Estd : 2001

RNS INSTITUTE OF TECHNOLOGY

(**AICTE** Approved, **VTU** Affiliated and **NAAC 'A+ Grade'** Accredited)

(**UG programs** – CSE, ECE, ISE, EIE and EEE have been Accredited by **NBA** up to **30/6/2025**)

Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098

DEPARTMENT OF COMPUTER SCIENCE &ENGINEERING

DESIGN & ANALYSIS OF ALGORITHMS LABORATORY MANUAL

For Fourth Semester B.E-2021 Batch [VTU/CBCS, 2021 syllabus]

Subject Code – 21CS42

NAME : ....

USN : ....

SECTION : .... **BATCH :** ....

VISION AND MISSION OF INSTITUTION

Vision

Building RNSIT into a World Class Institution

Mission

To impart high quality education in Engineering, Technology and Management with a Difference, Enabling Students to Excel in their Career by

1. Attracting quality Students and preparing them with a strong foundation in fundamentals so as to achieve distinctions in various walks of life leading to outstanding contributions
2. Imparting value based, need based, choice based and skill based professional education to the aspiring youth and carving them into disciplined, World class Professionals with social responsibility
3. Promoting excellence in Teaching, Research and Consultancy that galvanizes academic consciousness among Faculty and Students
4. Exposing Students to emerging frontiers of knowledge in various domains and make them suitable for Industry, Entrepreneurship, Higher studies, and Research & Development
5. Providing freedom of action and choice for all the Stake holders with better visibility

VISION AND MISSION OF CSE DEPARTMENT

Vision

Preparing better computer professionals for a real world

Mission

The Department of Computer Science and Engineering will make every effort to promote an intellectual and an ethical environment in which the strengths and skills of Computer Professionals will flourish by

1. Imparting Solid foundations and Applied aspects in both Computer Science Theory and Programming practices
2. Providing Training and encouraging R&D and Consultancy Services in frontier areas of Computer Science with a Global outlook
3. Fostering the highest ideals of Ethics, Values and creating Awareness on the role of Computing in Global Environment
4. Educating and preparing the graduates, highly Sought-after, Productive, and Well-respected for their work culture
5. Supporting and inducing Lifelong Learning practice

PREFACE

We have developed this comprehensive laboratory manual on **Design & Analysis of Algorithms Laboratory (21CS42)** with the primary objectives: To make the students comfortable withJava Programming Language using the eclipse software.

This material is comprised of solutions for 12 lab experiments, additional programs to show the different ways to solve the same problem and encourage students to come up with their own creative ways to solve them and to gain better knowledge on real world problems and includes Viva questions.

Our profound and sincere efforts will be fruitful only when students acquire the extensive knowledge by reading this manual and apply the concepts learnt apart from the requirements specified in Design & Analysis of Algorithms Laboratory as prescribed by VTU, Belagavi.

Department of CSE

ACKNOWLEDGMENT

A material of this scope would not have been possible without the contribution of many people. We express our sincere gratitude to Dr. Satish R Shetty, Chairman, RNS Group of Companies for his magnanimous support in all our endeavors.

We are grateful to Dr. Ramesh Babu H S, Principal, RNSIT and Dr. Kiran P, HOD, CSE for extending their constant encouragement and support.

Our heartfelt thanks to Dr. Bhavani Shankar K, Mrs. Rashmi M and Mrs. Meenakshi S J for their unparalleled contribution throughout the preparation of this comprehensive manual. We also acknowledge our colleagues for their timely suggestions and unconditional support.

Department of CSE

Design & Analysis of Algorithms Laboratory- 21CS42
INTERNAL EVALUATION SHEET

EVALUATION (MAX MARKS 20)			
TEST A	REGULAR EVALUATION B	RECORD C	TOTAL MARKS A+B+C
5	5	10	20

R1: REGULAR LAB EVALUATION WRITE UP RUBRIC (MAX MARKS 10)				
Sl. No.	Parameters	Good	Average	Needs improvement
a.	**Understanding of problem (3 marks)**	Clear understanding of problem statement while designing and implementing the program (3)	Problem statement is understood clearly but few mistakes while designing and implementing program (2)	Problem statement is not clearly understood while designing the program (1)
b.	**Writing program (4 marks)**	Program handles all possible conditions (4)	Average condition is defined and verified. (3)	Program does not handle possible conditions (1)
c.	**Result and documentation (3 marks)**	Meticulous documentation and all conditions are taken care (3)	Acceptable documentation shown (2)	Documentation does not take care all conditions (1)

R2: REGULAR LAB EVALUATION VIVA RUBRIC (MAX MARKS 10)					
Sl. No.	Parameter	Excellent	Good	Average	Needs Improvement
a.	**Conceptual understanding (10 marks)**	Answers 80% of the viva questions asked (10)	Answers 60% of the viva questions asked (7)	Answers 30% of the viva questions asked (4)	Unable to relate the concepts (1)

R3: REGULAR LAB PROGRAM EXECUTION RUBRIC (MAX MARKS 10)				
Sl. No.	Parameters	Excellent	Good	Needs Improvement
a.	**Design, implementation,and demonstration (5 marks)**	Program follows syntax and semantics of Java programming language. Demonstrates the complete knowledge of the program written (5)	Program has few logical errors, moderately demonstrates all possible concepts implemented in programs (3)	Syntax and semanticsof Java programming is not clear (1)
b.	**Result and documentation (5 marks)**	All expected results are demonstrated successful, allerrors are debugged with own practical knowledge and clear documentation according to the guidelines (5)	Moderately debugs the program and Partial documentation. (3)	Expected results are not demonstrated properly, unable to debug the errors and no proper documentation (1)

R4: RECORD EVALUATION RUBRIC (MAX MARKS 10)					
Sl. No.	Parameter	Excellent	Good	Average	Needs Improvement
a.	**Documentation (10 marks)**	Meticulous record writing including program, comments and expected output as per the guidelines mentioned (10)	Write up contains program and expected output, but comments are not included (8)	Write up contains only program (5)	Program written with few mistakes (3)

