DAYANANDA SAGAR COLLEGE OF ENGINEERING
(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi)
DEPARTMENT OF MASTER OF COMPUTER APPLICATIONS, BENGALURU-560078

ANALYSIS & DESIGN OF ALGORITHMS LABORATORY MANUAL

FACULTY: Prof Alamma B H
III Semester (22MCA37)

Sub. Code: 22MCA37	**CIA Marks: 50**
Hrs./Week: 03	**Exam Hrs: 03**
Total Hrs.: 42	**SEE Marks: 50**

Course Objectives:

CO1	Use various analyzing techniques for efficient design and development of algorithms.
CO2	Determine the Time and Space Complexity.
CO3	Use searching and sorting through analytical techniques to solve real world problems
CO4	Synthesize efficient algorithms in common engineering design situations.

- Analyze the asymptotic performance of algorithms.
- Write rigorous correctness proofs for algorithms.
- Demonstrate a familiarity with major algorithms and data structures
- Apply important algorithmic design paradigms and methods of analysis

Manual	Contents of the Manual
1.	**Implement Recursive Binary search and linear search and determine the time required to search an element. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.**
2.	**Sort a given set of elements using the Merge sort method and determine the time required to search an element. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.**
3.	**Obtain the Topological ordering of vertices in a given digraph.**
4.	**Sort a given set of elements using the insertion sort method and determine the time required to search an element. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.**
5.	**Implement Horspool algorithm for String Matching**
6.	**Implement 0/1 Knapsack problem using Dynamic programming.**
7.	**Implement the Minimum Cost Spanning Tree using greedy technique.** **Find Minimum Cost Spanning Tree of a given undirected graph usingKruskal's algorithm.**
8.	**Implement All-Pairs Shortest Paths Problem using Floyd's algorithm.**

9.	**Print all the nodes reachable from a given starting node in a digraph using BFS method.**
10.	**Check whether a given graph is connected or not using DFS method.**
11.	**Find Minimum Cost Spanning Tree of a given undirected graph using Prim's Algorithm.**
12.	**Coping with Limitations of Algorithm Power: Backtracking: Demonstrate subset and N Queen's problem using Back Tracking**

1. **Implement Recursive Binary search and linear search and determine the time required to search an element. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.**

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

public class SearchProgram {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Random random = new Random();

        System.out.print("Enter the size of the array: ");
        int size = scanner.nextInt();

        // Generate an array of random numbers
        int[] array = generateRandomArray(size, 1, 100, random);
        System.out.println("Generated Array: " + Arrays.toString(array));

        System.out.println("Choose search type:");
        System.out.println("1. Linear Search");
        System.out.println("2. Binary Search");

        int choice = scanner.nextInt();

        long startTime = System.nanoTime(); // Record start time

        switch (choice) {
            case 1:
                System.out.print("Enter the number to search: ");
                int linearSearchKey = scanner.nextInt();
                int linearSearchResult = linearSearch(array, linearSearchKey);
                if (linearSearchResult != -1)
                    System.out.println(linearSearchKey + " found at index " + linearSearchResult);
                else
                    System.out.println(linearSearchKey + " not found in the array.");
                break;

            case 2:
                // Binary search requires a sorted array
                Arrays.sort(array);
                System.out.print("Enter the number to search: ");
                int binarySearchKey = scanner.nextInt();
                int binarySearchResult = binarySearch(array, binarySearchKey);
                if (binarySearchResult != -1)
                    System.out.println(binarySearchKey + " found at index " + binarySearchResult);
                else
                    System.out.println(binarySearchKey + " not found in the array.");
                break;
```

```
            default:
                System.out.println("Invalid choice.");
        }

        long endTime = System.nanoTime(); // Record end time
        long duration = endTime - startTime;
        System.out.println("Time taken: " + duration + " nanoseconds");

        scanner.close();
    }

    // Linear Search Time Complexity: O(n)
    private static int linearSearch(int[] array, int key) {
        for (int i = 0; i < array.length; i++) {
            if (array[i] == key) {
                return i;
            }
        }
        return -1; // Key not found
    }

