Alara Zeybek
22102544

a) INSERTION

1) 40

2) 40
\
50

3) 45
/ \
40 50

4) 45
/ \
40 50
/
30

5) 45
/ \
40 50
/
30

6) 45
/ \
40 55
/ / \
30 50 60

7) 45
/ \
30 55
/ \ / \
20 40 50 60

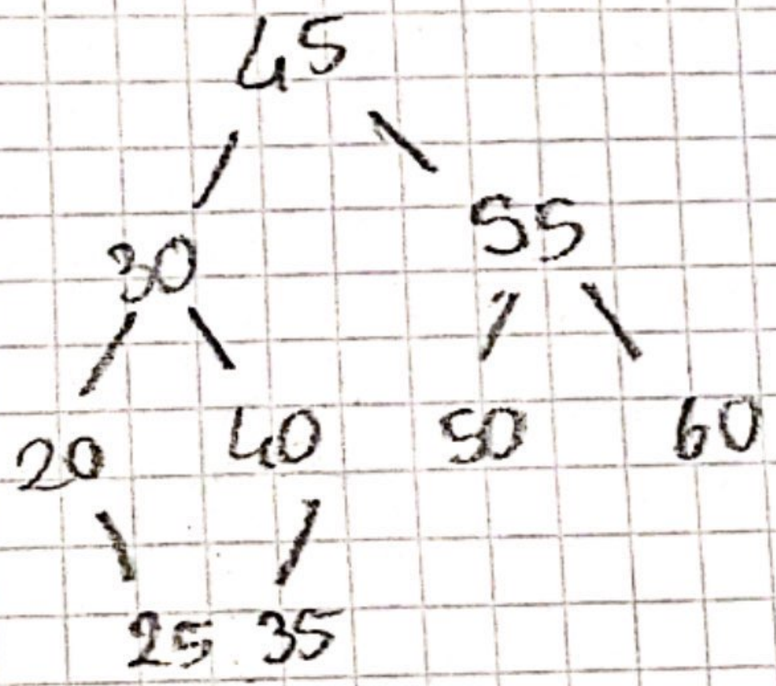
8) 45
/ \
30 55
/ \ / \
20 40 50 60
/
35

9) 45
/ \
30 55
/ \ / \
20 40 50 60
/ /
10 35

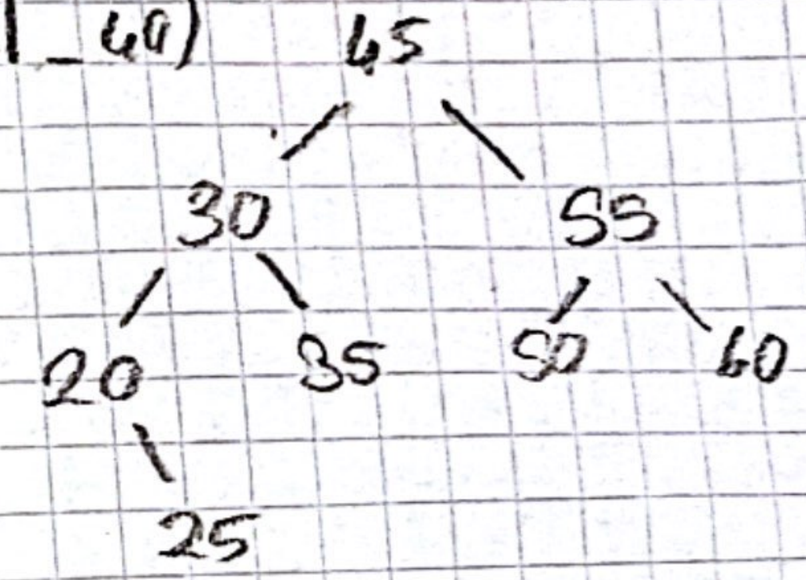
10) 45
/ \
30 55
/ \ / \
20 40 50 60
/ \ /
10 25 35

Deletion

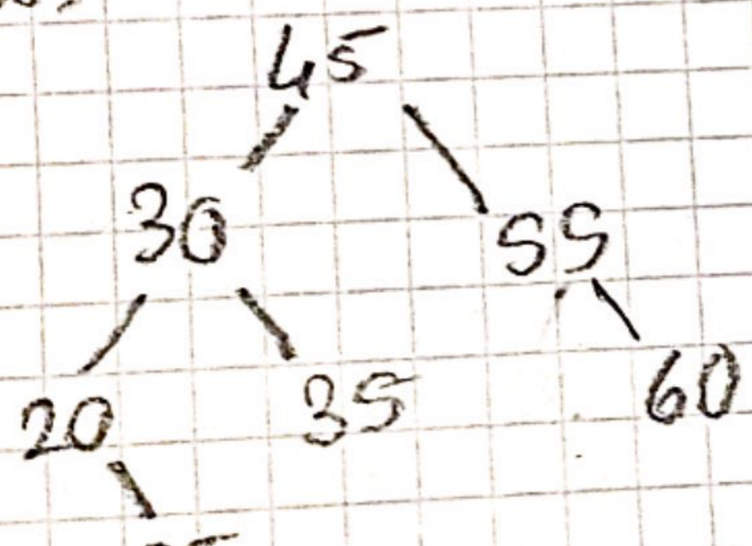
Del_10)



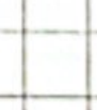
Del_40)