A. TEST /LAB INTERNALS MARKS (MAX MARKS 5)

TEST #	Write up 6	Execution 28	Viva 6	Sign	Total 40	Avg. 40	Final 5
TEST-1							
TEST-2						$\overline{40}$	$\overline{5}$

B. REGULAR LAB EVALUATION (MAX MARKS 5)

Program #	Date of Execution	Lab programs	Write up (10)	Exen. (10)	Viva (10)	Total 30	Teacher Signature
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
Avg. Total Marks		$\overline{30}$		$\overline{30}$		$\overline{5}$	

FINAL MARKS OBTAINED			
A: TEST (5)		TOTAL (A+B+C) $\overline{20}$	Signature of lab in charge:
B: REGULAR EVALUATION (5)			
C: RECORD (10)			

TABLE OF CONTENTS

SYALLABUS

SEMESTER – IV

DESIGN & ANALYSIS OF ALGORITHMS LABORATORY

(Effective from the academic year 2021-2022)

Course Code – 21CS42	CIE Marks - 50
Number of Contact Hours/Week -3:0:2:0	SEE Marks - 50
Total Number of Lab Contact Hours - 20	Exam Hours - 03

Course Learning Objectives:

This course will enable students to:

1. Explain the methods of analyzing the algorithms and to analyze performance of algorithms. State algorithm's efficiencies using asymptotic notations.
3. Solve problems using algorithm design methods such as the brute force method, greedy method, divide and conquer, decrease, and conquer, transform, and conquer, dynamic programming, backtracking and branch and bound.
4. Choose the appropriate data structure and algorithm design method for a specified application.
5. Introduce P and NP classes.

Programs List:

1. Sort a given set of n integer elements using **Selection 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.

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

3. 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.

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

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

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

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

9. Write Java programs to implement All-Pairs Shortest Paths problem using **Floyd's algorithm**.

10. Write Java programs to implement **Travelling Salesperson problem** using Dynamic programming.

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.

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

Laboratory Outcomes: The student should be able to:

- Apply various algorithmic techniques.
- Analyzing the algorithms for time efficiency.

Practical Sessions need to be assessed by appropriate rubrics and viva-voce method. This will contribute to 20 marks.

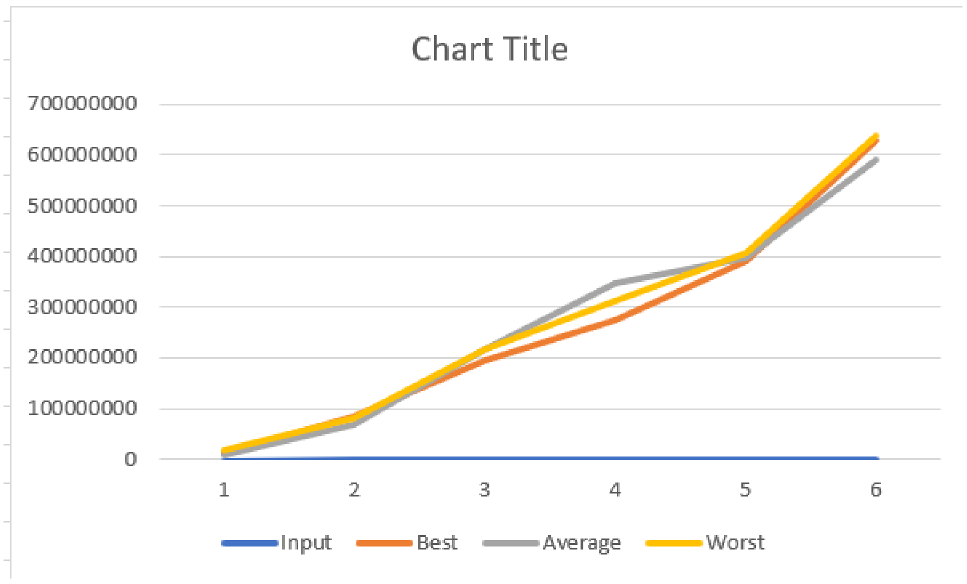
Note: Minimum of 80% of the laboratory components must be covered.

• Rubrics for each Experiment taken average for all Lab programs – 15 Marks.
• Viva-Voce– 5 Marks

1. ***Sort a given set of n integer elements using Selection Sort method and compute its timecomplexity. 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 fileor can be generated using the random number generator. Demonstrate using Java how thedivide-and-conquer method works along with its time complexity analysis: worst case, averagecase, and best case.***

```
import java.util.*;

public class SelectionSort {

    public void sort(int array[])

    {

        int min;

        for(int i=0; i<array.length-1; i++)

        {

                min=i;

                for(int j=i+1; j<array.length-1; j++)

                {

                        if(array[min]<array[j])

                                min=j;

                }

                if(min!=i)

                {

                        int temp=array[i];

                        array[i]=array[min];

                        array[min]=temp;

                }

        }

    } }

public class Demo {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Scanner sc = new Scanner(System.in);
        Random r = new Random();
        int n,op;
        System.out.println("Enter the size of array");
        n = sc.nextInt();
        int array[]= new int[n];
        for(int i=0; i<n; i++)
                array[i]=r.nextInt(0,1000000);
        SelectionSort ob= new SelectionSort();
        long start,stop;

        while(true)
        {
                System.out.println("Enter an option\n 1:Best case\n2:Average case\n3:Worst
case\n");
                op=sc.nextInt();
                switch(op)
                {
                case 1:
                        start=System.nanoTime();
                        ob.sort(array);
                        stop = System.nanoTime();
                        System.out.println("Best case: "+(stop-start));
                        break;
                case 2:
                        start=System.nanoTime();
                        ob.sort(array);
                        stop = System.nanoTime();
                        System.out.println("Average case: "+(stop-start));
                        break;
                case 3:
                        start=System.nanoTime();
                        ob.sort(array);
                        stop = System.nanoTime();
                        System.out.println("Worst case: "+(stop-start));
                        break;
                default:
                                System.exit(0);
                } } } }
```

Output:



