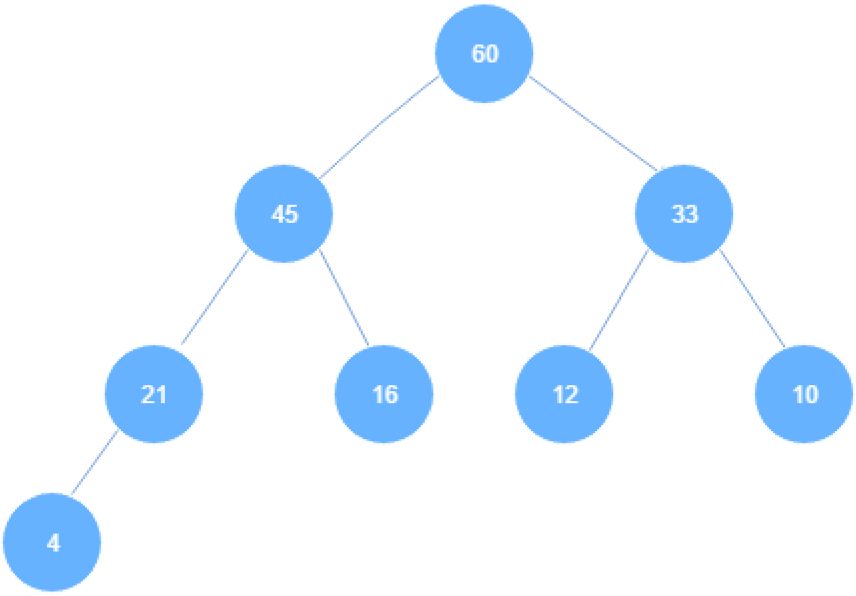
HOMEWORK 6

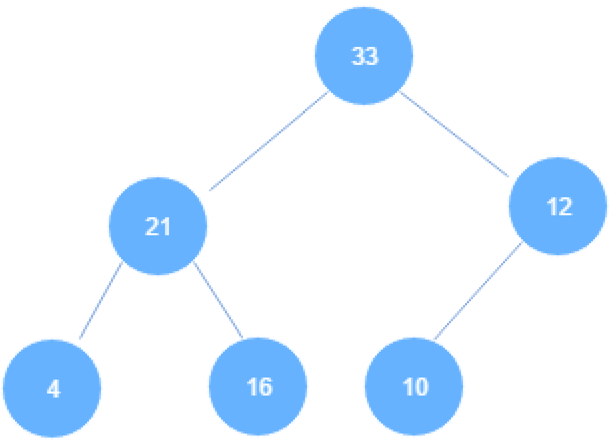
RAHAF ALOMAR – 435201926

PROBLEM1:

1.1:



1.2:



1.3:

A) WE WILL USE [MAX HEAP]

B)

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

C)

