[m];
        }
```

```
    void sort( int[] array,int low,int high)
   {
          if(low<high)
         {
               int mid=(low+high)/2;
               sort(array,low,mid);
               sort(array,mid+1,high);
               merge(array,low,mid,high);
         }
  }

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

         System.out.println("Enter the array size");
         Scanner sc =new Scanner(System.in);
         int n=sc.nextInt();

         array= new int[max];
         Random generator=new Random();

         for( i=0;i<n;i++)
         array[i]=generator.nextInt(100);
         System.out.println("Array before sorting");
         for( i=0;i<n;i++)
               System.out.println(array[i]+" ");
         long startTime=System.nanoTime();
         mergesort m=new mergesort();
         m.sort(array,0,n-1);
         long stopTime=System.nanoTime();
         long elapseTime=(stopTime-startTime);
         System.out.println("Time taken to sort seconds");
         System.out.println("Sorted array is: "+elapseTime+" Nanoseconds");
         for(i=0;i<n;i++)
               System.out.println(array[i]);
  }
}
```

<u>Output:</u>

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

```
import java.util.*;

public class prims {
        public static void main(String[] args)
{
    int w[][]=new int[10][10];

    int n,i,j,s,k;
    int min;
    int sum=0;
    int u=0,v=0;
    int flag=0;
    int sol[]=new int[10];
    System.out.println("Enter the number of vertices");
    Scanner sc=new Scanner(System.in);
    n=sc.nextInt();
    System.out.println("Enter the weighted graph");
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            w[i][j]=sc.nextInt();
    for(i=1;i<=n;i++)
        sol[i]=0;
    System.out.println("Enter the source vertex");
    s=sc.nextInt();
    sol[s]=1;
    k=1;
    while (k<=n-1)
    {
        min=99;
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                if(sol[i]==1 && sol[j]==0)
                if(w[i][j]<min)
                {
                    min=w[i][j];
                    u=i;
                    v=j;
                }
        sol[v]=1;
        sum=sum+min;
```

```
            k++;
            System.out.println(u+"->"+v+"="+min);
        }

        for(i=1;i<=n;i++)
            if(sol[i]==0)
            flag=1;
        if(flag==1)
            System.out.println("No spanning tree");
        else
            System.out.println("The cost of minimum spanning tree is "+sum);
        }
}
```

Output:

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

```
import java.util.Scanner;
public class kruskal
 {
     int parent[]=new int[10];

     int find(int m)
     {
          int p=m;
          while(parent[p]!=0)
               p=parent[p];
          return p;
     }

     void union(int i,int j)
     {
          if(i<j)
               parent[i]=j;
          //else
              //parent[j]=i;
     }


     void krkl(int[][]a, int n)
     {
          int u=0,v=0,min,k=1,i,j,sum=0;
          while(k<=n-1)
          {
               min=99;
               for(i=1;i<=n;i++)
                    for(j=1;j<=n;j++)
                         if(a[i][j]<min)
                         {
                              min=a[i][j];
                              u=i;
                              v=j;
                         }

               i=find(u);
               j=find(v);
```

```
            if(i!=j)
            {
                union(i,j);
                System.out.println("("+u+","+v+")"+"="+a[u][v]);
                sum=sum+a[u][v];
                k++;
            }

        a[u][v]=a[v][u]=99;

        }
        System.out.println("The cost of minimum spanning tree = "+sum);
    }

    public static void main(String[] args) {
    int a[][]=new int[10][10];
    int i,j;

    System.out.println("Enter the number of vertices of the graph");
    Scanner sc=new Scanner(System.in);
  int n;
  n=sc.nextInt();
  System.out.println("Enter the wieghted matrix");
  for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
           a[i][j]=sc.nextInt();
  kruskal k=new kruskal();
  k.krkl(a,n);
 // sc.close();
  }