2. ***Sort a given set of n integer elements using Quick Sort method and compute its timecomplexity. 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 fileor can be generated using the random number generator. Demonstrate using Java how thedivide-and-conquer method works along with its time complexity analysis: worst case, averagecase, and best case.***

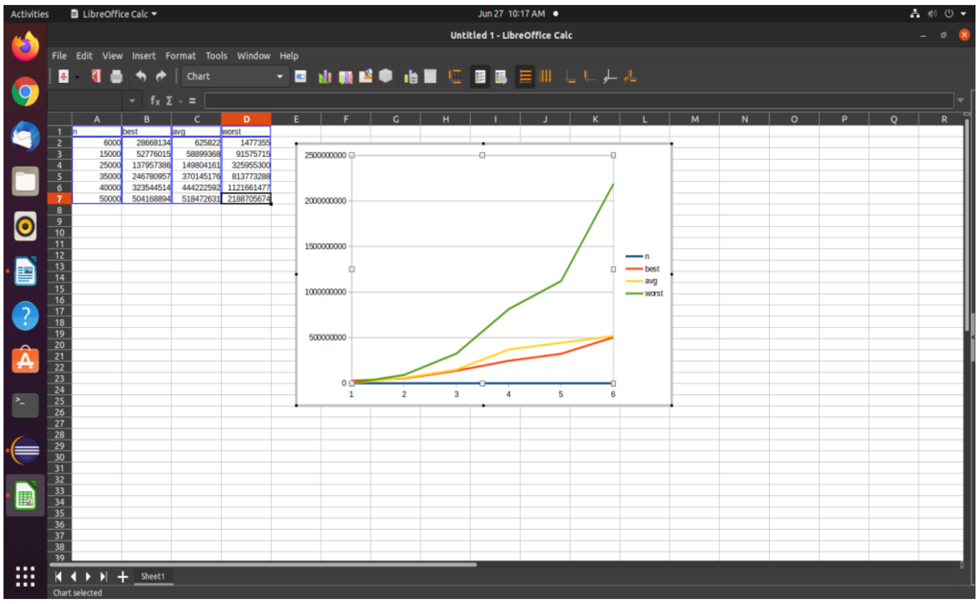
```
import java.util.*;
public class QuickSort {
    public int partition(int arr[],int low,int high)
    {
            int key=arr[low];
            int i=low;
            int j=high;
            while(i<j)
            {
                    while(i<high && key>=arr[i]){
                            i=i+1;
                    }
                    while(j>low && key<=arr[j]){
                            j=j-1;
                    }
                    if(i<j)
                    {
                            int temp=arr[i];
                            arr[i]=arr[j];
                            arr[j]=temp;
                    }
            }
            int temp=arr[low];
            arr[low]=arr[j];
            arr[j]=temp;
```

```
                return j;
        }
        public void sort(int arr[],int low,int high)
        {
                if(low<high)
                {
                        int k=partition(arr,low,high);
                        sort(arr,low,k-1);
                        sort(arr,k+1,high);
                }
                else
                        return;
        }
}

public class Demo {

        public static void main(String[] args) {
                // TODO Auto-generated method stub
                Random r= new Random();
                Scanner sc = new Scanner(System.in);
                QuickSort ob = new QuickSort();
                        System.out.println("Enter the size");
                        int n=sc.nextInt();
                        int arr[]=new int[n+1];
                        for(int i=0;i<n; i++)
                                arr[i]=r.nextInt(0,5000);
                        arr[n]=Integer.MAX_VALUE;
                        int op;
                        long start,stop;
```

```
                    while(true){
                        System.out.println("Enter the option");
                        op=sc.nextInt();
                    switch(op)
                    {
                    case 1:
                        start=System.nanoTimEnter the number of vertices
                        ob.sort(arr,0,n);
                        stop=System.nanoTime();
                        System.out.println("*****\nBest                    case:"+(stop-
start)+"\n******");
                        break;
                    case 2:
//                      for(int i=0; i<n; i++)
//                          System.out.print(arr[i]+" ");
                        start=System.nanoTime();
                        ob.sort(arr,0,n);
                        stop=System.nanoTime();
                        System.out.println("*****\nAverage                 case:"+(stop-
start)+"\n******");
//                      for(int i=0; i<n; i++)
//                          System.out.print(arr[i]+" ");
//                      System.out.println(" ");
                        break;
                    case 3:
                        start=System.nanoTime();
                        ob.sort(arr,0,n);
                        stop=System.nanoTime();
                        System.out.println("*****\nWorst                   case:"+(stop-
start)+"\n******");
                        break;
```

```
                                        default:

                                                System.exit(0);

                                        }

                }


        }

        }
```

Output:



	A	B	C	D
1	n	best	avg	worst
2	6000	28668134	625822	1477355
3	15000	52776015	58899368	91575715
4	25000	137957386	149804161	325955300
5	35000	246780957	370145176	813773288
6	40000	323544514	444222592	1121661477
7	50000	504168894	518472631	2188705674