    // Binary Search Time Complexity: O(log n)
    private static int binarySearch(int[] array, int key) {
        int low = 0;
        int high = array.length - 1;

        while (low <= high) {
            int mid = (low + high) / 2;
            if (array[mid] == key)
                return mid; // Key found
            else if (array[mid] < key)
                low = mid + 1;
            else
                high = mid - 1;
        }

        return -1; // Key not found
    }

    private static int[] generateRandomArray(int size, int min, int max, Random random) {
        int[] array = new int[size];

        for (int i = 0; i < size; i++) {
            array[i] = random.nextInt(max - min + 1) + min;
        }

        return array;
    }
}
```

==========Output==========

1:

Enter the size of the array: 6

Generated Array: [15, 33, 95, 30, 62, 17]

Choose search type:

1. Linear Search

2. Binary Search

1

Enter the number to search: 74

74 not found in the array.

Time taken: 7925599000 nanoseconds

2:

Enter the size of the array: 8

Generated Array: [82, 70, 47, 97, 13, 78, 51, 5]

Choose search type:

1. Linear Search

2. Binary Search

2

Enter the number to search: 47

47 found at index 2

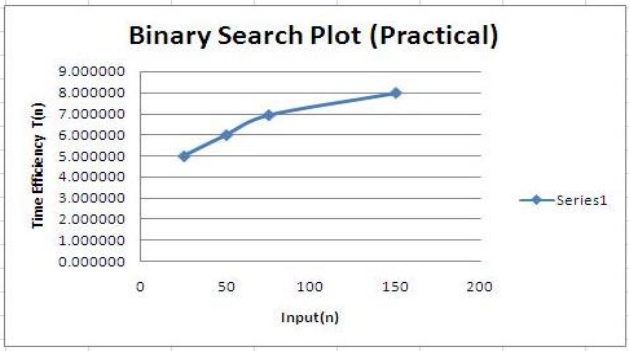
Time taken: 2583879900 nanoseconds

n	time taken
200	3083116800
100	3808853200
300	2293456800
50	1514937500
250	2338507500

Graph output:

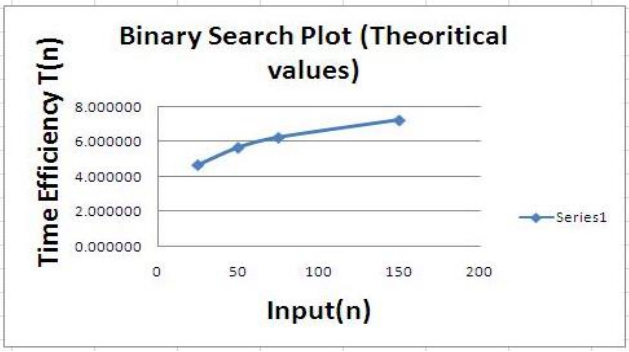
Binary Search Plot (Practical Values)

Input(n)	Time Efficiency T(n)
25	5.000000
50	5.989011
75	6.923077
150	7.967033



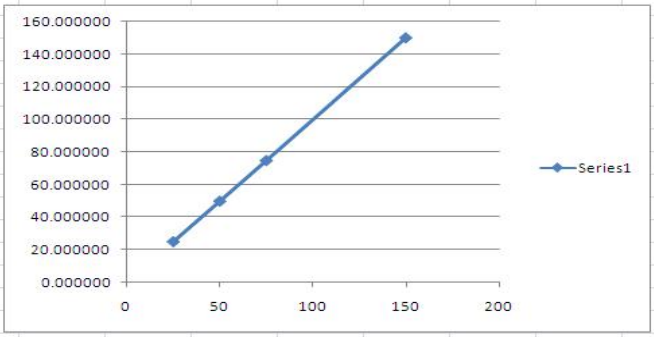
Binary Search Plot (Theoritcal Values)

Input(n)	Time Efficiency T(n) is log2n
25	4.643856
50	5.643856
75	6.228819
150	7.228819



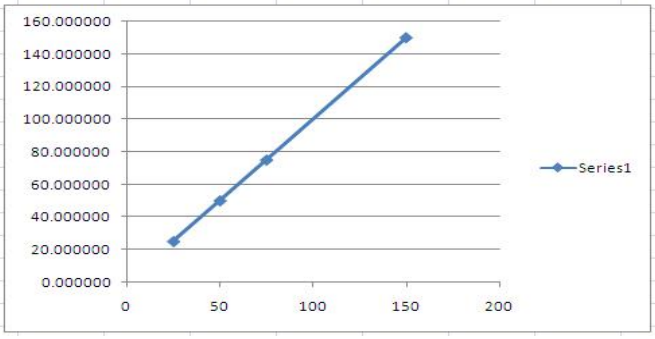
Linear Search Plot (Practical Values)

Input(n)	Time Efficiency T(n)
25	24.835165
50	49.670330
75	74.560440
150	149.758242



Linear Search Plot (Theoritcal Values)

Input(n)	Time Efficiency T(n) is log2n
25	25.000000
50	50.000000
75	75.000000
150	150.000000



2. Sort a given set of elements using the Merge sort method and determine the time required to search an element. Repeat the experiment for different values of n, thenumber of elements in the list to be searched and plot a graph of the time taken versus n.

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

public class MergeSortProgram {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Take input for the size of the array
        System.out.print("Enter the size of the array (n): ");
        int n = scanner.nextInt();

        Random random = new Random();

        // Generate an array of random numbers
        int[] array = generateRandomArray(n, 1, 100, random);
        System.out.println("Generated Array: " + Arrays.toString(array));

        long startTime = System.nanoTime(); // Record start time

        mergeSort(array, 0, array.length - 1);

        long endTime = System.nanoTime(); // Record end time
        long duration = endTime - startTime;
        System.out.println("Sorted Array: " + Arrays.toString(array));
        System.out.println("Time taken for Merge Sort: " + duration + " nanoseconds");

        // Close the scanner to prevent resource leak
        scanner.close();
    }

    // Merge Sort Time Complexity: O(n log n)
    private static void mergeSort(int[] array, int left, int right) {
        if (left < right) {
            int mid = (left + right) / 2;

            // Recursively sort the two halves
            mergeSort(array, left, mid);
```

```
        mergeSort(array, mid + 1, right);

        // Merge the sorted halves
        merge(array, left, mid, right);
    }
}

private static void merge(int[] array, int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    // Create temporary arrays
    int[] leftArray = new int[n1];
    int[] rightArray = new int[n2];

    // Copy data to temporary arrays
    System.arraycopy(array, left, leftArray, 0, n1);
    System.arraycopy(array, mid + 1, rightArray, 0, n2);

    // Merge the temporary arrays
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (leftArray[i] <= rightArray[j]) {
            array[k++] = leftArray[i++];
        } else {
            array[k++] = rightArray[j++];
        }
    }

    // Copy remaining elements if any
    while (i < n1) {
        array[k++] = leftArray[i++];
    }

    while (j < n2) {
        array[k++] = rightArray[j++];
    }
}

private static int[] generateRandomArray(int size, int min, int max, Random random) {
    int[] array = new int[size];

    for (int i = 0; i < size; i++) {
        array[i] = random.nextInt(max - min + 1) + min;
    }

    return array;
```

```
    }
}
```