}
```

Output:

5. 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.Scanner;

public class dijkstra
 {
            public static void main(String[] args)
           {
                      int cost[][]=new int[10][10],dist[]=new int[10],visited[]=new int[20];
                      int i,j;
                      Scanner in = new Scanner(System.in);
                      System.out.println("**** DIJKSTRA'S  ALGORITHM ******");
                      System.out.println("Enter the number of nodes: ");
                      int n = in.nextInt();
                      System.out.println("Enter the cost matrix");
                      for(i=1;i<=n;i++)
                      {
                                          for(j=1;j<=n;j++)
                                          {
                                          cost[i][j] = in.nextInt();
                                  }
                       }
                      System.out.println("Enter the source vertex: ");
                      int sv = in.nextInt();
                       dij(cost,dist,sv,n,visited);
                      System.out.println("Total Distance from Source to each Destination is");
                       for(i=1;i<=n;i++)
                      {
                                          System.out.println("From "+sv+ "to "+i+ "= "+dist[i]);
                                }
                      System.out.println("\n********* *************** *********");
}

static void dij(int cost[][],int dist[],int v,int n,int s[])
{
                      int w,u,k;
                       int i;
                      for(i=1;i<=n;i++)
                      {
                                s[i]=0;
                                dist[i]=cost[v][i];
                      }
```

```
                s[v]=1;
                dist[v]=0;
                for(i=2;i<=n;i++)
                {
                        u=min(n,dist,s);
                        s[u]=1;
                        for(w=1;w<=n;w++)
                        {
                                if(dist[w] > dist[u]+cost[u][w])
                                {
                                         dist[w]=dist[u]+cost[u][w];
                                }
                        }
                }
}

static int min(int n,int dist[],int s[])
{
                int i,p = 0,min=99;
                for(i=1;i<=n;i++)
                {
                        if(min > dist[i] && s[i]==0)
                        {
                                 min=dist[i];
                                p=i;
                        }
                }
                return p;
}
}
```

Output:

6. Implement in Java, the 0/1 Knapsack problem using Greedy method.

```
import java.util.Scanner;

public class lab6b
{
public static void main(String[] args)
 {
float w[]=new float[10],p[]=new float[10];
float ratio[]=new float[10];
Scanner in = new Scanner(System.in); int i;
System.out.println("********* KNAPSACK  PROBLEM *******");
System.out.println("Enter the total number of items: ");
int n = in.nextInt();
System.out.println("Enter the weight of each item: ");
for(i=1;i<=n;i++)
	{
			w[i] = in.nextFloat();
	}
System.out.println("Enter the profit of each item: ");
for(i=1;i<=n;i++)
	{
			p[i] = in.nextFloat();
	}
System.out.println("Enter the knapsack capacity: ");
int m= in.nextInt();
for(i=1;i<=n;i++)
	{
			ratio[i]=p[i]/w[i];
	}

System.out.println("Information about knapsack problem are");
displayinfo(n,w,p,ratio);
System.out.println("Capacity = "+m);
sortArray(n,ratio,w,p);
System.out.println("\nDetails after sorting items based on Profit/Weight ratio in descending
order: ");
displayinfo(n,w,p,ratio);
knapsack(m,n,w,p);
System.out.println("*************************************");
}

static void sortArray(int n,float ratio[],float w[],float p[])
```

```
{
 int i,j;
for(i=1; i<=n; i++)
            {
                                for(j=1; j<=n-i; j++)
            {
if(ratio[j]<ratio[j+1])
                    {
float temp=ratio[j];
 ratio[j]=ratio[j+1];
 ratio[j+1]=temp;


temp=w[j];
w[j]=w[j+1];
w[j+1]=temp;

temp=p[j];
p[j]=p[j+1];
p[j+1]=temp;
            }
                                }
                        }
            }

static void displayinfo(int n,float w[],float p[],float ratio[])
{
System.out.println("ITEM\tWEIGHT\tPROFIT\tRATIO(PROFIT/WEIGHT)");
 for(int i=1; i<=n; i++)
             {
                                System.out.println(i+"\t"+w[i]+"\t"+p[i]+"\t"+ratio[i]);
             }
}

static void knapsack(int u,int n,float w[],float p[])
{
float x[]=new float[10],tp=0;
int i;
for(i=1; i<=n; i++)
            {
                                x[i]=0;
            }
for(i=1; i<=n; i++)
```

```
        {
        if(w[i]>u)
                {  break;   }
                        else
                {
                        x[i]=1;
                                tp=tp+p[i];
                        u=(int) (u-w[i]);
                }
                }
        if(i<=n)
        {
                        x[i]=(float) u/w[i];
        }
        tp=tp+(x[i]*p[i]);
        System.out.println("\nThe result is = ");
        for(i=1; i<=n; i++)
                {
                        System.out.print("\t"+x[i]);
                }
                System.out.println("\nMaximum profit is = "+tp);
        }
}
```

Output:

7. Implement in Java, the 0/1 Knapsack problem using Dynamic Programming method.

```
import java.util.Scanner;
public class lab6a
{
public static void main(String[] args)
 {
int v[][]=new int[10][10], w[]=new int[10], p[]=new int[10];
                    Scanner in = new Scanner(System.in);
int i, j;
System.out.println("************* KNAPSACK  PROBLEM ***********");
System.out.println("Enter the total number of items: ");
 int n = in.nextInt();
System.out.println("Enter the weight of each item: ");
for(i=1;i<=n;i++)
          {
                              w[i] = in.nextInt();
          }
System.out.println("Enter the profit of each item: ");
           for(i=1;i<=n;i++)
                      {
                              p[i] = in.nextInt();
                    }
System.out.println("Enter the knapsack capacity: ");
int m= in.nextInt();
displayinfo(m,n,w,p);
 knapsack(m,n,w,p,v);
System.out.println("The contents of the knapsack table are");
for(i=0; i<=n; i++)
          {
                              for(j=0; j<=m; j++)
                              {
 System.out.print(v[i][j]+" " );
                                  }
                              System.out.println();
          }

optimal(m,n,w,v); //call optimal function
}

static void  displayinfo(int m,int n,int w[],int p[])
{
System.out.println("Entered information about knapsack problem are");
```

```
                    System.out.println("ITEM\tWEIGHT\tPROFIT");
                    for(int i=1; i<=n; i++)
                    {
                                        System.out.println(i+"\t"+w[i]+"\t"+p[i]);
                    }
System.out.println("Capacity = "+m);
}

static void knapsack(int m,int n,int w[],int p[],int v[][])
{
for(int i=0; i<=n; i++)
                    {
                              for(int j=0; j<=m; j++)
                              {
                                        if(i==0 ||j==0)
                                        {
                                                  v[i][j]=0;
                                        }
                    else if(j < w[i])
                    {
                              v[i][j]=v[i-1][j];
                              }
                    else
                    {
                              v[i][j]=max(v[i-1][j], v[i-1][j-w[i]]+p[i]);
                              }
                     }
          }
}
private static int max(int i, int j)
{
          if(i>j)
                     {
                    return i;
                       }
                    else
                    {
                              return j;
                     }
}

static void optimal(int m,int n,int w[],int v[][])
{
 int i = n, j = m, item=0;
                          int x[]=new int[10];
                          while( i != 0 && j != 0)
                         {

            if(v[i][j] != v[i-1][j])
            {
                                x[i] = 1;
                                  j = j-w[i];
                          }
                       i = i-1;
}
System.out.println("Optimal solution is :"+ v[n][m]);
System.out.println("Selected items are: ");
for(i=1; i<= n;i++)
{
                       if(x[i] == 1)
{
                                     System.out.print(i+" ");

                                   }
}
}
}
```

Output:

8. Write Java programs to

(a) Compute the Transitive Closure of a given directed graph using Warshall's Algorithm.

(b) Implement All-Pairs Shortest Paths problem using Floyd's algorithm.

8 a) Compute the Transitive Closure of a given directed graph using Warshall's Algorithm.

```
import java.util.Scanner;
public class warshall {
    void wshall(int[][] a,int n)
    {
        int i,j,k;
        for(k=1;k<=n;k++)
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        if(a[i][j]!=1)
        if(a[i][k]==1 && a[k][j]==1)
                a[i][j]=1
    }

    public static void main(String[] args) {
        int a[][]=new int[10][10];
        int n,i,j;
        System.out.println("enter the number of vertices");
        Scanner sc=new Scanner(System.in);
        n=sc.nextInt();

        System.out.println("Enter the adjacency matrix");
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                a[i][j]=sc.nextInt();

        warshall w=new warshall();
        w.wshall(a, n);

        System.out.println("The shortest path matrix is");
        for(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++)
            {
                System.out.print(a[i][j]+" ");
            }
```

```
                System.out.println();
            }
                }
}
```

Output:

8 b) Implement All-Pairs Shortest Paths problem using Floyd's algorithm.

```
import java.util.Scanner;
public class floyd {
    void flyd(int[][] w,int n)
    {
        int i,j,k;
        for(k=1;k<=n;k++)
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            w[i][j]=Math.min(w[i][j], w[i][k]+w[k][j]);
    }

    public static void main(String[] args) {
        int a[][]=new int[10][10];
        int n,i,j;
        System.out.println("enter the number of vertices");
        Scanner sc=new Scanner(System.in);
        n=sc.nextInt();

        System.out.println("Enter the weighted matrix");
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                a[i][j]=sc.nextInt();

        floyd f=new floyd();
        f.flyd(a, n);

        System.out.println("The shortest path matrix is");
        for(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++)
            {
                System.out.print(a[i][j]+" ");
            }
            System.out.println();
        }
            }
}
```

Output:

9. Implement Travelling Sales Person problem using Dynamic programming.

```
import java.util.Scanner;
public class tsp
{
public static void main(String[] args)
	{
		 int c[][]=new int[10][10], tour[]=new int[10];
		Scanner in = new Scanner(System.in);
		int i, j,cost;
		System.out.println("**** TSP  DYNAMIC  PROGRAMMING *******");
		System.out.println("Enter the number of cities: ");
		int n = in.nextInt();
		if(n==1)
		{
			 System.out.println("Path is not possible");
			System.exit(0);
		 }
		System.out.println("Enter the cost matrix");
		for(i=1;i<=n;i++)
		 for(j=1;j<=n;j++)
			c[i][j] = in.nextInt();
		System.out.println("The entered cost matrix is");
		 for(i=1;i<=n;i++)
		 {
			for(j=1;j<=n;j++)
			 {
					System.out.print(c[i][j]+"\t");
			}
			System.out.println();
		}
		 for(i=1;i<=n;i++)
		tour[i]=i;
		cost = tspdp(c, tour, 1, n);
		System.out.println("The accurate path is");
		for(i=1;i<=n;i++)
		System.out.print(tour[i]+"->");
		System.out.println("1");
		System.out.println("The accurate mincost is "+cost);
		System.out.println("******* ************* ***************");
	}
static int tspdp(int c[][], int tour[], int start, int n)
{
```

```
                     int mintour[]=new int[10], temp[]=new int[10], mincost=999, ccost, i, j, k;
                    if(start == n-1)
                    {
                            return (c[tour[n-1]][tour[n]] + c[tour[n]][1]);
                    }
                    for(i=start+1; i<=n; i++)
                    {
for(j=1; j<=n; j++)
                            temp[j] = tour[j];
                            temp[start+1] = tour[i];
                            temp[i] = tour[start+1];
                            if((c[tour[start]][tour[i]]+(ccost=tspdp(c,temp,start+1,n)))<mincost)
                            {
                                    mincost = c[tour[start]][tour[i]] + ccost;
                                    for(k=1; k<=n; k++)
                                            mintour[k] = temp[k];
                            }
                     }
                    for(i=1; i<=n; i++)
                     tour[i] = mintour[i];
                     return mincost;
}
}
```

Output:

10. 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 int c=0;
public static void main(String[] args)
{
int w[]=new int[10];
int n, d, i, sum=0; int x[]=new int[10];
Scanner in=new Scanner(System.in);
System.out.println("********** SUBSET PROBLEM ************");
System.out.println("Enter the number of elements: ");
n=in.nextInt();
System.out.println("Enter the elements in increasing order");
for(i=0;i<n;i++)
w[i]=in.nextInt();
System.out.println("Enter the value of d: ");
d=in.nextInt();
for(i=0;i<n;i++)
sum=sum+w[i];
System.out.println("SUM ="+sum);
if(sum < d || w[0] > d)
{
System.out.println("Subset is not possible ! ");
System.out.println("********** *********** *************");
System.exit(0);
}
subset(0,0,sum,x,w,d);
if(c==0)
System.out.println("Subset is not possible ! "); System.out.println("\n**********
********* *************");
}

static void subset(int cs, int k, int r,int x[],int w[],int d){
x[k] = 1;
if(cs+w[k] == d)
{
c++;
System.out.print("\nSolution "+c+" is {");
for(int i=0;i<=k;i++)
if(x[i] == 1)
{
System.out.print(w[i]+" ");
}
System.out.print("}");
}
else if((cs + w[k] + w[k+1]) <= d)
subset(cs + w[k], k+1, r-w[k],x,w,d);
if((cs + r - w[k]) >= d && (cs + w[k+1]) <= d)
{
                    x[k] = 0;
subset(cs, k+1, r-w[k],x,w,d);
}