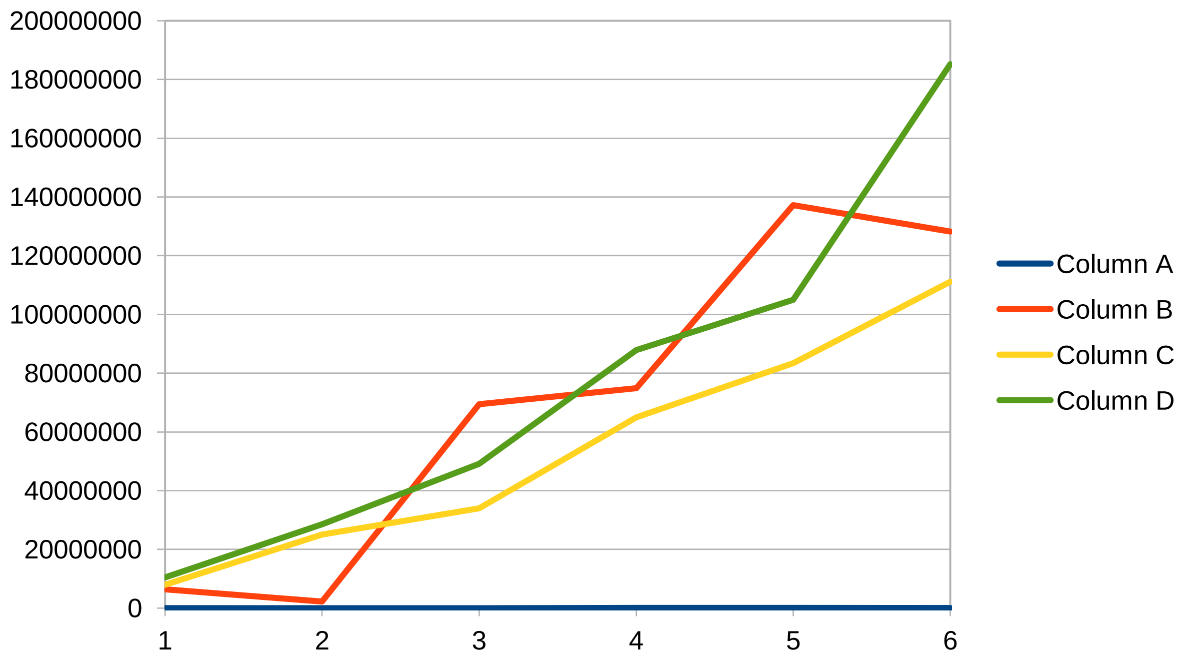
3. ***Sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. Plot a graph of the time taken versus n on graph sheet. The elements can be read from a file or can be generated using 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.Scanner;
import java.util.Random;
class MergeSort
{
        public static void main(String[] args)
        {
                Scanner scan=new Scanner(System.in);
                Random ran=new Random();
                long start,stop;
                System.out.println("Enter no of elements");
                int n=scan.nextInt();
                int[] a=new int[n];
                System.out.println(" Enter the choice 1: Best Case  2: Average Case  3:  Worst
                Case");
                int ch=scan.nextInt();
                switch(ch)
                {
                        case 1: System.out.println(" Best Case");
                                        for(int i=0;i<n;i++)
                                                a[i]=i;
                                break;
                        case 2: System.out.println(" Average Case");
                                        for(int i=0;i<n;i++)
                                                a[i]=ran.nextInt(n);
                                break;
                        case 3: System.out.println(" Worst Case");
                                        for(int i=0;i<n;i++)
                                                a[i]=n-i;
                                break;

                }// end switch
                //recording the start time
                start=System.nanoTime();
                //function call
                Mergesort(a,0,n-1);
                // recording the end time
                stop=System.nanoTime();
                display(a);
                System.out.println("\nTime taken to sort " +a.length+ " elements =" +(stop-
                start));

        }// end main
```

```
private static void display(int[] a)
{
        // TODO Auto-generated method stub
        System.out.println("the sorted array is");
        for(int i=0;i<a.length;i++)
                System.out.println(a[i]);

}//end dispaly

//function o divide the array
public static void Mergesort(int[] a, int low, int high)
{
        int mid;
        if(low<high)// array contains more than one element
        {
                mid=(low+high)/2;// dividing the array in to two sub arrays
                Mergesort(a, low, mid);// sorting sub arrays
                Mergesort(a, mid+1, high);
                Merge(a,low,mid,high);// combining or merging the sorted arrays
        }
}// end Mergesort

//function to merge two sorted arrays
public static void Merge(int[] arr,int low,int mid,int high)
{
        int k,h=low,i=low,j=mid+1;
        int[] b=new int[arr.length];
        while(h<=mid && j<=high)
        {
                if(arr[h]<=arr[j])
                {
                        b[i]=arr[h];
                        h++;
                }
                else
                {
                        b[i]=arr[j];
                        j++;
                }
                i++;
        }// end while
        if(h>mid) // for remaining elements in upper half
        {
                for(k=j;k<=high;k++)
                {
                        b[i]=arr[k];
                        i++;
                }
        }
```

```
                    else // for remaining elements in lower half
                    {
                              for(k=h;k<=mid;k++)
                              {
                                        b[i]=arr[k];
                                        i++;
                              }
                    }
                    //copy the contents from auxiliary array i.e. from b to arr
                    for(k=low;k<=high;k++)
                              arr[k]=b[k];
          }// end merge
}// end MergeSort class
```

Output:



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

```
package kk;
import java.util.Scanner;

class knapsack {

    private double[] weight, profit, ratio, solnVector;
    private double capacity;

    knapsack() {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter number of Items");
int n = scanner.nextInt();
        weight = new double[n];
        profit = new double[n];
        ratio = new double[n];
        solnVector = new double[n];

        System.out.println("Enter Weights of Items");
        for (int i = 0; i < n; i++) {
            weight[i] = scanner.nextDouble();
        }

        System.out.println("Enter Profits of Items");
        for (int i = 0; i < n; i++) {
            profit[i] = scanner.nextDouble();
        }

        System.out.println("Enter Capacity of Knapsack");
        capacity = scanner.nextDouble();
    }

    int getNext() {
        int index = -1;
        double highest = 0;

        for (int i = 0; i < ratio.length; i++) {
            if (ratio[i] > highest) {
                highest = ratio[i];
                index = i;
            }
        }
        return index;
    }

    void fill() {
        double currentWeight = 0;
        double currentProfit = 0;

        for (int i = 0; i < ratio.length; i++) {
            ratio[i] = profit[i] / weight[i];
        }

        System.out.print("Items Considered: ");
        while (currentWeight < capacity) {
            int item = getNext();

            if (item == -1) {
                break;
            }

            System.out.print((item + 1) + " ");

            if (currentWeight + weight[item] <= capacity) {
                currentWeight += weight[item];
                currentProfit += profit[item];
                solnVector[item] = 1;
                ratio[item] = 0;
            } else {
                currentProfit += ratio[item] * (capacity - currentWeight);
                solnVector[item] = (capacity - currentWeight) / weight[item];
                break;
            }
        }

        System.out.println();
        System.out.println("Maximum Profit is: " + currentProfit);
        System.out.print("Solution Vector: ");
        for (int i = 0; i < solnVector.length; i++) {
            System.out.print(solnVector[i] + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        knapsack knapsack = new knapsack();
        knapsack.fill();
    }
}
```

Output:

```
Enter number of Items
3

Enter Weights of Items

18

15

10

Enter Profits of Items

25

24

15

Enter Capacity of Knapsack

20

Items Considered: 2 3

Maximum Profit is: 31.5

Solution Vector: 0.0 1.0 0.5
```

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.

