DIVIDE AND CONQUER ALGORITHM (General Algorithm)

- In this approach ,we solve a problem recursively by applying 3 steps
  1. **DIVIDE**-break the problem into several sub problems of smaller size.
  2. **CONQUER**-solve the problem recursively.
  3. **COMBINE**-combine these solutions to create a solution to the original problem.

CONTROL ABSTRACTION FOR DIVIDE AND CONQUER ALGORITHM

```
Algorithm D and C (P)
{
    if small(P)
        then return S(P)
    else
        { divide P into smaller instances $P_1$ ,$P_2$ .....$P_k$
        Apply D and C to each sub problem
          Return combine (D and C($P_1$)+ D and C($P_2$)+.......+D and C($P_k$))
    }
```

Minimum Spanning Tree:

Prims Algorithm:

The algorithm may informally be described as performing the following steps:

1. Initialize a tree with a single vertex, chosen arbitrarily from the graph.
2. Grow the tree by one edge: of the edges that connect the tree to vertices not yet in the tree, find the minimum-weight edge, and transfer it to the tree.
3. Repeat step 2 (until all vertices are in the tree).

In more detail, it may be implemented following the pseudocode below.

1. Associate with each vertex *v* of the graph a number *C*[*v*] (the cheapest cost of a connection to *v*) and an edge *E*[*v*] (the edge providing that cheapest connection). To initialize these values, set all values of *C*[*v*] to +∞ (or to any number larger than the maximum edge weight) and set each *E*[*v*] to a special flag value indicating that there is no edge connecting *v* to earlier vertices.
2. Initialize an empty forest *F* and a set *Q* of vertices that have not yet been included in *F* (initially, all vertices).
3. Repeat the following steps until *Q* is empty:
   a. Find and remove a vertex *v* from *Q* having the minimum possible value of *C*[*v*]

b. Add *v* to *F* and, if *E*[*v*] is not the special flag value, also add *E*[*v*] to *F*
c. Loop over the edges *vw* connecting *v* to other vertices *w*. For each such edge, if *w* still belongs to *Q* and *vw* has smaller weight than *C*[*w*], perform the following steps:
    i. Set *C*[*w*] to the cost of edge *vw*
    ii. Set *E*[*w*] to point to edge *vw*.

4. Return *F*

Time complexity = $O(V \log E)$

Kruskals Algorithm:

// V is the set of vertices, E is the set of edges and W is the adjacency matrix to store the weights of the links. //

Time complexity = $O(E \log E)$

```
{
A = Φ ;
for (each vertex u in V)
        Make_set(u)
for (each least weight edge (u, v) in E) // least weight edge is the root of the heap//
        if (Find_set(u) ≠Find_set(v)){ // u and v are in two different sets //
                A = AU{u, v}
                Union(u, v)
}
}
return A ;
}
```

The Kruskal's algorithm first creates n trees from n vertices which is done in O(n) time. Then, a heap is created in O(n) time using heapify procedure. The least weight edge is at the root of the heap. Hence, the edges are deleted one by one from the heap and either added to the MST or discarded if it forms a cycle. This deletion process requires O(nlog2n). Hence, the time complexity of Kruskal's algorithm is O(nlog2n)