==========Output==========

1:

Generated Array: [90, 13, 10, 63, 48, 82, 8, 68, 98, 64]

Sorted Array: [8, 10, 13, 48, 63, 64, 68, 82, 90, 98]

Time taken for Merge Sort: 15600 nanoseconds

2.

Generated Array: [13, 62, 25, 21, 79, 72, 65, 25, 77, 82]

Sorted Array: [13, 21, 25, 25, 62, 65, 72, 77, 79, 82]

Time taken for Merge Sort: 11700 nanoseconds

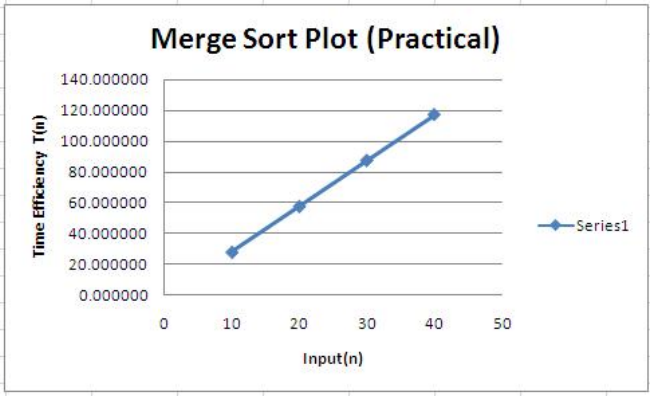
n	time taken
200	138700
100	57800
300	340700
50	31899
250	228500

Graph output:

Analysis of Merge Sort Algorithm

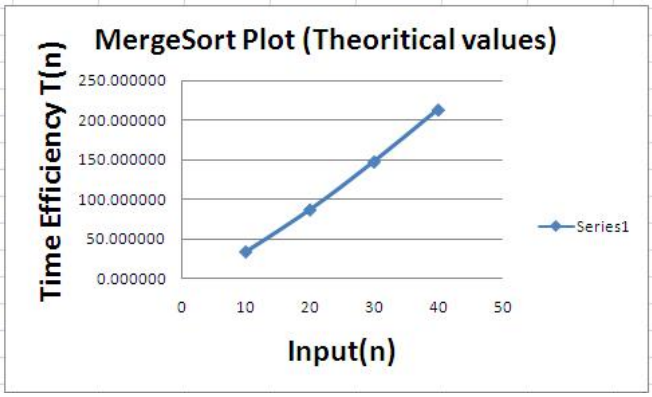
MergeSort Plot (Practical Values)

Input(n)	Time Efficiency T(n)
10	27.857143
20	57.582418
30	87.472527
40	117.197802



Merge Sort Plot (Theoritcal Values)

Input(n)	Time Efficiency T(n) is nlog2n
10	33.219281
20	86.438562
30	147.206718
40	212.877124



3. Obtain the Topological ordering of vertices in a given digraph.

```
import java.util.ArrayList;
import java.util.Stack;

class TopologicalSort {

   private int V; // Number of vertices
   private ArrayList<ArrayList<Integer>> adjList;

   TopologicalSort(int v) {
      V = v;
      adjList = new ArrayList<>(v);
      for (int i = 0; i < v; ++i)
         adjList.add(new ArrayList<>());
   }

   void addEdge(int v, int w) {
      adjList.get(v).add(w);
   }

   void topologicalSortUtil(int v, boolean[] visited, Stack<Integer> stack) {
      visited[v] = true;

      for (Integer neighbor : adjList.get(v)) {
         if (!visited[neighbor]) {
            topologicalSortUtil(neighbor, visited, stack);
         }
      }

      stack.push(v);
   }
```

```
    void topologicalSort() {
        Stack<Integer> stack = new Stack<>();
        boolean[] visited = new boolean[V];

        for (int i = 0; i < V; i++) {
            if (!visited[i]) {
                topologicalSortUtil(i, visited, stack);
            }
        }

        System.out.println("Topological Ordering:");
        while (!stack.isEmpty()) {
            System.out.print(stack.pop() + " ");
        }
    }

    public static void main(String[] args) {
        TopologicalSort g = new TopologicalSort(6);
        g.addEdge(5, 2);
        g.addEdge(5, 0);
        g.addEdge(4, 0);
        g.addEdge(4, 1);
        g.addEdge(2, 3);
        g.addEdge(3, 1);

        g.topologicalSort();
    }
}
```

==========Output==========

Topological Ordering:

5 4 2 3 1 0

4. **Sort a given set of elements using the insertion sort method and determine the time required to search an element. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.**

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

public class InsertionSortExample {

   public static void insertionSort(int[] arr) {
      int n = arr.length;
      for (int i = 1; i < n; ++i) {
         int key = arr[i];
         int j = i - 1;

         while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
         }
         arr[j + 1] = key;
      }
   }

   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter the value of n: ");
      int n = scanner.nextInt();
```

```
        Random rand = new Random();

        // Generate a random array of size n
        int[] elements = new int[n];
        for (int i = 0; i < n; i++) {
            elements[i] = rand.nextInt(10000);
        }

        System.out.println("Original array: " + Arrays.toString(elements));

        // Measure the time taken to sort the array
        long startTime = System.nanoTime();
        insertionSort(elements);
        long endTime = System.nanoTime();
        double sortTime = (endTime - startTime) / 1e9; // Convert to seconds