```
package kk;
import java.util.Scanner;

class Main {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter Number of Vertices");
        int n = scanner.nextInt();

        int adj[][] = new int[n][n];

        System.out.println("Enter Adjacency Matrix");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                adj[i][j] = scanner.nextInt();
            }
        }

        System.out.println("Enter Source vertex");
        int src = scanner.nextInt();

        int[] dist = dijkstra(adj, src);

        for (int i = 0; i < n; i++) {
            if ((src - 1) == i) {
                continue;
            }
            System.out.println("Shortest Distance from " + src + " to " + (i + 1) + " is " + dist[i]);
        }
    }

    static int[] dijkstra(int adj[][], int src) {
        int n = adj.length;
        int[] dist = new int[n];
        boolean[] visited = new boolean[n];
        int min_dist, unvis = -1;

        for (int i = 0; i < n; i++) {
            dist[i] = adj[src - 1][i];
            visited[i] = false;
        }

        visited[src - 1] = true;

        for (int i = 1; i < n; i++) {
            min_dist = Integer.MAX_VALUE;
            for (int j = 0; j < n; j++) {
                if (!visited[j] && dist[j] < min_dist) {
                    unvis = j;
                    min_dist = dist[j];
                }
            }

            visited[unvis] = true;

            for (int v = 0; v < n; v++) {
                if (!visited[v] && dist[unvis] + adj[unvis][v] < dist[v]) {
                    dist[v] = dist[unvis] + adj[unvis][v];
                }
            }
        }

        return dist;
    }
}
```

Output

```
Enter Number of Vertices
5
Enter Adjacency Matrix
0
3
99
7
99
3
0
4
2
99
99
4
0
5
6
7
2
5
0
4
99
99
6
4
```

```
0
Enter Source vertex
1
Shortest Distance from 1 to 2 is 3
Shortest Distance from 1 to 3 is 7
Shortest Distance from 1 to 4 is 5
Shortest Distance from 1 to 5 is 9
```

6. 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.Scanner;
public class Kruskal_8 {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      int n;
      System.out.println("Enter the number of vertices");
      n = sc.nextInt();
      int cA[][] = new int[n][n];
      System.out.println("Enter the cost adjacency matrix.Enter 9999 for infinity");
      for (int i = 0; i < n; i++)
         for (int j = 0; j < n; j++)
            cA[i][j] = sc.nextInt();
      int edges[] = new int[100];
      int source[] = new int[100];
      int destination[] = new int[100];
      int k = 0;
      for (int i = 0; i < n; i++)// Creating an edges array which contain the
         // edges and corresponding source and
         // destination arrays for each edge from the
         // cost adjacency matrix
         for (int j = 0; j < n; j++) {
            if ((cA[i][j] != 0) && (cA[i][j] != 9999)) {
               edges[k] = cA[i][j];
               source[k] = i;
               destination[k] = j;
               cA[j][i] = 0;
               k++;
            }
         }
      for (int i = 0; i < k - 1; i++)// Sorting the edges array in increasing
         // order and swapping the corresponding
         // source and destination(Bubble Sort)
         for (int j = 0; j < k - i - 1; j++) {
            if (edges[j] > edges[j + 1]) {
               swap(edges, j);
               swap(source, j);
               swap(destination, j);
            }
         }
      int parent[] = new int[n];
      int minimumCost = 0;
      System.out.println("The edges in the MCST are:");
      for (int i = 0; i < n; i++)
         parent[i] = -1;
      for (int j = 0; j < k; j++) {
         int a = find(parent, source[j]);// Applying union find algorithm to
         // detect cycle in the graph
         int b = find(parent, destination[j]);
```

```
            if (a != b) {
                minimumCost += edges[j];
                union(parent, a, b);
                System.out.println((source[j]+1) + " -> " + (destination[j]+1) + " = " + edges[j]);
            }
        }
        System.out.println("Minimum Cost of the Spanning Tree = " + minimumCost);
        sc.close();
    }

    public static void swap(int array[], int index) {
        int temp = array[index];
        array[index] = array[index + 1];
        array[index + 1] = temp;
    }

    public static int find(int parent[], int i) {
        if (parent[i] == -1)
            return i;
        return find(parent, parent[i]);
    }

    public static void union(int parent[], int a, int b) {
        parent[a] = b;
    }
}
```

Output:

```
Enter the number of vertices
4
Enter the cost adjacency matrix.Enter 9999 for infinity
0 10 10 99
10 0 99 10
10 99 0 10
99 10 10 0
The edges in the MCST are:
1 -> 2 = 10
1 -> 3 = 10
2 -> 4 = 10
  Minimum Cost of the Spanning Tree = 30
```

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

```
import java.util.Scanner;
public class Prim_9 {
   public static void main(String[] args) {
      int n;
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter the number of vertices");
      n = sc.nextInt();
      int cA[][] = new int[n][n];
      int visited[] = new int[n];
      int unvisited[] = new int[n];
      System.out.println("Enter the cost adjacency matrix");
      for (int i = 0; i < n; i++)
         for (int j = 0; j < n; j++)
            cA[i][j] = sc.nextInt();
      System.out.println("Enter the source vertex");
      int sV = sc.nextInt();
      visited[sV - 1] = 1;
      for (int i = 0; i < n; i++) {
         if (i != sV - 1)
            unvisited[i] = 1;
      }
      int index, minCost, source;
      index = minCost = source = 0;
      System.out.println("Edge set :");
      for (int i = 1; i < n; i++) {
         int min = 9999;
         for (int j = 0; j < n; j++) {
            if (visited[j] == 1) {
               for (int k = 0; k < n; k++) {
                  if ((unvisited[k] == 1) && (cA[j][k] != 9999)) {
                     if (min > cA[j][k]) {
                        min = cA[j][k];
                        source = j;
                        index = k;
                     }
                  }
               }
            }
         }
         visited[index] = 1;
         unvisited[index] = 0;
         System.out.print("(" + (source + 1) + "," + (index + 1) + ")");
         minCost += min;
      }
      System.out
            .println("\nThe minimum cost of the spanning tree = " + minCost);
   }}
```