}

}
```

Output:

VIVA QUESTIONS AND ANSWERS

***************************SET I***************************

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 are important problem types? (or) Enumerate some important types of problems.
1. Sorting 2. Searching
3. Numerical problems 4. Geometric problems
5. Combinatorial Problems 6. Graph Problems
7. String processing Problems

3. Name some basic Efficiency classes
1. Constant 2. Logarithmic 3. Linear 4. nlogn
5. Quadratic 6. Cubic 7. Exponential 8. Factorial

4. What are algorithm design techniques?
Algorithm design techniques (or strategies or paradigms) are general approaches to solving problems algorithmatically, applicable to a variety of problems from different areas of computing. General design techniques are:
(i) Brute force (ii) divide and conquer
(iii) decrease and conquer (iv) transform and concquer
(v) greedy technique (vi) dynamic programming
(vii) backtracking (viii) branch and bound

5. How is an algorithm's time efficiency measured?
Time efficiency indicates how fast the algorithm runs. An algorithm's time efficiency is measured as a function of its input size by counting the number of times its basic operation (running time) is executed. Basic operation is the most time consuming operation in the algorithm's innermost loop.

6. How is the efficiency of the algorithm defined?

The efficiency of an algorithm is defined with the components.
(i) Time efficiency -indicates how fast the algorithm runs
(ii) Space efficiency -indicates how much extra memory the algorithm needs

7. What are the characteristics of an algorithm?
Every algorithm should have the following five characteristics
(i) Input
(ii) Output
(iii) Definiteness
(iv) Effectiveness
(v) Termination
Therefore, an algorithm can be defined as a sequence of definite and effective instructions, which terminates with the production of correct output from the given input.

8. Write general plan for analyzing non-recursive algorithms.
i. Decide on parameter indicating an
input's size. **i** . Identify the algorithm's
basic operation
iii. Checking the no.of times basic operation executed depends on size of input.
iv. Set up sum expressing the no.of times the basic operation is executed. depends on some additional property,then best,worst,avg.cases need to be investigated (establishing order of growth)

9. Define the terms: pseudocode, flow chart
A pseudocode is a mixture of a natural language and programming language like constructs. A pseudocode is usually more precise than natural language. A flowchart is a method of expressing an algorithm by a collection of connected geometric shapes containing descriptions of the algorithm's steps.

10. Write general plan for analyzing recursive algorithms.
i. Decide on parameter indicating an
input's size. **i** . Identify the algorithm's
basic operation
iii. Checking the no.of times basic operation executed depends on size of input.if it
depends on some additional property,then best,worst,avg.cases need to be investigated
of times the basic operation is executed
iv. Set up the recurrence relation,with an appropriate initial condition,for the number
v. Solve recurrence (establishing order of growth)

11. Define the divide an conquer method.
Given a function to compute on 'n' inputs the divide-and-comquer strategy suggests splitting the inputs in to'k' distinct susbsets, 1<k <n, yielding 'k' subproblems. The subproblems must be solved recursively, and then a method must be found to combine subsolutions into a solution of the whole.

12. What is Merge sort?
Merge sort is divide and conquer strategy that works by dividing an input array
in to two halves,sorting them recursively and then merging the two sorted halves
to get the original array sorted

13. What is general divide and conquer recurrence?
Time efficiency T(n)of many divide and conquer algorithms satisfies the equation T(n)=a.T(n/b)+f(n).This is the general recurrence relation.

14. Describe the recurrence relation for merge sort?
If the time for the merging operation is proportional to n, then the computing time
of merge sort is described by the recurrence relation
T(n) = a n = 1, a a
constant 2T (n/2) + n
n >1, c a constant

15. The relation between order of growth of functions
$O(1) < O(\log n) < O(n) < O(n * \log n) < O(n^2) < O(n^3) < O(2^n)$

16. Asymptotic notations

Big Oh(Worst Case), Big Theta (Average Case), Big Omega(Best Case)

GREEDY METHOD

1. 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 candevice an algorithm that works in stages considering one input at a time.

2. 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

3. What are the constraints of knapsack problem?

To maximize ∑pixi

The constraint is : ∑wixi ≥ m and 0 ≤ xi ≤ 1 1≤ i ≤ n

where m is the bag capacity, n is the number of objects and for each object i wi and pi are the weight and profit of object respectively.

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

a) Prim's Algorithm

b) Kruskal's Algorithm

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

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

6. 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)