        System.out.printf("n=%d, Sort Time: %.6f seconds%n", n, sortTime);
        System.out.println("Sorted array: " + Arrays.toString(elements));
    }
}
```

==========Output==========

1.

Enter the value of n: 25

Original array: [68, 8485, 7198, 5986, 3278, 1282, 8080, 3163, 1153, 1407, 4782, 1768, 1026, 2677, 5737, 3155, 7821, 2496, 5817, 4710, 8008, 1823, 7081, 4464, 8612]

n=25, Sort Time: 0.000007 seconds

Sorted array: [68, 1026, 1153, 1282, 1407, 1768, 1823, 2496, 2677, 3155, 3163, 3278, 4464, 4710, 4782, 5737, 5817, 5986, 7081, 7198, 7821, 8008, 8080, 8485, 8612]

2.

Enter the value of n: 12

Original array: [7967, 1926, 5847, 7222, 6140, 1465, 7267, 9037, 9951, 6783, 3206, 1527]

n=12, Sort Time: 0.000008 seconds

Sorted array: [1465, 1527, 1926, 3206, 5847, 6140, 6783, 7222, 7267, 7967, 9037, 9951]

5. Implement Horspool algorithm for String Matching

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class HorspoolAlgorithm {

    public static int horspoolSearch(String text, String pattern) {
        int textLength = text.length();
        int patternLength = pattern.length();

        if (patternLength > textLength) {
            return -1; // Pattern cannot be longer than the text
        }

        Map<Character, Integer> shiftTable = preprocessShiftTable(pattern);

        int i = patternLength - 1;

        while (i < textLength) {
            int k = 0;

            while (k < patternLength && pattern.charAt(patternLength - 1 - k) == text.charAt(i - k)) {
                k++;
            }

            if (k == patternLength) {
                return i - patternLength + 1; // Match found
            } else {
                char mismatchChar = text.charAt(i);
                int shift = shiftTable.containsKey(mismatchChar) ? shiftTable.get(mismatchChar) : patternLength;
                i += shift;
```

```
        }
    }

    return -1; // No match found
}

private static Map<Character, Integer> preprocessShiftTable(String pattern) {
    Map<Character, Integer> shiftTable = new HashMap<>();

    int patternLength = pattern.length();

    for (int i = 0; i < patternLength - 1; i++) {
        char currentChar = pattern.charAt(i);
        int shift = patternLength - i - 1;
        shiftTable.put(currentChar, shift);
    }

    return shiftTable;
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter the text: ");
    String text = scanner.nextLine();

    System.out.print("Enter the pattern: ");
    String pattern = scanner.nextLine();

    int result = horspoolSearch(text, pattern);

    if (result != -1) {
        System.out.println("Pattern found at index " + result);
    } else {
        System.out.println("Pattern not found in the text");
    }
```

```
    }
}
```

==========Output==========

1.

Enter the text: Analysis in Java

Enter the pattern: sis

Pattern found at index 5

2.

Enter the text: Analysis Design of algorithm

Enter the pattern: ab

Pattern not found in the text

6. Implement 0/1 Knapsack problem using Dynamic programming.

Program:

```
import java.util.Scanner; class Knapsack
{
public static void main(String[] args)
{
Scanner sc=new Scanner(System.in); int object,m;
System.out.println("Enter the Total Objects"); object=sc.nextInt();
int weight[]=new int[object]; int profit[]=new int[object]; for(int i=0;i<object;i++)
{
System.out.println("Enter the Profit"); profit[i]=sc.nextInt(); System.out.println("Enter the weight");
weight[i]=sc.nextInt();
}
System.out.println("Enter the Knapsack capacity"); m=sc.nextInt();
double p_w[]=new double[object]; for(int i=0;i<object;i++)
{
p_w[i]=(double)profit[i]/(double)weight[i];
}
System.out.println(""); System.out.println("            ");
System.out.println("-----Data-Set    ");
System.out.print(" ");
System.out.println(""); System.out.print("Objects"); for(int i=1;i<=object;i++)
{
System.out.print(i+"          ");
}
System.out.println(); System.out.print("Profit "); for(int i=0;i<object;i++)
{
System.out.print(profit[i]+"          ");
}
System.out.println(); System.out.print("Weight "); for(int i=0;i<object;i++)
{
System.out.print(weight[i]+"          ");
```

```
}
System.out.println(); System.out.print("P/W          "); for(int i=0;i<object;i++)
{
System.out.print(p_w[i]+" ");
}
for(int i=0;i<object-1;i++)
{
for(int j=i+1;j<object;j++)
{
if(p_w[i]<p_w[j])
{
double temp=p_w[j]; p_w[j]=p_w[i]; p_w[i]=temp;
int temp1=profit[j]; profit[j]=profit[i]; profit[i]=temp1;
int temp2=weight[j]; weight[j]=weight[i]; weight[i]=temp2;
}
}
}
System.out.println(""); System.out.println("          ");
System.out.println("--After Arranging--");
System.out.print(" ");
System.out.println(""); System.out.print("Objects"); for(int i=1;i<=object;i++)
{
System.out.print(i+"          ");
}
System.out.println(); System.out.print("Profit "); for(int i=0;i<object;i++)
{
System.out.print(profit[i]+"          ");
}
System.out.println(); System.out.print("Weight "); for(int i=0;i<object;i++)
{
System.out.print(weight[i]+"          ");
}
System.out.println(); System.out.print("P/W          "); for(int i=0;i<object;i++)
{
System.out.print(p_w[i]+" ");
}
```

```
int k=0; double sum=0;
System.out.println(); while(m>0)
{
if(weight[k]<m)
{
sum+=1*profit[k]; m=m-weight[k];
}
else
{
double x4=m*profit[k]; double x5=weight[k]; double x6=x4/x5; sum=sum+x6;
m=0;
} k++;
}
System.out.println("Final Profit is="+sum);
}
}
```