Output:

```
Enter the number of vertices
4
Enter the cost adjacency matrix
0 10 10 99
10 0 99 10
10 99 0 10
99 10 10 0
Enter the source vertex
1
Edge set :
(1,2)(1,3)(2,4)
The minimum cost of the spanning tree = 30
```

8. Write Java programs to implement All-Pairs Shortest Paths problem using Floyd's algorithm.

```
import java.util.Scanner;
public class Floyd_10_a {
   public static void main(String args[])
   {
      Scanner Sc =new Scanner(System.in);
      System.out.println("Enter the total number of nodes");
      int n=Sc.nextInt();
      System.out.println("Enter the adjacency matrix");
      int d[][]= new int[n][n];
      for(int i=0;i<n;i++){
         for(int j=0;j<n;j++){
            d[i][j]=Sc.nextInt();
         }
      }
      for(int k=0;k<n;k++){
         for(int j=0;j<n;j++){
            for(int i=0;i<n;i++){
               d[i][j]=Math.min(d[i][j],d[i][k]+d[k][j]);
            }
         }
      }
      for(int i=0;i<n;i++){
         for(int j=0;j<n;j++){
            System.out.println("the matrix is "+d[i][j]);
         }
      }
   }
}
```

Output:

```
Enter the total number of nodes
4
Enter the adjacency matrix
0 99 3 99
2 0 99 99
99 7 0 1
6 99 99 0
The matrix is
0 10 3 4
2 0 5 6
7 7 0 1
6 16 9 0
```

9. Write Java programs to implement Travelling Salesperson problem using Dynamic programming.

```
import java.util.Scanner;
public class Tsp_10_b {
    public static void main(String args[])
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the number of cities");
        int n = scan.nextInt();
        int cost[][] = new int[n + 1][n + 1];
        int s[] = new int[n + 1];
        System.out.println("Enter the cost adjacency matrix");
        for(int i = 1; i <= n; i++)
        {
            for(int j = 1; j <= n; j++)
                cost[i][j] = scan.nextInt();
            s[i] = 1;
        }
        int a = g(1, n - 1, cost, s, n);
        System.out.println("The minimum cost in optimal tour is "+a);
        int b = approx(cost, n);
        int approx_val = ((b * 100) / a) - 100;
        System.out.println("The approximation tour is "+approx_val);
    }

    static int g(int i, int s, int cost[][], int set[], int n)
    {
        int newset[] = new int[n + 1];
        int selected[] = new int[n + 1];
        for(int j = 1; j <= n; j++)
            newset[j] = set[j];
        newset[i] = 0;
        if(s == 0)
            return cost[i][1];
        for(int k = 0, j = 2; k < s; j++)
        {
            if(newset[j] != 0) //CHECK THIS
            {
                selected[k] = cost[i][j] + g(j,s-1,cost,newset,n);
                k++;
            }
        }
        int min = 999;
        for(int j = 0; j < s; j++)
        {
            if(selected[j]<min)
            {
                min=selected[j];
            }
        }
        return min;
    }

    static int approx(int cost[][], int n)
    {
        int visited[] = new int[n + 1];
        for(int i = 1; i <= n; i++)
        {
            visited[i]=0;
        }
        visited[1] = 1;
        int dist = 0;
        int next = 0;
        for(int i = 1; i < n; i++)
        {
            int min = 999;
            for(int j = 1; j <= n; j++)
            {
                if((cost[i][j] != 0) && (cost[i][j] < min) && (visited[j] == 0))
                {
                    next=j;
                    min=cost[i][j];
                }
            }
            dist=dist+min;
            visited[next]=1;
        }
        dist = dist + cost[next][1];
        System.out.println("Minimum cost using approximation algorithm is "+dist);
        return dist;
    }
}
```

Output

```
Enter the number of cities
4
Enter the cost adjacency matrix
0  10 15 20
5   0   9  10
6  13  0  12
8   8   9   0
The minimum cost in optimal tour is 35
Minimum cost using approximation algorithm is 39
The approximation tour is 11
```

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

```
import java.util.*;

public class Knapsack_DynamicProgramming
{
   public static void main(String[] args)
   {
      Scanner inscan = new Scanner(System.in);
      System.out.println("Enter number of items:");
      int items = inscan.nextInt();
      System.out.println("Enter maximum capacity of the Knapsack:");
      int capacity = inscan.nextInt();
      System.out.println("Enter the weights of items:");
      int[] weight = new int[items+1];
      for(int i=1; i<=items; i++)
         weight[i] = inscan.nextInt();
      System.out.println("Enter the values of items:");
      int[] value = new int[items+1];
      for(int i=1; i<=items; i++)
         value[i] = inscan.nextInt();
      knapsack(items, capacity, weight, value);
   }

   public static void knapsack(int items, int capacity, int[] weight, int[] value)
   {
      Scanner inscan = new Scanner(System.in);
      int[][] soln_matrix = new int[items+1][capacity+1];
      for(int i=0; i<=items; i++)
         for(int j=0; j<=capacity; j++)
         {
            if(i==0 || j==0)
               soln_matrix[i][j] = 0;
         }
      for(int i=1; i<=items; i++)
         for(int j=1; j<=capacity; j++)
         {
            if( weight[i] > j)
               soln_matrix[i][j] = soln_matrix[i-1][j];
            else
               soln_matrix[i][j]  =  maximum(  soln_matrix[i-1][j]  ,  (soln_matrix[i-1][j-weight[i]]  +
value[i]));
         }
      profit_table( items, capacity, soln_matrix);
      selected_items( items, capacity, soln_matrix, weight);
      System.out.println("Maximum value that can be put in knapsack of capacity "+capacity+" is:
"+soln_matrix[items][capacity]);
   }

   public static int maximum( int a, int b)
   {
      return (a>b)?a:b;
```

```
    }

    public static void profit_table(int items, int capacity, int[][] soln_matrix)
    {
        System.out.println("Product table:");
        for(int i=0; i<=items; i++)
        {
            for(int j=0; j<=capacity; j++)
                System.out.print(soln_matrix[i][j]+"\t");
            System.out.println();
        }
    }

    public static void selected_items(int items, int capacity, int[][] soln_matrix, int[] weight)
    {
        System.out.println("Selected items:");
        while(items>0 && capacity>0)
        {
            if(soln_matrix[items][capacity] != soln_matrix[items-1][capacity])
            {
                System.out.println(items);
                capacity -= weight[items];
                items -= 1;
            }
            else
                items -= 1;
        }
    }
}
```