7. State Kruskal Algorithm.

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

8. 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)

DYNAMIC PROGRAMMING

1. Define multistage graph

A multistage graph G =(V,E) is a directed graph in which the vertices are partitioned into 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)

2. Define All pair shortest path problem

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

3. Define floyd's algorithm

To find all pair shortest path

4. State the time efficiency of floyd's algorithm

$O(n^3)$

It is cubic

*******************************SET II*******************************

1) What is the time complexity of linear search?

$\Theta(n)$

2) What is the time complexity of binary search?

$\Theta(\log_2 n)$

3) What is the major requirement for binary search?

The given list should be sorted.

4) What is binary search?

It is an efficient method of finding out a required item from a given list, provided the list is in order.

The process is:

1. First the middle item of the sorted list is found.
2. Compare the item with this element.
3. If they are equal search is complete.
4. If the middle element is greater than the item being searched, this process is repeated in the upper half of the list.

5. If the middle element is lesser than the item being searched, this process is repeated in the lower half of the list.

5) What is parental dominance?

The key at each node is greater than or equal to the keys at its children.

6) What is heap?

A **heap** can be defined as a binary tree with keys assigned to its nodes (one key per node) provided the following two conditions are met:

1 The tree's shape requirement – The binary tree is essentially complete (or simply

complete), that is, all its levels are full except possibly the last level, where only some rightmost leaves may be missing.
2. The parental dominance requirement – The key at each node is greater than or equal to the keys at its children

7) What is height of Binary tree?
It is the longest path from the root to any leaf.

8) What is the time complexity of heap sort?
Θ(n logn)

9) What is Merge Sort?
Merge sort is an O (n log n) comparison-based sorting algorithm. Where the given array is divided into two equal parts. The left part of the array as well as the right part of the array is sorted recursively. Later, both the left sorted part and right sorted part are merged into a single sorted array.

10) Who invented Merge Sort?
John Von Neumann in 1945.

11) On which paradigm is it based?
Merge-sort is based on the divide-and-conquer paradigm.

12) What is the time complexity of merge sort?
n log n.

13) What is the space requirement of merge sort?
Space requirement: Θ(n) (not in-place).

14) When do you say an algorithm is stable and in place?
Stable – if it retains the relative ordering. In – place if it does not use extra memory.

15) Who invented Dijkstra's Algorithm?
Edsger Dijkstra invented this algorithm.

16) What is the other name for Dijkstra's Algorithm?
Single Source Shortest Path Algorithm.

17) Which technique the Dijkstra's algorithm is based on?
Greedy Technique.

18) What is the purpose of Dijkstra's Algorithm?
To find the shortest path from source vertex to all other remaining vertices

19) Name the different algorithms based on Greedy technique.
Prim's Algorithm, Kruskal's algorithm

20) What is the constraint on Dijkstra's Algorithm?
This algorithm is applicable to graphs with non-negative weights only.

21) What is a Weighted Graph?
A weighted graph is a graph with numbers assigned to its edges. These numbers are called weights or costs.

22) What is a Connected Graph?
A graph is said to be connected if for every pair of its vertices u and v there is a path from u to v.

23) What is the time complexity of Dijkstra's algorithm?
For adjacency matrix, It is O(IVI*IVI)
For adjacency list with edges stored as min heap, it is O(IEIlogIVI)

24) Differentiate b/w Traveling Salesman Problem(TSP) and Dijkstra's Algorithm.
In TSP, given n cities with known distances b/w each pair , find the shortest tour that passes through all the cities exactly once before returning to the starting city.
In Dijkstra's Algorithm, find the shortest path from source vertex to all other remaining vertices

25) Differentiate b/w Prim's Algorithm and Dijkstra's Algorithm?
Prim's algorithm is to find minimum cost spanning tree.
Dijkstra's algorithm is to find the shortest path from source vertex to all other remaining vertices.

26) What are the applications of Dijkstra's Algorithm?
It finds its application even in our day to day life while travelling. They are helpful in routing applications.

27) Who was the inventor of the Quicksort?
C.A.R.HOARE, a British Scientist.

28) Which design strategy does Quicksort uses?
Divide and Conquer

29) What is Divide and Conquer Technique?
(I) Divide the instance of a problem into two or more smaller instances
(II) Solve the smaller instances recursively
(III) Obtain solution to original instances by combining these solutions

30) What is the another name for Quicksort?
Partition Exchange Sort

31)Is Quicksort Stable as well as In place?