==========Output==========

Enter the Total Objects

7

Enter the Profit

10

Enter the weight

5

Enter the Profit

2

Enter the weight

15

Enter the Profit

5

Enter the weight

7

Enter the Profit

7

Enter the weight

2

Enter the Profit

6

Enter the weight

2

Enter the Profit

16

Enter the weight

13

Enter the Profit

6

Enter the weight

2

Enter the Knapsack capacity

15

-----Data-Set

Objects1	**2**	**3**	**4**	**5**	**6**	**7**
Profit 10	**2**	**5**	**7**	**6**	**16**	**6**
Weight 5	**15**	**7**	**2**	**2**	**13**	**2**

P/W 2.0 0.13333333333333333 0.7142857142857143 3.5 3.0 1.2307692307692308 3.0

--After Arranging--

Objects1	**2**	**3**	**4**	**5**	**6**	**7**
Profit 7	**6**	**6**	**10**	**16**	**5**	**2**
Weight 2	**2**	**2**	**5**	**13**	**7**	**15**

P/W 3.5 3.0 3.0 2.0 1.2307692307692308 0.7142857142857143 0.13333333333333333

Final Profit is=33.92307692307692

7. Implement the Minimum Cost Spanning Tree using greedy technique.
Find Minimum Cost Spanning Tree of a given undirected graph usingKruskal's algorithm.

Program:

```
import java.util.Scanner; public class KRUSKAL
{
public static void main(String[] args)
{
int cost[][]=new int[10][10]; int i, j,mincost=0;
Scanner in = new Scanner(System.in);
System.out.println("********* KRUSKAL'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("The entered cost matrix is"); for(i=1;i<=n;i++){
for(j=1;j<=n;j++){ System.out.print(cost[i][j]+"\t");
}
System.out.println();
}
mincost=kruskals(n,mincost,cost); System.out.println("The minimum spanning tree cost is:");
System.out.println(mincost);
}
static int kruskals(int n,int mincost,int cost[][] )
{
int ne = 1,a=0,u=0,b=0,v=0,min; int parent[]=new int[10]; while(ne < n){
min=999;
for(int i=1; i<=n; i++)
{
for(int j=1; j<=n; j++)
{
if(cost[i][j] < min){
min = cost[i][j]; a=u=i;
b=v=j;
```

```
}
}
}
while(parent[u]>0) u = parent[u]; while(parent[v]>0) v = parent[v];
if(u != v)
{
System.out.print((ne++)+">minimum edge is :"); System.out.println("("+a+","+b+") and its cost is:"+min);
mincost += min;
parent[v] = u;
}
cost[a][b] = cost[b][a] = 999;
}
return mincost;
}
}
```

==========Output==========

Enter the number of nodes:

3

Enter the cost matrix 0

2

6

2

0

2

6

2

0

The entered cost matrix is

0 2 6

2 0 2

6 2 0

1>minimum edge is :(1,2) and its cost is:2

2>minimum edge is :(2,3) and its cost is:2

The minimum spanning tree cost is:

4

8. Implement All-Pairs Shortest Paths Problem using Floyd's algorithm.

Program:

```
import java.util.Scanner; class Floyd {
public static void main(String[] args)
{
int a[][]=new int[10][10]; int i, j;
Scanner in = new Scanner(System.in); System.out.println("**********FLOYD'SALGORITHM**********");
System.out.println("Enter the number of vertices: ");
int n = in.nextInt();
System.out.println("Enter the adjacency matrix"); for (i=1;i<=n;i++)
for (j=1;j<=n;j++) a[i][j] = in.nextInt();
System.out.println("Entered adjacency matrix is: "); for(i=1;i<=n;i++)
{
for(j=1; j<=n; j++)
{
System.out.print(a[i][j]+"\t");
}
System.out.println();
}
floyd(a,n);
System.out.println("All pair shortest path matrix:"); for (i=1; i<=n; i++)
{
for (j=1; j<=n; j++) System.out.print(a[i][j]+"\t"); System.out.println();
}
}
static void floyd(int a[][],int n)
{
for (int k=1; k<=n; k++)
{
for (int i=1; i<=n; i++) for (int j=1; j<=n; j++)
a[i][j] = min(a[i][j], a[i][k] + a[k][j]);
}
```

```
}
static int min(int a,int b)
{
if(a>b) return b; else return a;
}
}
```