Output

```
Enter the weights of items:
2
1
3
2
Enter the values of items:
12
10
20
15
Product table:
0       0       0       0       0       0
0       0       12      12      12      12
0       10      12      22      22      22
0       10      12      22      30      32
0       10      15      25      30      37
Selected items:
4
2
1
  Maximum value that can be put in knapsack of capacity 5 is: 37
```

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} andd= 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.Scanner;
public class Subset_11 {
   static int set[]=new int[10];
   static int solvector[]=new int [10];
   static int target;
   static int count=0;
   public static void main(String[] args)
   {
      Scanner sc=new Scanner(System.in);
      System.out.println("Enter the number of items: ");
      int no_ele= sc.nextInt();
      System.out.println("Enter the numbers in ascending order");
      for(int i=0; i<no_ele;i++)
         set[i]=sc.nextInt();
      System.out.println("Enter the target sum");
      target=sc.nextInt();
      int tot_sum=0;
      for(int i=0;i<no_ele;i++)
      {
         tot_sum=tot_sum+set[i];
      }



      if(target>tot_sum)
      {
         System.out.println("Solution doesnt Exist");
         System.exit(0);
      }
      System.out.println("The solutions are");
      Subset(0,0,tot_sum);
   }

   public static void Subset(int sumsofar, int  index,int remsum)
   {
      solvector[index]=1;
      if(sumsofar+set[index]==target)
      {
         System.out.println("Solution No: = "+ (++count));
         for(int i=0;i<=index;i++)
         {
            if(solvector[i]==1)
               System.out.println(" "+set[i]);
         }
      }
      else if(sumsofar+set[index]+set[index+1]<=target)
```

```
        {
          Subset(sumsofar+set[index], index+1, remsum-set[index]);
        }
        if((sumsofar+remsum-set[index]>=target)&& (sumsofar+set[index+1]<=target))
        {
          solvector[index]=0;
          Subset(sumsofar,index+1,remsum-set[index]);
        }
     }
  }
```

Output

```
Enter the number of items:
5
Enter the numbers in ascending order
2
3
4
5
6
Enter the target sum
7
The solutions are
Solution No: = 1
 2
 5
Solution No: = 2
 3
 4
```

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.Scanner;
public class Hamiltonian_cycle_12 {
   static int n;
   static int adj[][] = new int[10][10];
   static int x[] = new int[10];
   static int flag = 0;

   public static void main(String args[])
   {
      Scanner scan = new Scanner(System.in);
      System.out.println("Enter the number of vertices");
      n = scan.nextInt();
      System.out.println("Enter the adjacency matrix");
      for(int i = 1; i <= n; i++)
         for(int j = 1; j <= n; j++)
            adj[i][j] = scan.nextInt();
      x[1] = 1;
      Hamiltonian(2);
      if(flag == 0)
         System.out.println("No Hamiltonian cycle present for the given graph");
   }

   static void Hamiltonian(int k)
   {
      do
      {
         NextValue(k);
         if(x[k] == 0)
            return;
         if(k == n)
         {
            flag = 1;
            System.out.println("The Hamiltonian cycle is:");
            for(int i = 1; i <= n; i++)
               System.out.print(x[i]+" ");
            System.out.print("1\n");
         }
         else
            Hamiltonian(k + 1);
      }while(true);
   }

   static void NextValue(int k)
   {
      int j;
      do
      {
         x[k] = (x[k] + 1) % (n + 1);
```

```
            if(x[k] == 0)
                return;
            if(adj[x[k - 1]][x[k]] == 1)
            {
                for(j = 1; j <= k - 1; j++)
                    if(x[j] == x[k])
                        break;
                if(j == k)
                    if((k < n) || ((k == n) && adj[x[n]][x[1]] == 1))
                        return;
            }
        }while(true);
    }
}
```

Output

```
Enter the number of vertices
5
Enter the adjacency matrix
0 1 1 1 99
1 0 99 1 99
1 99 0 1 1
1 1 1 0 1
99 99 1 1 0
The Hamiltonian cycle is:
1 2 4 5 3 1
The Hamiltonian cycle is:
1 3 5 4 2 1
```

Appendix-A

This section lists the Viva questions for each lab session:

RNSIT-CSE Bengaluru

SELECTION SORT

1. As the input size increases, the performance of selection sort decreases. (T/F)
2. What is the best case, worst case, average case time complexity for quick sort algorithm?
3. Explain working of selection sort

QUICK SORT

1. Which sorting technique is also called partition exchange sort?
2. Which is the technique used by quick sort?
3. What is the best case, worst case, average case time complexity for quick sort algorithm?
4. Explain divide and conquer technique.
5. Define in place sorting algorithm.
6. List different ways of selecting pivot element.
7. Is quick sort stable?

MERGE SORT

1. What is the best case, worst case, average case time complexity for merge sort algorithm?
2. What is the output of merge sort after the 1st pass given the following sequence of numbers?
   25 57 48 37 12 92 86 33
3. What is the running time of merge sort?
4. What technique is used to sort elements in merge sort?
5. Is merge sort in place sorting algorithm?
6. Define stable sort algorithm. Is merge sort stable?