Not Stable but In place. When do you say that a Quick sort having best case complexity? When the pivot exactly divides the array into equal half

32) When do you say that Quick sort having worst case complexity?
When any one of the subarray is empty

33) How many more comparisons are needed for average case compared to best case?
38% more

34) when do you say that the pivot element is in its final position?
When all the elements towards the left of pivot are <= pivot and all the elements towards the right of pivot are >= pivot.

35) What technique does kruskal's algorithm follow.
Greedy technique

36) What is the time complexity of kruskal algorithm.
With an efficient sorting algorithm, the time efficiency of an kruskal's algorithm will be in O(|E|log|E|).

37) Who is the inventor of kruskal's algorithm.
Joseph Kruskal.

38) Which technique is used to solve BFS & DFS problems?
Decrease and Conquer

39) What is DFS traversal?
Consider an arbitrary vertex v and mark it as visited on each iteration, proceed to an unvisited vertex w adjacent to v. we can explore this new vertex depending upon its adjacent information. This process continues until dead end is encountered.(no adjacent unvisited vertex is available). At the dead end the algorithm backs up by one edge and continues the process of visiting unvisited vertices. This process continues until we reach the starting vertex.

40) What are the various applications of BFS & DFS tree traversal technique? Application of BFS
Checking connectivity and checking acyclicity of a graph. Check whether there is only one root in the BFS forest or not.
If there is only one root in the BFS forest, then it is connected graph otherwise disconnected graph.
For checking cycle presence, we can take advantage of the graph's representation in the form of a BFS forest, if the latter vertex does not have cross edge, then the graph is acyclic.
Application of DFS
To check whether given graph is connected or not. To check whether the graph is cyclic or not.
To find the spanning tree.
Topological sorting.
41) Which data structures are used in BFS & DFS tree traversal technique?

We use Stack data structure to traverse the graph in DFS
traversal. We use queue data structure to traverse the graph in
BFS traversal.

42) What is Dynamic Programming?
It is a technique for solving problems with overlapping sub problems.

43) What does Dynamic Programming have in common with Divide and- Conquer?
Both solve the problems by dividing the problem into sub problems. Using the solutions of sub problems, the solutions for larger instance of the problem can be obtained.

44) What is a principle difference between the two techniques?
Only one instance of the sub problem is computed & stored. If the same instance of sub problem is encountered, the solution is retrieved from the table and never recomputed. Very precisely re - computations are not preformed.

45) What is a spanning tree ?
A spanning tree of a weighted connected graph is a connected acyclic(no cycles) subgraph (i.e. a tree)that contains all the vertices of a graph and number of edges is one less than number of vertices.

46) What is a minimum spanning tree ?
Minimum spanning tree of a connected graph is its spanning tree of smallestweight.

47) Prim's algorithm is based on which technique.
Greedy technique.

48) Name some of the application greedy technique
They are helpful in routing applications: In the case of network where large number of computers are connected, to pass the data from one node to another node we can find the shortest distance easily by this algorithm. Communication Networks: Suppose there are 5 different cities and we want to establish computer connectivity among them, then we take into the account of cost factor for communication links. Building a road network that joins n cities with minimum cost.

49) Does Prim's Algorithm always yield a minimum spanning tree ?
Yes

50) What is the purose of Floyd's algoreithm?
- To find the all pair shortest path.

51) What is the time efficiency of floyd's algorithm?
- It is same as warshal 's algorithm that is $\theta(n^3)$.

52) What is shortest path?
- It is the path which is having shortest length among all possible paths.

53) warshall's algorithm is optimization problem or non-optimization problem ?

Non-optimization problem

54) For what purpose we use Warshall's algorithm?
Warshal 's algorithm for computing the transitive closure of a directed graph

55) Warshall's algorithm depends on which property?
Transitive property

56) what is the time complexity of Warshall's algorithm?
Time complexity of warshal 's algorithm is θ(n^3).
57) what is state space tree?
Constructing a tree of choices being made and processing is known as state-spacetree. Root represents an initial state before the search for solution begins.

60) what are promising and non-promising state-space-tree?
Node which may leads to solution is called promising.
Node which doesn't leads to solution is cal ed non-promising.

61) What are the requirements that are needed for performing Backtracking?
Complex set of constraints. They are:
i. Explicit constraints.
ii. Implicit constraints.