==========Output==========

Enter the number of vertices:

4

Enter the adjacency matrix 0

1

3

1

2

0

5

999

4

7

999

1

33

2

1

3

Entered adjacency matrix is:

0 1 3 1

2 0 5 999

4 7 999 1

33 2 1 3

All pair shortest path matrix:

0 1 2 1

2 0 4 3

4 3 2 1

4 2 1 2

9. Print all the nodes reachable from a given starting node in a digraph using BFS method.

```
import java.util.*;

public class BFSExample {

    public static void bfs(int startNode, Map<Integer, List<Integer>> graph) {
        Set<Integer> visited = new HashSet<>();
        Queue<Integer> queue = new LinkedList<>();

        visited.add(startNode);
        queue.add(startNode);

        System.out.println("Nodes reachable from node " + startNode + " using BFS:");

        while (!queue.isEmpty()) {
            int current = queue.poll();
            System.out.print(current + " ");

            List<Integer> neighbors = graph.getOrDefault(current, Collections.emptyList());
            for (int neighbor : neighbors) {
                if (!visited.contains(neighbor)) {
                    visited.add(neighbor);
                    queue.add(neighbor);
                }
            }
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of nodes in the graph: ");
```

```
        int numNodes = scanner.nextInt();

        Map<Integer, List<Integer>> graph = new HashMap<>();

        System.out.println("Enter the directed edges (node1 -> node2): ");
        for (int i = 0; i < numNodes; i++) {
            System.out.print("Enter the number of edges for node " + i + ": ");
            int numEdges = scanner.nextInt();

            List<Integer> neighbors = new ArrayList<>();
            for (int j = 0; j < numEdges; j++) {
                System.out.print("Enter neighbor for node " + i + ": ");
                int neighbor = scanner.nextInt();
                neighbors.add(neighbor);
            }

            graph.put(i, neighbors);
        }

        System.out.print("Enter the starting node for BFS: ");
        int startNode = scanner.nextInt();

        bfs(startNode, graph);
    }
}
```

==========Output==========

Enter the number of nodes in the graph: 2
Enter the directed edges (node1 -> node2):
Enter the number of edges for node 0: 2
Enter neighbor for node 0: 6
Enter neighbor for node 0: 8
Enter the number of edges for node 1: 1
Enter neighbor for node 1: 3
Enter the starting node for BFS: 5
Nodes reachable from node 5 using BFS: 5

10. Check whether a given graph is connected or not using DFS method

```
import java.util.*;

public class DFSExample {

   public static void dfs(int current, Map<Integer, List<Integer>> graph, Set<Integer> visited) {
      visited.add(current);
      System.out.print(current + " ");

      List<Integer> neighbors = graph.getOrDefault(current, Collections.emptyList());
      for (int neighbor : neighbors) {
         if (!visited.contains(neighbor)) {
            dfs(neighbor, graph, visited);
         }
      }
   }

   public static void performDFS(Map<Integer, List<Integer>> graph) {
      Set<Integer> visited = new HashSet<>();

      for (int node : graph.keySet()) {
         if (!visited.contains(node)) {
            dfs(node, graph, visited);
         }
      }
   }

   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter the number of nodes in the undirected graph: ");
```

```
        int numNodes = scanner.nextInt();

        Map<Integer, List<Integer>> graph = new HashMap<>();

        System.out.println("Enter the edges (node1 - node2): ");
        for (int i = 0; i < numNodes; i++) {
            System.out.print("Enter the number of neighbors for node " + i + ": ");
            int numNeighbors = scanner.nextInt();

            List<Integer> neighbors = new ArrayList<>();
            for (int j = 0; j < numNeighbors; j++) {
                System.out.print("Enter neighbor for node " + i + ": ");
                int neighbor = scanner.nextInt();
                neighbors.add(neighbor);
            }

            graph.put(i, neighbors);
        }

        System.out.println("DFS traversal starting from each node:");
        performDFS(graph);
    }}
```