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

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

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

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

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

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

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

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

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

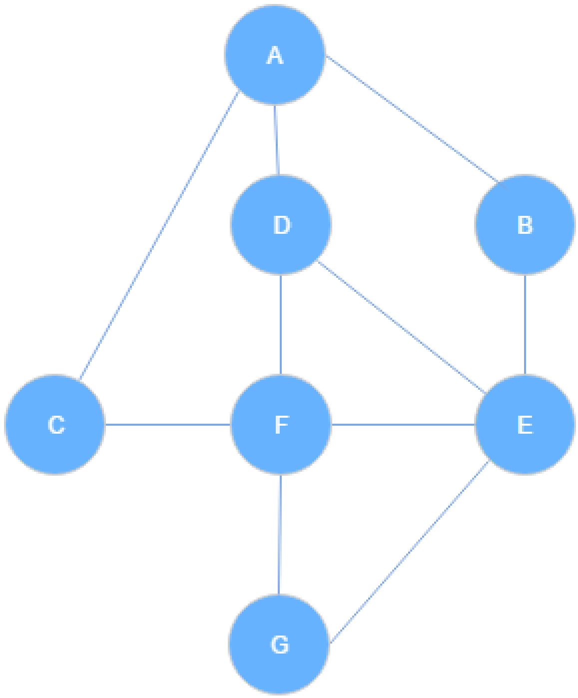
0	1	2	3	4	5	6
-	12	12	18	20	23	28

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

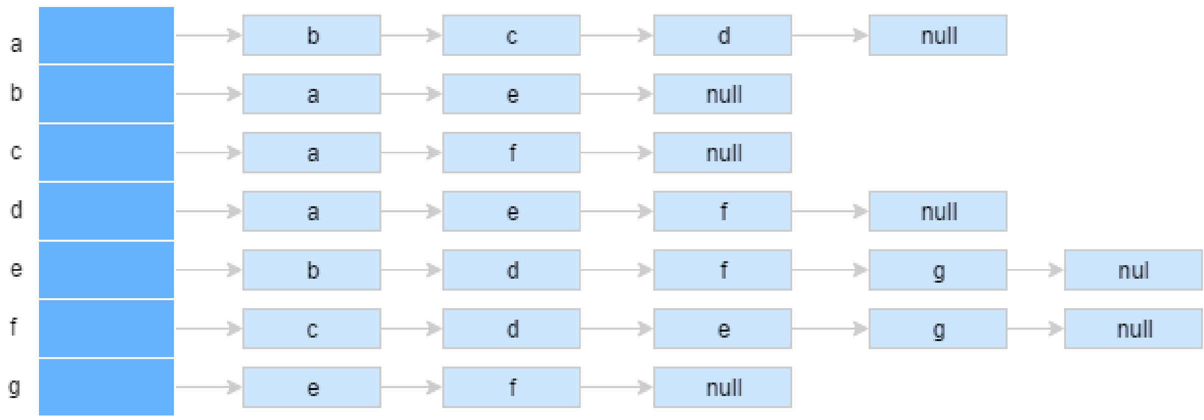
0	1	2	3	4	5	6
-	12	15	18	20	23	28

PROBLEM 2:

2.1:



2.2:



2.3:

BFS: A B C D E F G

DFS: A D F G E C B

PROBLEM 3:

3.1:

SINCE THE IMPLEMENTAION OF HEAP IS ARRAY, SO WE CAN GO TO ANY INDEX WITH O(1).

THEN TO REMOVE THE KEY WE NEED TO REPLACE IT WITH THE LAST ELEMENT THEN CALL (SIFTDOWN) METHOD, WHICH NEEDS O(LOG N), SO WE WILL NEED **O(LOG N)** TIME TO REMOVE A KEY.

3.2:

SINCE THE IMPLEMENTAION OF HEAP IS ARRAY, SO WE CAN GO TO ANY INDEX WITH O(1).

THEN UPDATING THE KEY NEEDS TO CALL (SIFTUP) METHOD, WHICH NEEDS O(LOG N), SO WE WILL NEED **O(LOG N)** TIME TO UPDATE A KEY.

PROBLEM 4:

The adjacency list structure for a graph adds extra information to the edge list structure

that supports direct access to the incident edges (and thus to the adjacent vertices)

of each vertex. Specifically, for each vertex v, we maintain a collection I(v),

called the incidence collection of v, whose entries are edges incident to v. In the

case of a directed graph, outgoing and incoming edges can be respectively stored in

two separate collections, Iout(v) and Iin(v). Traditionally, the incidence collection

I(v) for a vertex v is a list, which is why we call this way of representing a graph

the adjacency list structure.

We require that the primary structure for an adjacency list maintain the collection

V of vertices in a way so that we can locate the secondary structure I(v)

for a given vertex v in O(1) time. This could be done by using a positional list

to represent V, with each Vertex instance maintaining a direct reference to its I(v)

incidence collection; we illustrate such an adjacency list structure of a graph in Figure

14.5. If vertices can be uniquely numbered from 0 to n−1, we could instead

use a primary array-based structure to access the appropriate secondary lists.

The primary benefit of an adjacency list is that the collection I(v) (or more

specifically, Iout(v)) contains exactly those edges that should be reported by the

method outgoingEdges(v). Therefore, we can implement this method by iterating

the edges of I(v) in O(deg(v)) time, where deg(v) is the degree of vertex v. This

is the best possible outcome for any graph representation, because there are deg(v)

edges to be reported.

THE PERFORMANCE OF EACH OF THE OPERATIONS:

OPERATION:	RUNNING TIME:
ADD A NODE.	**O(1)**
REMOVE A NODE	**O(deg(v))**
ADD AN EDGE	**O(1)**
REMOVE AN EDGE	**O(1)**
FIND THE DEGREE OF A NODE	**O(1)**
FIND ALL NEIGHBOURS OF A NODE.	**O(1)**

REFERENCE:

GOODRICH, TAMASSIA, GOLDWASSER. *Data Structures and Algorithms in Java*, 6th Edition, 2014