WARSHALL’S ALGORITHM

1. Define transitive closure.
2. Define topological sequence.
3. What is the time complexity of Warshall's algorithm?

KNAPSACK PROBLEM

1. Define knapsack problem.
2. Define the principle of optimality.
3. What is the optimal solution for knapsack problem?
4. What is the time complexity of knapsack problem?
5. Provide a use case where greedy knapsack cannot yield optimal solution.

SHORTEST PATHS ALGORITHM

1. What is the time complexity of Dijkstra’s algorithm?
2. Define cost matrix.
3. Define directed graph.
4. Define connected graph.

MINIMUM COST SPANNING TREE

1. What is the time complexity of Kruskal’s algorithm?
2. Define spanning tree.
3. Define minimum cost spanning tree.

GRAPH TRAVERSALS

1. Define graph, connected graph.
2. List the different graph traversals.
3. Explain DFS traversal.
4. Explain BFS traversal.
5. What are the time complexities of BFS and DFS algorithms?
6. What are the data structures used to implement DFS and BFS traversals?

SUM OF SUBSETS PROBLEM

1. Define Back-Tracking.
2. Explain Sum of subset problem.
3. What is the time complexity of sum of subset problem?
4. Let S= {1,2,3,4,5} and d= 6. Obtain the subsets whose sum leads to d

TRAVELLING SALESPERSON PROBLEM

1. Define Optimal Solution.
2. Explain Travelling Salesman’s Problem.
3. What is the time complexity of Travelling Salesman’s Problem?
4. For n- cities________solutions are possible.
5. Deterministic algorithms exist for TSP(True/False)
6. Is TSP problem is NP-Hard or NP-Complete? Explain

MINIMUM COST SPANNING TREE

1. What is Minimum Cost spanning Tree.
2. Explain Prim’s algorithm.
3. What is the time complexity of Prim’s algorithm?
4. Explain the concept of Kruskal’s algorithm
5. Derive the time complexity for Kruskal’s algorithm.

RNSIT-CSE Bengaluru

SINGLE SOURCE SHORTEST PATH

1. Define Single Source Shortest Path problem.
2. Explain positive and negative weight cycle along with some examples.
3. Estimate the time complexity of the Dijkstra's algorithm.

ALL PAIRS SHORTEST PATH

1. Define All Pair Shortest Path Floyd's algorithm (APSP)
2. What is the time complexity of Floyd's algorithm?
3. Define Distance Matrix.

BACKTRACKING

1. Define n- Queens's problem?
2. 3- Queen problem has a solution (T/ F)
3. DFS can be solved using backtracking (T/F).
4. What is promising and non-promising node?
5. Define Hamiltonian Cycle.

Appendix-A

This section lists the Viva questions for each lab session:

SELECTION SORT

1. As the input size increases, the performance of selection sort decreases. (T/F)
2. What is the best case, worst case, average case time complexity for selection sortalgorithm?
3. Explain the working of selection sort.
4.

QUICK SORT

1. Which sorting technique is also called partition exchange sort?
2. Which is the technique used by quick sort?
3. What is the best case, worst case, average case time complexity for quick sort algorithm?
4. Explain divide and conquer technique.
5. Define in place sorting algorithm.
6. List different ways of selecting pivot element.
7. Is quick sort stable?

MERGE SORT

1. What is the best case, worst case, average case time complexity for merge sort algorithm?
2. What is the output of merge sort after the 1st pass given the following sequence of numbers?
   25 57 48 37 12 92 86 33
3. What is the running time of merge sort?
4. What technique is used to sort elements in merge sort?
5. Is merge sort in place sorting algorithm?
6. Define stable sort algorithm. Is merge sort stable?

WARSHALL'S ALGORITHM

1. Define transitive closure.
2. Define topological sequence.
3. What is the time complexity of Warshall's algorithm?

KNAPSACK PROBLEM

1. Define knapsack problem.
2. Define the principle of optimality.
3. What is the optimal solution for knapsack problem?
4. What is the time complexity of knapsack problem?
5. Provide a use case where greedy knapsack cannot yield optimal solution.

SHORTEST PATHS ALGORITHM

1. What is the time complexity of Dijkstra’s algorithm?
2. Define cost matrix.
3. Define directed graph.
4. Define connected graph.
5. Define Single Source Shortest Path problem.
6. Explain positive and negative weight cycle along with some examples.
7. Estimate the time complexity of the Dijkstra’s algorithm.

MINIMUM COST SPANNING TREE

1. What is the time complexity of Kruskal’s algorithm?
2. Define spanning tree.
3. Define minimum cost spanning tree.

GRAPH TRAVERSALS

1. Define graph, connected graph.
2. List the different graph traversals.
3. Explain DFS traversal.
4. Explain BFS traversal.
5. What are the time complexities of BFS and DFS algorithms?
6. What are the data structures used to implement DFS and BFS traversals?

SUM OF SUBSETS PROBLEM

1. Define Back-Tracking.
2. Explain Sum of subset problem.
3. What is the time complexity of sum of subset problem?
4. Let S= {1,2,3,4,5} and d= 6. Obtain the subsets whose sum leads to d

TRAVELLING SALESPERSON PROBLEM

1. Define Optimal Solution.
2. Explain Travelling Salesman’s Problem.
3. What is the time complexity of Travelling Salesman’s Problem?
4. For n- cities________solutions are possible.
5. Deterministic algorithms exist for TSP(True/False)
6. Is TSP problem is NP-Hard or NP-Complete? Explain

MINIMUM COST SPANNING TREE

1. What is Minimum Cost spanning Tree.
2. Explain Prim’s algorithm.
3. What is the time complexity of Prim’s algorithm?
4. Explain the concept of Kruskal’s algorithm.
5. Derive the time complexity for Kruskal’s algorithm.

ALL PAIRS SHORTEST PATH

1. Define All Pair Shortest Path Floyd’s algorithm (APSP)
2. What is the time complexity of Floyd’s algorithm?
3. Define Distance Matrix.

BACKTRACKING

1. Define n- Queens’s problem?
2. 3- Queen problem has a solution (T/ F)
3. DFS can be solved using backtracking (T/F).
4. What is promising and non-promising node?
5. Define Hamiltonian Cycle.