==========Output==========

Enter the number of nodes in the undirected graph: 2
Enter the edges (node1 - node2):
Enter the number of neighbors for node 0: 2
Enter neighbor for node 0: 4
Enter neighbor for node 0: 6
Enter the number of neighbors for node 1: 5
Enter neighbor for node 1: 7
Enter neighbor for node 1: 9
Enter neighbor for node 1: 10
Enter neighbor for node 1: 13
Enter neighbor for node 1: 1
DFS traversal starting from each node:
0 4 6 1 7 9 10 13

11. Find Minimum Cost Spanning Tree of a given undirected graph using Prim's Algorithm.

```
import java.util.Scanner; public class PRIM
{
public static void main(String[] args)
{
int cost[][]=new int[10][10];
int i, j, mincost = 0;
Scanner in = new Scanner(System.in);
System.out.println("********* PRIMS 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("The entered cost matrix is"); for(i=1; i<=n; i++)
{
for(j=1; j<=n; j++)
{
System.out.print(cost[i][j]+"\t");
}
System.out.println();
}
System.out.println("Minimum Spanning Tree Edges and costs are"); mincost=prims(cost,n,mincost);
System.out.print("The minimum spanning tree cost is:"); System.out.print(+mincost);
}
static int prims(int cost[][],int n,int mincost)
{
int nearV[]=new int[10],t[][]=new int[10][3],u = 0,i,j,k;
for(i=2; i<=n; i++)
```

```
nearV[i]=1; nearV[1]=0; for(i=1; i<n; i++)
{
int min=999; for(j=1;j<=n;j++)
{
if(nearV[j]!=0 && cost[j][nearV[j]]<min)
{
min=cost[j][nearV[j]]; u=j;
}
}
t[i][1] = u;
t[i][2] = nearV[u]; mincost += min; nearV[u] = 0; for(k=1; k<=n; k++){
if(nearV[k] != 0 && cost[k][nearV[k]] > cost[k][u]) nearV[k] = u;
}
System.out.print(i+") Minimum edge is ("+t[i][1]);
System.out.println(","+t[i][2]+") and its cost is :"+min);
}
return mincost;
}
}
```

==========Output==========

********** PRIMS ALGORITHM **********

Enter the number of nodes

3

Enter the cost matrix 0

2

999

2

0

1

999

1

0

The entered cost matrix is

0 2 999

2 0 1

999 1 0

Minimum Spanning Tree Edges and costs are

1) Minimum edge is (2,1) and its cost is :2

2) Minimum edge is (3,2) and its cost is :1

The minimum spanning tree cost is:3

12. Coping with Limitations of Algorithm Power: Backtracking:

Demonstrate subset and N Queen's problem using Back Tracking

```
public class NQueenProblem {
   final int N = 4;

   void printSolution(int board[][]) {
      for (int i = 0; i < N; i++) {
         for (int j = 0; j < N; j++)
            System.out.print(" " + board[i][j] + " ");
         System.out.println();
      }
   }

   boolean isSafe(int board[][], int row, int col) {
      int i, j;
      for (i = 0; i < col; i++)
         if (board[row][i] == 1)
            return false;

      for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
         if (board[i][j] == 1)
            return false;

      for (i = row, j = col; j >= 0 && i < N; i++, j--)
         if (board[i][j] == 1)
            return false;

      return true;
   }

   boolean solveNQUtil(int board[][], int col) {
```

```
    if (col >= N)
        return true;

    for (int i = 0; i < N; i++) {
        if (isSafe(board, i, col)) {
            board[i][col] = 1;
            if (solveNQUtil(board, col + 1))
                return true;
            board[i][col] = 0;
        }
    }
    return false;
}

boolean solveNQ() {
    int board[][] = { { 0, 0, 0, 0 },
                      { 0, 0, 0, 0 },
                      { 0, 0, 0, 0 },
                      { 0, 0, 0, 0 } };

    if (!solveNQUtil(board, 0)) {
        System.out.print("Solution does not exist");
        return false;
    }

    printSolution(board);
    return true;
}

public static void main(String args[]) {
    NQueenProblem Queen = new NQueenProblem();
    Queen.solveNQ();
}
```

```
}
```

==========Output==========

```
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
```