Name & Student Number:

1. Answer each of the following questions as True or False.
   (a) A MST contains a cycle. False
   (b) If we remove an edge from a MST, the resulting subgraph is still a MST. False
   (c) If we add an edge to a MST, the resulting subgraph is still a MST. False
   (d) If there are V vertices in a given graph, a MST of that graph contains $|V| - 1$ edges. True
   (e) Every MST is a sparse graph. True

   In sparse graph $|E| = O(|V|)$
   In dense graph $|E| = O(|V|^2)$

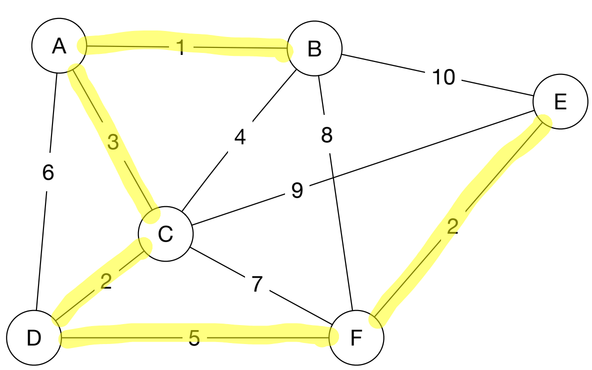
2. Following is the pseudocode for Kruskal's algorithm to find a MST.

```
function Kruskal(Graph G)
    initialize each vertex to be a component
    sort all edges by weight
    for each edge (u, v) in sorted order do
        if u and v are in different components then
            add edge (u,v) to the MST
            update u and v to be in the same component
        end if
    end for
end function
```

edgesAccepted = 0
while (edgesAccepted < |V|-1)
u,v is the next smallest edge

edgesAccepted++;

(a) Execute Kruskal's algorithm on the following graph. Fill the table.



Step	Components	Edge	Include?
1	{A} {B} {C} {D} {E} {F}	A, B	Yes
2	{A, B} {C} {D} {E} {F}	D, ~~B~~, C	Yes
3	{A, B} {C, D} {E} {F}	E, F	Yes
4	{A, B} {C, D} {E, F}	A, C	Yes
5	{A, B, C, D} {E, F}	B, C	No
6	— " —	D, F	Yes
7	{A, B, C, D, E, F}	A, D	No
8	— " —	C, F	No
9	— " —	B, F	No
10	— " —	C, E	No
11	— " —	E, B	No

(b) In this graph there are 6 vertices and 11 edges, and the for loop in the above pseudocode iterates 11 times, a few more times after the MST is found. How would you optimize the pseudocode so the for loop terminates early, as soon as a valid MST is found. Annotate the given pseudocode to add/edit lines.

Terminate for loop early, when MST has |V|-1 edges

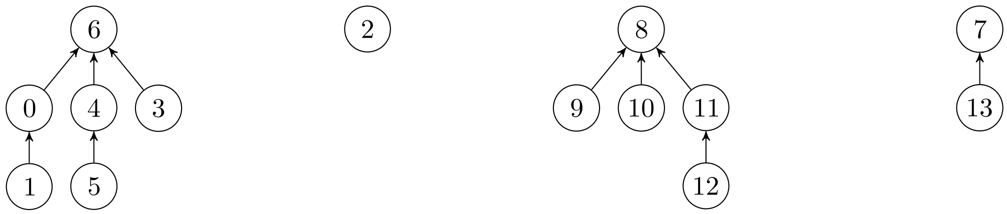
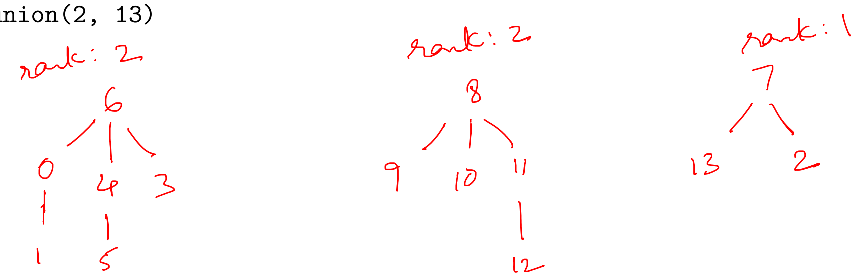
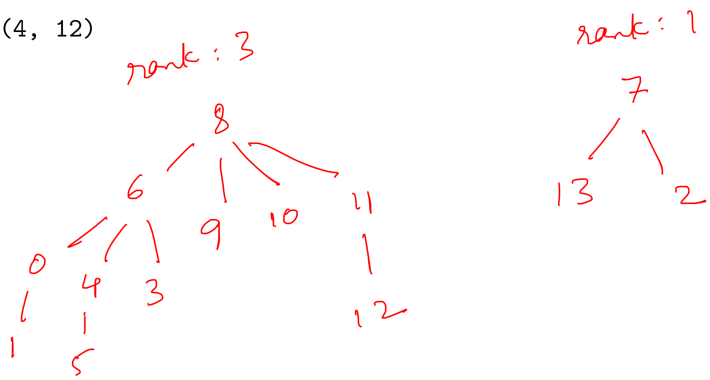


Figure 1: Disjoin-set. Rank of trees in the forest (from left): 2, 0, 2, 1.

3. Consider the disjoin-set shown in Figure ~~??~~ 1
   What would be the result of the following calls on union if we add the "union-by-rank" optimization. Draw the forest at each stage with corresponding ranks for each tree
   i. `union(2, 13)`



   ii. `union(4, 12)`



   iii. `union(2, 8)`