Task 2: Kruskal's Algorithm for MST

find the Implement minimum spanning tree of a given connected, undirected graph with non-negative edge weights.

```java
package Day9_10;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
public class Kruskal {


//Java program for Kruskal's algorithm
	// Defines edge structure
	static class Edge {
		int src, dest, weight;

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

	// Defines subset element structure
	static class Subset {
		int parent, rank;

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

	// Starting point of program execution
	public static void main(String[] args)
	{
		int V = 4;
		List<Edge> graphEdges = new ArrayList<Edge>(
			List.of(new Edge(0, 1, 10), new Edge(0, 2, 6),
					new Edge(0, 3, 5), new Edge(1, 3, 15),
					new Edge(2, 3, 4)));

		// Sort the edges in non-decreasing order
		// (increasing with repetition allowed)
		graphEdges.sort(new Comparator<Edge>() {
			@Override public int compare(Edge o1, Edge o2)
			{
				return o1.weight - o2.weight;
			}
		});

		kruskals(V, graphEdges);
```

```
	}

	// Function to find the MST
	private static void kruskals(int V, List<Edge> edges)
	{
		int j = 0;
		int noOfEdges = 0;

		// Allocate memory for creating V subsets
		Subset subsets[] = new Subset[V];

		// Allocate memory for results
		Edge results[] = new Edge[V];

		// Create V subsets with single elements
		for (int i = 0; i < V; i++) {
			subsets[i] = new Subset(i, 0);
		}

		// Number of edges to be taken is equal to V-1
		while (noOfEdges < V - 1) {

			// Pick the smallest edge. And increment
			// the index for next iteration
			Edge nextEdge = edges.get(j);
			int x = findRoot(subsets, nextEdge.src);
			int y = findRoot(subsets, nextEdge.dest);

			// If including this edge doesn't cause cycle,
			// include it in result and increment the index
			// of result for next edge
			if (x != y) {
				results[noOfEdges] = nextEdge;
				union(subsets, x, y);
				noOfEdges++;
			}

			j++;
		}

		// Print the contents of result[] to display the
		// built MST
		System.out.println(
			"Following are the edges of the constructed MST:");
		int minCost = 0;
		for (int i = 0; i < noOfEdges; i++) {
			System.out.println(results[i].src + " -- "
									+ results[i].dest + " == "
									+ results[i].weight);
			minCost += results[i].weight;
		}
		System.out.println("Total cost of MST: " + minCost);
	}

	// Function to unite two disjoint sets
	private static void union(Subset[] subsets, int x,
									int y)
	{
		int rootX = findRoot(subsets, x);
```

```
            int rootY = findRoot(subsets, y);

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

        // Function to find parent of a set
        private static int findRoot(Subset[] subsets, int i)
        {
            if (subsets[i].parent == i)
                return subsets[i].parent;

            subsets[i].parent
                = findRoot(subsets, subsets[i].parent);
            return subsets[i].parent;
        }
}
```