PROGRAM 1: TO SORT A LIST OF N ELEMENTS USING SELECTION SORT TECHNIQUE.

```
public class Lab1 {
public static void selectionSort(int[] arr) {
        for (int i = 0; i < arr.length - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < arr.length; j++)
                if (arr[j] < arr[minIndex]) minIndex = j;
            int temp = arr[i]; arr[i] = arr[minIndex]; arr[minIndex] = temp;
        }
    }

    public static void main(String[] args) {
        int[] arr = {9, 14, 3, 2, 43, 11, 58, 22};
        System.out.println("Before:" + java.util.Arrays.toString(arr));
        selectionSort(arr);
        System.out.println("After:" + java.util.Arrays.toString(arr));
    }
}
```

PROGRAM 2: TO PERFORM TRAVELING SALESMAN PROBLEM

```
import java.util.Scanner;

class Lab2 {
    static int findHamiltonianCycle(int[][] distance, boolean[] visited, int currPos, int cities, int
count, int cost) {
        if (count == cities) {
            return (distance[currPos][0] > 0) ? cost + distance[currPos][0] : Integer.MAX_VALUE;
        }

        int minCost = Integer.MAX_VALUE;
        for (int i = 0; i < cities; i++) {
            if (!visited[i] && distance[currPos][i] > 0) {
                visited[i] = true;
                minCost = Math.min(minCost, findHamiltonianCycle(distance, visited, i, cities, count +
1, cost + distance[currPos][i]));
                visited[i] = false;
            }
        }
        return minCost;
    }
```

```java
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter total number of cities: ");
        int cities = sc.nextInt();
        int[][] distance = new int[cities][cities];

        for (int i = 0; i < cities; i++) {
            for (int j = 0; j < cities; j++) {
                System.out.print("Distance from city " + (i + 1) + " to city " + (j + 1) + ": ");
                distance[i][j] = sc.nextInt();
            }
        }

        boolean[] visitCity = new boolean[cities];
        visitCity[0] = true;

        int hamiltonianCycle = findHamiltonianCycle(distance, visitCity, 0, cities, 1, 0);
        if (hamiltonianCycle == Integer.MAX_VALUE) {
            System.out.println("No Hamiltonian Cycle found.");
        } else {
            System.out.println("Minimum Distance: " + hamiltonianCycle);
        }
    }
}
```

PROGRAM 3: TO IMPEMENT THE BREADTH FIRST SEARCH (BFS) ALGORITHM FOR A GRAPH.

```java
import java.util.*;

class Graph {
    private List<List<Integer>> adj = new ArrayList<>();

    Graph(int v) {
        for (int i = 0; i < v; i++) adj.add(new ArrayList<>());
    }

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

    void BFS(int start) {
```

```
        boolean[] visited = new boolean[adj.size()];
        Queue<Integer> queue = new LinkedList<>();
        queue.add(start);
        visited[start] = true;

        while (!queue.isEmpty()) {
            int node = queue.poll();
            System.out.print(node + " ");
            for (int neighbor : adj.get(node)) {
                if (!visited[neighbor]) {
                    visited[neighbor] = true;
                    queue.add(neighbor);
                }
            }
        }
    }

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

PROGRAM 4: TO IMPLEMENT DEPTH FIRST TRAVERSAL (DFS)

```
import java.util.*;

class Graph {
    private List<List<Integer>> adj = new ArrayList<>();

    Graph(int v) {
        for (int i = 0; i < v; i++) adj.add(new ArrayList<>());
    }

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

    void DFS(int start) {
        boolean[] visited = new boolean[adj.size()];
        Stack<Integer> stack = new Stack<>();
```

```
        stack.push(start);

        while (!stack.isEmpty()) {
            int node = stack.pop();
            if (!visited[node]) {
                visited[node] = true;
                System.out.print(node + " ");
            }

            List<Integer> neighbors = adj.get(node);
            Collections.sort(neighbors, Collections.reverseOrder());
            for (int neighbor : neighbors) {
                if (!visited[neighbor]) {
                    stack.push(neighbor);
                }
            }
        }
    }

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

PROGRAM 5: TO FIND MINIMUM AND MAXIMUM VALUE IN AN ARRAY USING DIVIDE AND CONQURE.

```
public class MinMax {

    public static int[] minMax(int[] arr, int low, int high) {
        if (low == high) {
            return new int[]{arr[low], arr[low]};
        }
        if (high == low + 1) {
            return new int[]{Math.min(arr[low], arr[high]), Math.max(arr[low], arr[high])};
        }
        int mid = (low + high) / 2;
        int[] left = minMax(arr, low, mid);
        int[] right = minMax(arr, mid + 1, high);
        return new int[]{Math.min(left[0], right[0]), Math.max(left[1], right[1])};
```

```java
    }

    public static void main(String[] args) {
        int[] arr = {3, 5, 2, 8, 6, 9, 1, 4};
        int[] result = minMax(arr, 0, arr.length - 1);
        System.out.println("The minimum array element is " + result[0]);
        System.out.println("The maximum array element is " + result[1]);
    }
}
```

