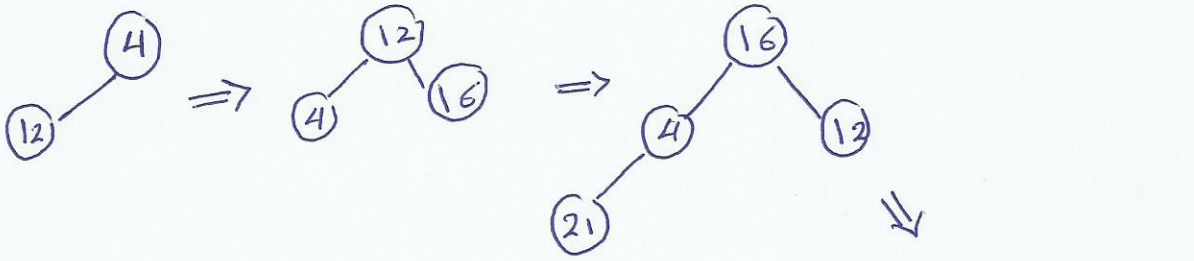
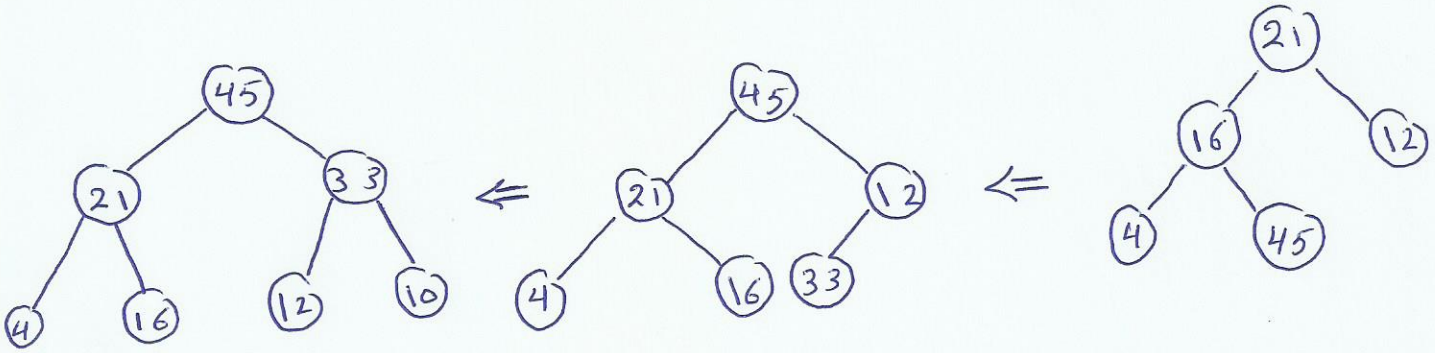
Homework 06

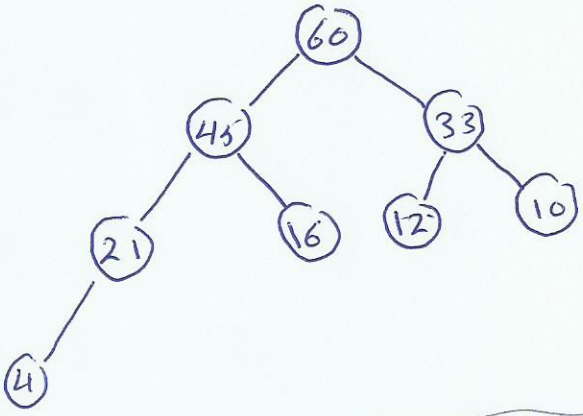
Solution

Problem 1:

1)



