Task 1:
Kruskal's Algorithm for MST Implement Kruskal's algorithm to find the minimum spanning tree of a given connected, undirected graph with non-negative edge weights
ANS:

```
package Day16;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

class Edge {
    int src, dest, weight;

    public Edge(int src, int dest, int weight) {
        this.src = src;
        this.dest = dest;
        this.weight = weight;
    }
}

class Graph {
    int vertices;
    List<Edge> edges;

    public Graph(int vertices) {
        this.vertices = vertices;
        edges = new ArrayList<>();
    }

    public void addEdge(int src, int dest, int weight) {
        edges.add(new Edge(src, dest, weight));
    }

    public List<Edge> getEdges() {
        return edges;
    }
}

class Subset {
    int parent, rank;

    public Subset(int parent, int rank) {
        this.parent = parent;
        this.rank = rank;
    }
}
```

```
public class KruskalAlgorithm {

    public static int find(Subset[] subsets, int i) {
        if (subsets[i].parent != i) {
            subsets[i].parent = find(subsets, subsets[i].parent);
        }
        return subsets[i].parent;
    }

    public static void union(Subset[] subsets, int x, int y) {
        int rootX = find(subsets, x);
        int rootY = find(subsets, y);

        if (subsets[rootX].rank < subsets[rootY].rank) {
            subsets[rootX].parent = rootY;
        } else if (subsets[rootX].rank > subsets[rootY].rank) {
            subsets[rootY].parent = rootX;
        } else {
            subsets[rootY].parent = rootX;
            subsets[rootX].rank++;
        }
    }

    public static List<Edge> kruskalMST(Graph graph) {
        List<Edge> result = new ArrayList<>();
        List<Edge> edges = graph.getEdges();

        Collections.sort(edges, Comparator.comparingInt(edge -> edge.weight));

        Subset[] subsets = new Subset[graph.vertices];
        for (int i = 0; i < graph.vertices; ++i) {
            subsets[i] = new Subset(i, 0);
        }

        int i = 0;
        while (result.size() < graph.vertices - 1) {
            Edge nextEdge = edges.get(i++);

            int x = find(subsets, nextEdge.src);
            int y = find(subsets, nextEdge.dest);

            if (x != y) {
                result.add(nextEdge);
                union(subsets, x, y);
            }
        }
        return result;
    }
```

```
    public static void main(String[] args) {
        Graph graph = new Graph(4);
        graph.addEdge(0, 1, 10);
        graph.addEdge(0, 2, 6);
        graph.addEdge(0, 3, 5);
        graph.addEdge(1, 3, 15);
        graph.addEdge(2, 3, 4);

        List<Edge> mst = kruskalMST(graph);

        System.out.println("Edges in the MST:");
        for (Edge edge : mst) {
            System.out.println(edge.src + " -- " + edge.dest + " == " + edge.weight);
        }
    }
}
```

OUTPUT:

```
Edges in the MST:
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
```