PROGRAM 6: TO IMPLEMENT MERGE SORT ALGORITHM FOR SORTING A LIST OF INTEGERS IN ASCENDING ORDER.

```java
public class MergeSort {

    public static void mergeSort(int[] arr) {
        if (arr.length < 2) return;
        int mid = arr.length / 2;
        int[] left = new int[mid];
        int[] right = new int[arr.length - mid];
        System.arraycopy(arr, 0, left, 0, mid);
        System.arraycopy(arr, mid, right, 0, arr.length - mid);
        mergeSort(left);
        mergeSort(right);
        merge(arr, left, right);
    }

    private static void merge(int[] arr, int[] left, int[] right) {
        int i = 0, j = 0, k = 0;
        while (i < left.length && j < right.length) arr[k++] = (left[i] <= right[j]) ? left[i++] : right[j++];
        while (i < left.length) arr[k++] = left[i++];
        while (j < right.length) arr[k++] = right[j++];
    }

    public static void main(String[] args) {
        int[] arr = {11, 30, 24, 7, 31, 16, 39, 41};
        System.out.print("Before sorting: ");
        for (int num : arr) System.out.print(num + " ");
        mergeSort(arr);
        System.out.print("\nAfter sorting: ");
        for (int num : arr) System.out.print(num + " ");
    }
}
```

PROGRAM 7: TO IMPLEMENT DIVIDE AND CONQUER STRATEGY FOR QUICK SORT ALGORITHM.

```
import java.util.Arrays;

public class QuickSort {
    public static void quickSort(int[] arr) {
        if (arr.length < 2) return; // Base case for empty or single element array
        quickSort(arr, 0, arr.length - 1);
    }

    private static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pivot = arr[low], i = low + 1, j = high;
            while (i <= j) {
                while (i <= high && arr[i] <= pivot) i++;
                while (arr[j] > pivot) j--;
                if (i < j) swap(arr, i, j);
            }
            swap(arr, low, j);
            quickSort(arr, low, j - 1);
            quickSort(arr, j + 1, high);
        }
    }

    private static void swap(int[] arr, int i, int j) {
        int temp = arr[i]; arr[i] = arr[j]; arr[j] = temp;
    }

    public static void main(String[] args) {
        int[] arr = {10, 7, 8, 6, 14, 2, 9};
        System.out.println("Original array: " + Arrays.toString(arr));
        quickSort(arr);
        System.out.println("Sorted array: " + Arrays.toString(arr));
    }
}
```

PROGRAM 8: TO IMPLEMENTS PRIMS ALGORITHM TO GENERATE MINIMUM COST SPANNING TREE.

```
    System.out.println("Enter the cost matrix:");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
```

```
            cost[i][j] = sc.nextInt();
            if (cost[i][j] == 0) cost[i][j] = Integer.MAX_VALUE;
        }
    }

    prim(n, cost);
}

private static void prim(int n, int[][] cost) {
    boolean[] visited = new boolean[n];
    visited[0] = true;
    int mincost = 0;

    for (int edges = 0; edges < n - 1; edges++) {
        int min = Integer.MAX_VALUE, a = -1, b = -1;

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (visited[i] && !visited[j] && cost[i][j] < min) {
                    min = cost[i][j];
                    a = i;
                    b = j;
                }
            }
        }

        if (!(a == -1) && !(b == -1)) { // Using ! with == to check for not equal
            System.out.println("Edge(" + a + "," + b + ") = " + min);
            mincost += min;
            visited[b] = true;
        }
    }

    System.out.println("The minimum cost of spanning tree is " + mincost);
}
```

PROGRAM 9: TO IMPLEMENTS KRUSKALS ALGORITHM TO GENERATE MINIMUM COST SPANNING TREE.

```
import java.util.Scanner;

public class KruskalsDemo {
    static int n, cost[][] = new int[20][20], parent[] = new int[20];
```

```java
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.print("Enter number of vertices: "); n = scan.nextInt();
        System.out.println("Enter cost adjacency matrix:");
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++) {
                cost[i][j] = scan.nextInt();
                if (cost[i][j] == 0) cost[i][j] = Integer.MAX_VALUE;
            }
        for (int i = 1; i <= n; i++) parent[i] = i;
        int minCost = 0, edges = 0;
        System.out.println("Edges of Minimum Cost Spanning Tree:");
        while (edges < n - 1) {
            int min = Integer.MAX_VALUE, a = -1, b = -1;
            for (int i = 1; i <= n; i++)
                for (int j = 1; j <= n; j++)
                    if (cost[i][j] < min) { min = cost[i][j]; a = i; b = j; }
            int u = find(a), v = find(b);
            if (u != v) {
                parent[v] = u; System.out.println(++edges + " edge (" + a + "," + b + ") = " + min);
                minCost += min;
            }
            cost[a][b] = cost[b][a] = Integer.MAX_VALUE;
        }
        System.out.println("Minimum cost: " + minCost);
    }
    static int find(int i) { return (parent[i] == i) ? i : (parent[i] = find(parent[i])); }
}
```