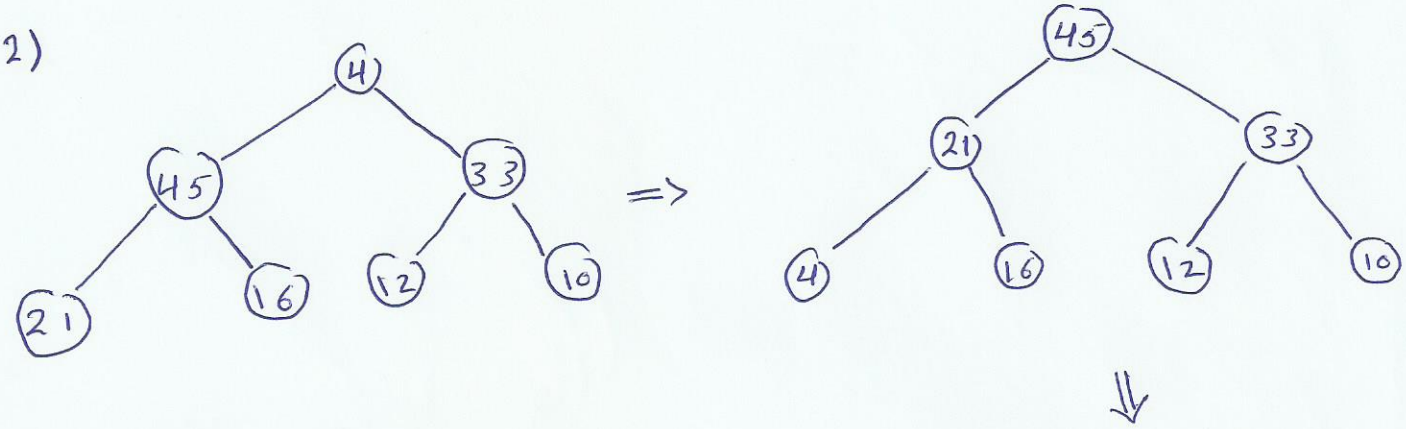


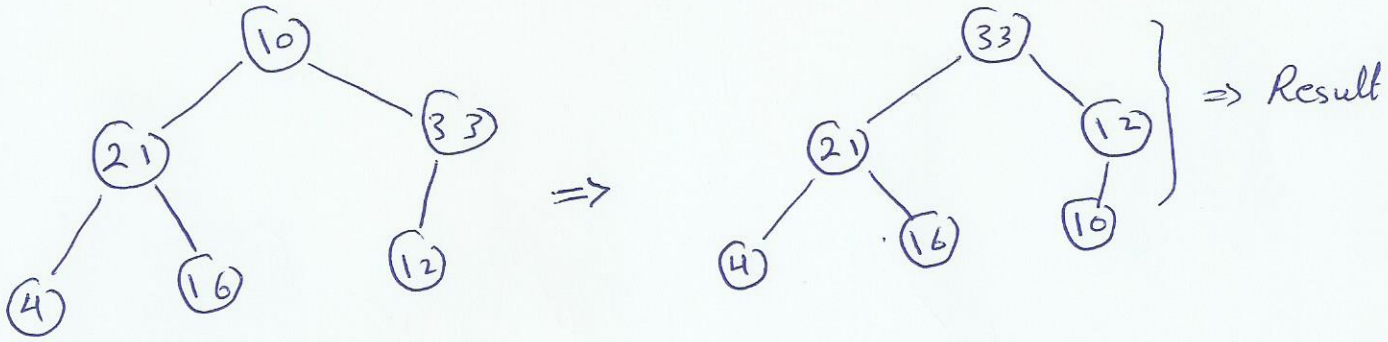


=> Result

---

2)





3) @ we will use min heap

b)

0	1	2	3	4	5	6
-	12	20	15	23	28	18

c) Delete:

0	1	2	3	4	5	6
-	18	20	15	23	28	-

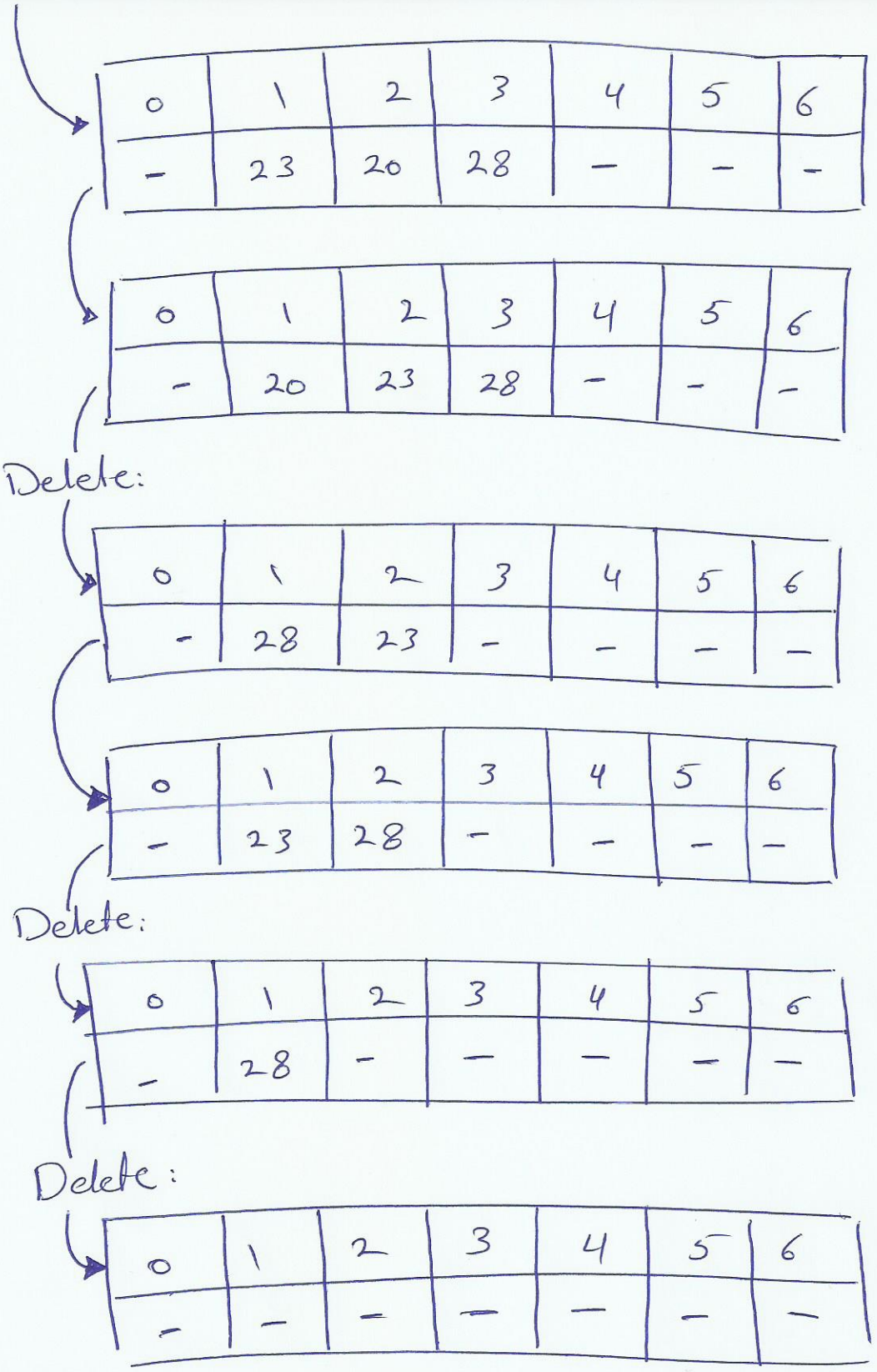
0	1	2	3	4	5	6
-	15	20	18	23	28	-

Delete:

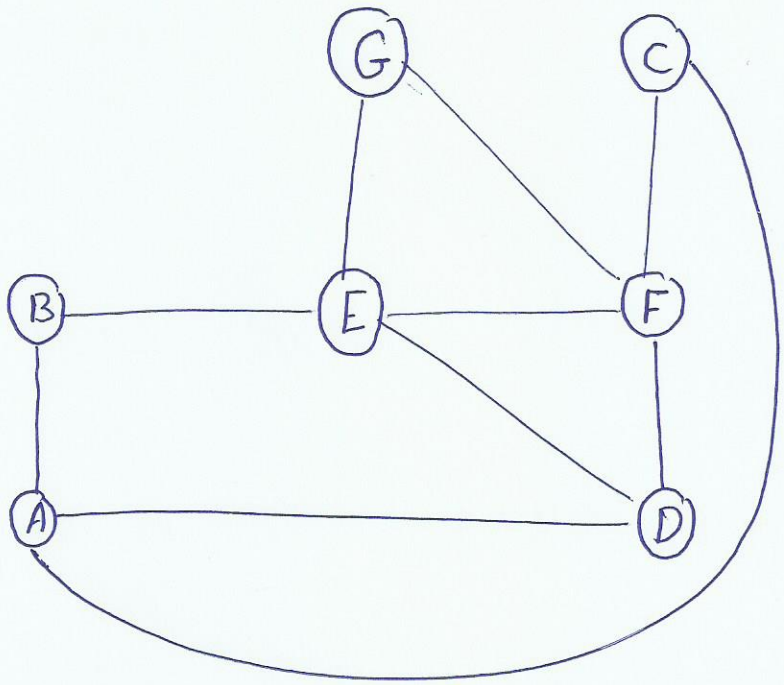
0	1	2	3	4	5	6
-	28	20	18	23	-	-

0	1	2	3	4	5	6
-	18	20	28	23	-	-

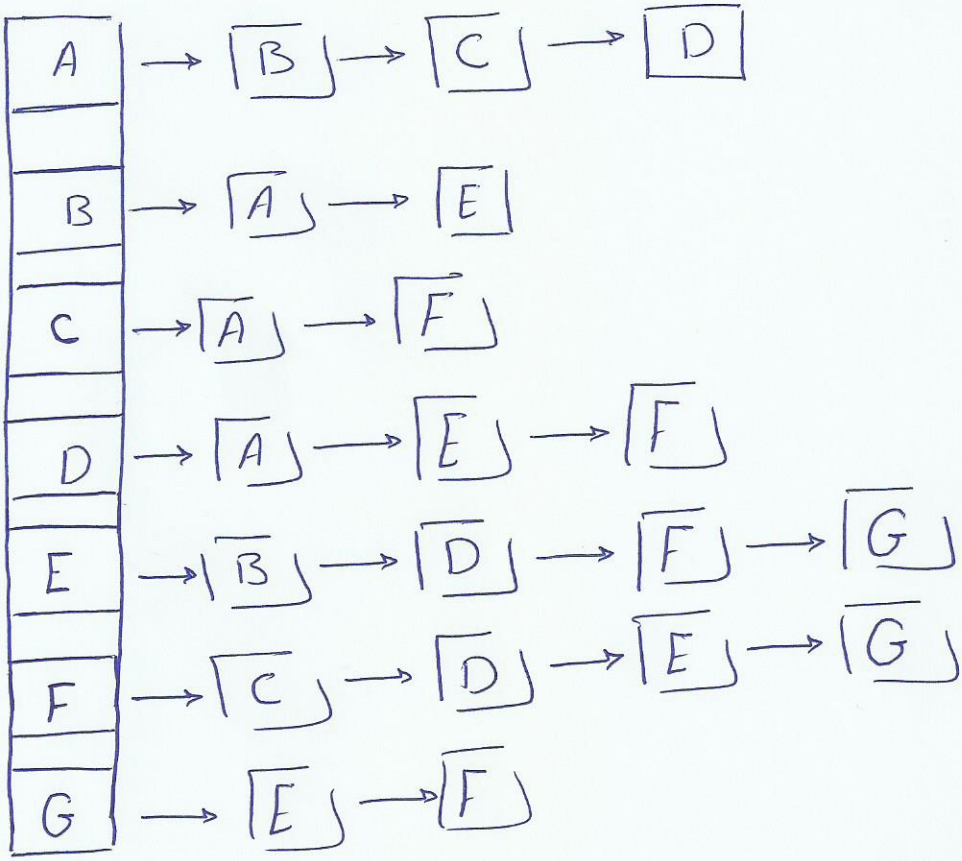
Delete:


0 1 2 3 4 5 6
- 23 20 28 - - -
0 1 2 3 4 5 6
- 20 23 28 - - -
Delete:
0 1 2 3 4 5 6
- 28 23 - - - -
0 1 2 3 4 5 6
- 23 28 - - - -
Delete:
0 1 2 3 4 5 6
- 28 - - - - -
Delete:
0 1 2 3 4 5 6
- - - - - - -

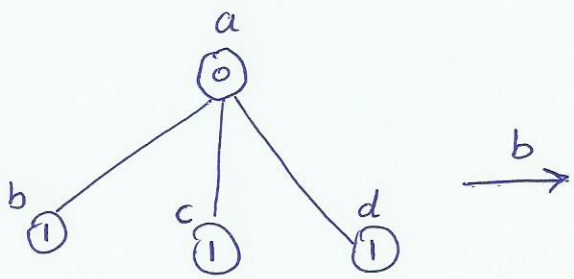
Problem 2:1:



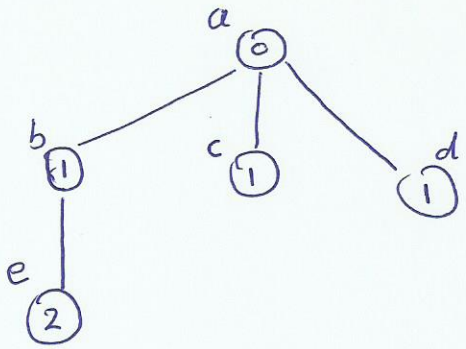
2:



3: BFS:



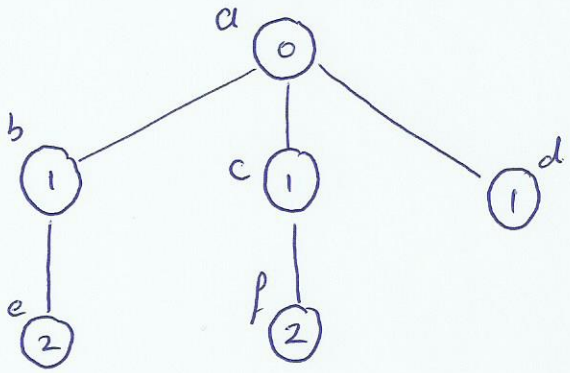
b



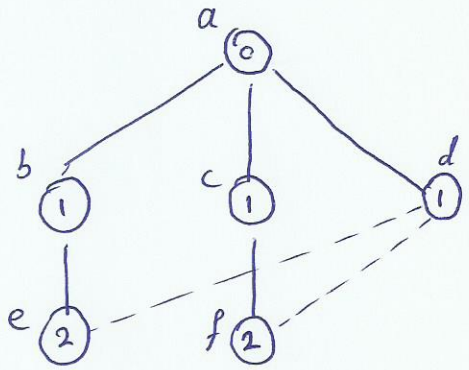
Q: b, c, d

Q: c, d, e

c



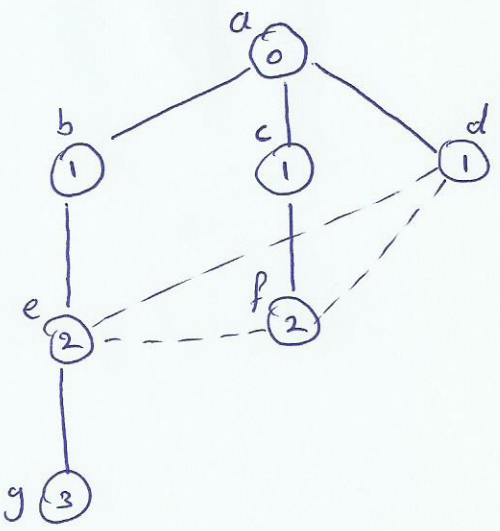
d



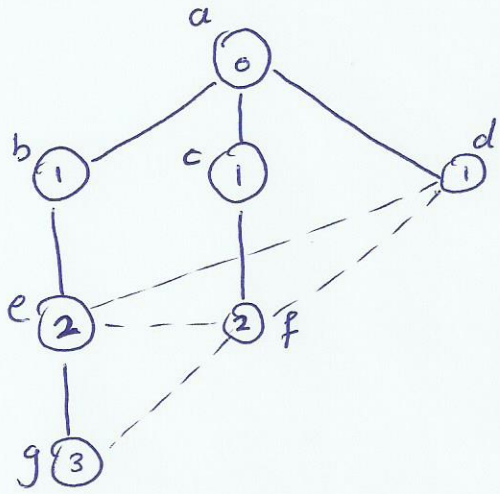
Q: d, e, f

Q: e, f

e



f, g

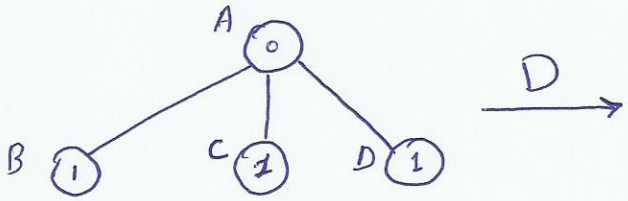


Result

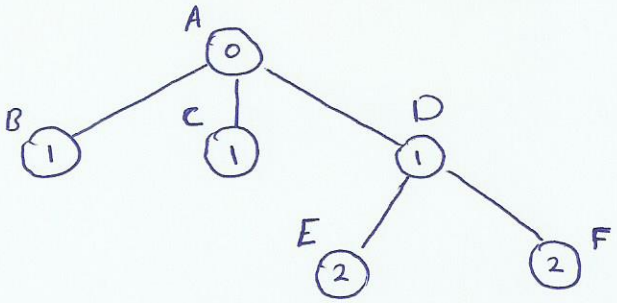
Q: f, g

Q:

DFS:



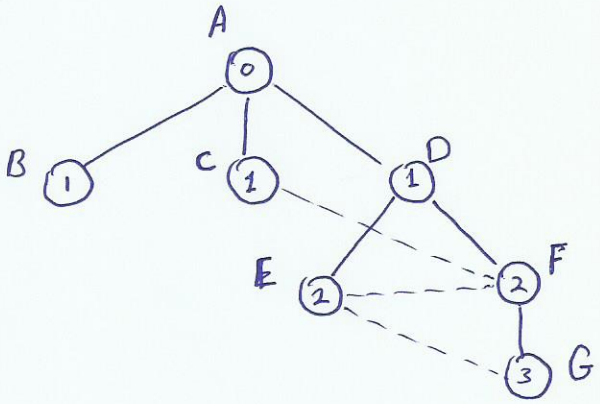
D



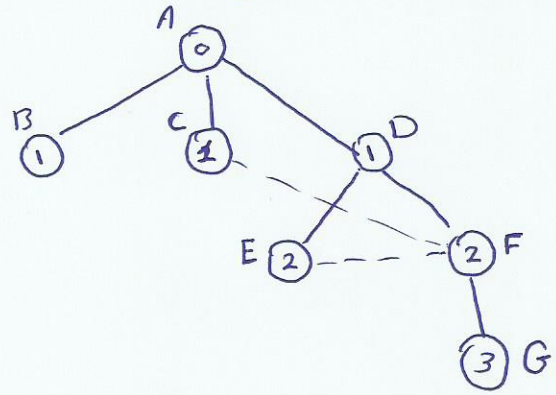
S: B, C, D

S: B, C, E, F

F



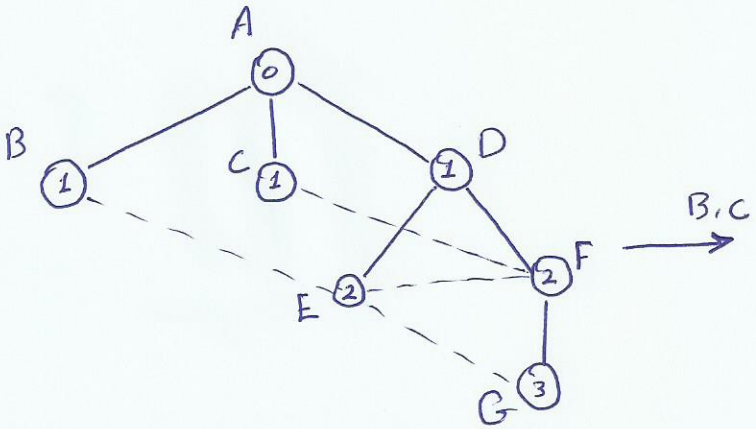
G



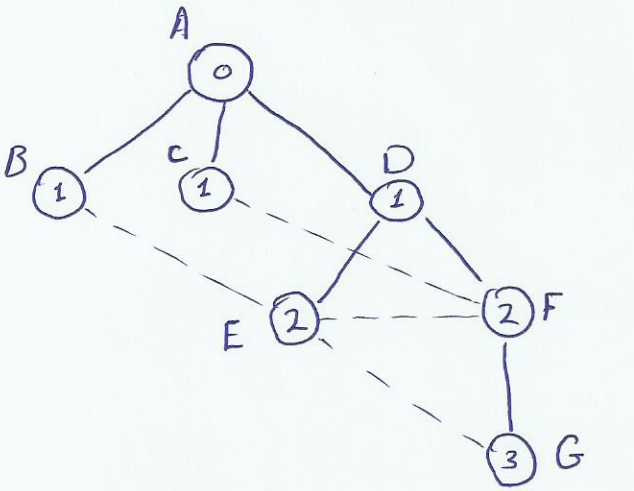
S: B, C, E

S: B, C, E, G

E



B, C



S: B, C

S:

Result

Problem 3:

1) */////

////

1 delete the chosen node

2 replace it with the most right node in the lowest layer.

3 loop while the node < its parent AND node > its childs

a- if(node < its parent) then sift up().

b - if (node > its childs) then siftdown().

Remove () is $O(\log n)$

2)

1 update the value of the chosen node.

2 loop while the node < its parent AND node > its child.

a - if (node < its parent) then ~~siftdown()~~ siftup().

b - if (node > its childs) then siftdown().

Update () is $O(\log n)$

Problem 4:

To represt the graph, I will use LinkedList of LinkedList of node. The outer linkedlist represent the node. The inner LinkedList represent the relation between the nodes.

* LinkedList<LinkedList<node>> graph;
* add a node:
    1. check if this node is already exist
        then do nothing.
    2. else: a. graph.insert(new LinkedList<>());
        b. graph.retrieve().insert(node);

add is O(n)

* Remove a node:
    1. Remove edge between this node and other nodes
    2. Remove the node from the list.

RemoveNode is $O(n^2)$

* addEdge(Node n_1, Node n_2)
    1. loop through the List
        a. if (n_1 is found)
            then graph.retrieve().insert(n_2)
        b. if (n_2 is found)
            then graph.retrieve().insert(n_1)

addEdge is O(n)

* RemoveEdge (Node n_1, Node n_2)

1. Loop through the List

a. if (n_1 is found)

then remove n_2 from inner List.

b. if (n_2 is found)

then remove n_1 from inner List.

RemoveEdge is $O(n^2)$

* findDegree (Node n):

1. Loop through the List.

if (n is found)

then Count number of element in the inner List and return it

findDegree is $O(n^2)$

* find All neighbours (Node n)

1. loop through the list

if (n is found)

then return all the elements in the inner List

find All neighours is $O(n^2)$