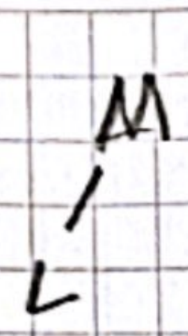
Del_50)



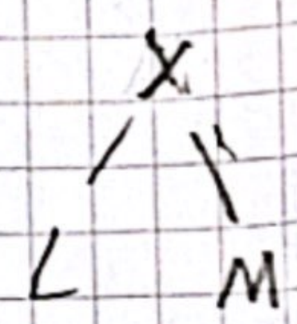
1)



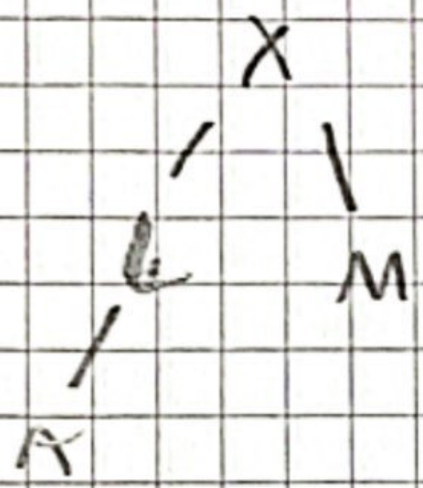
2)



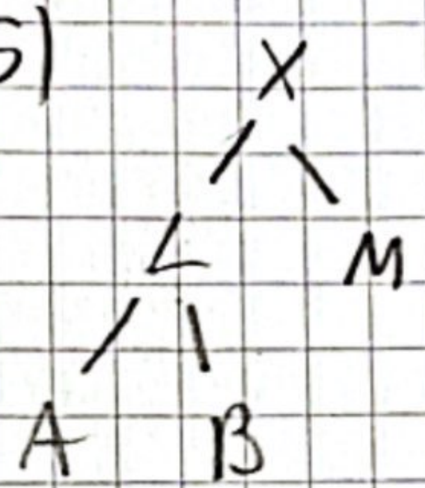
3)



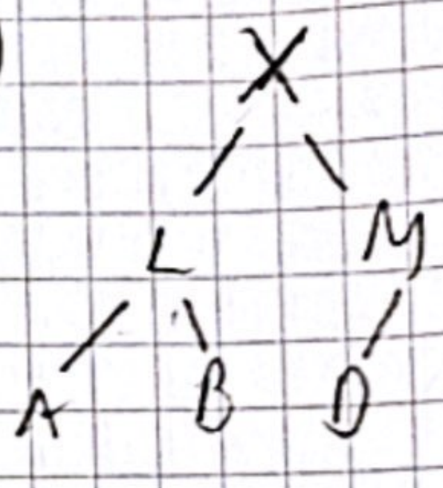
4)



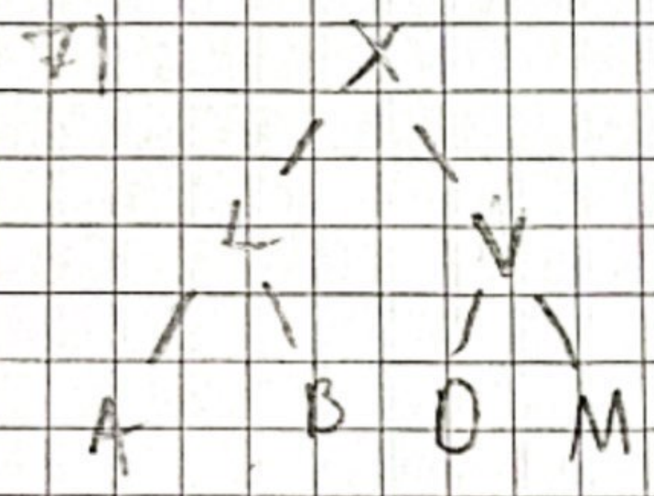
5)



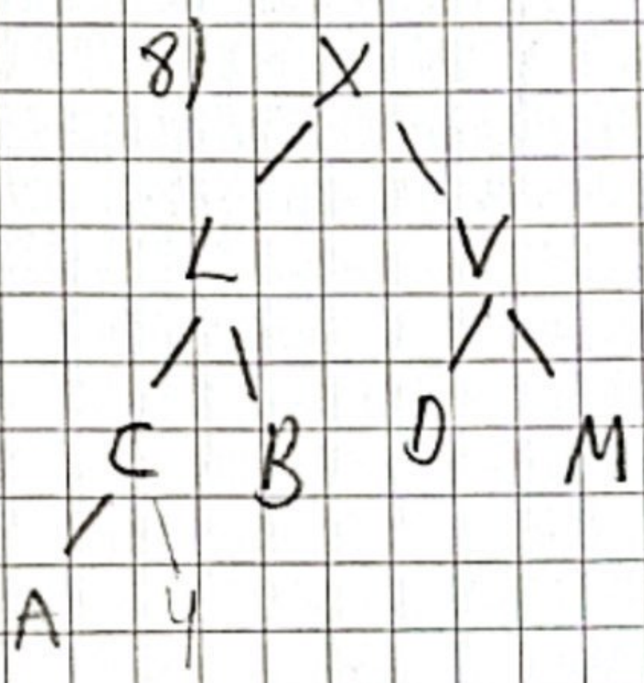
6)



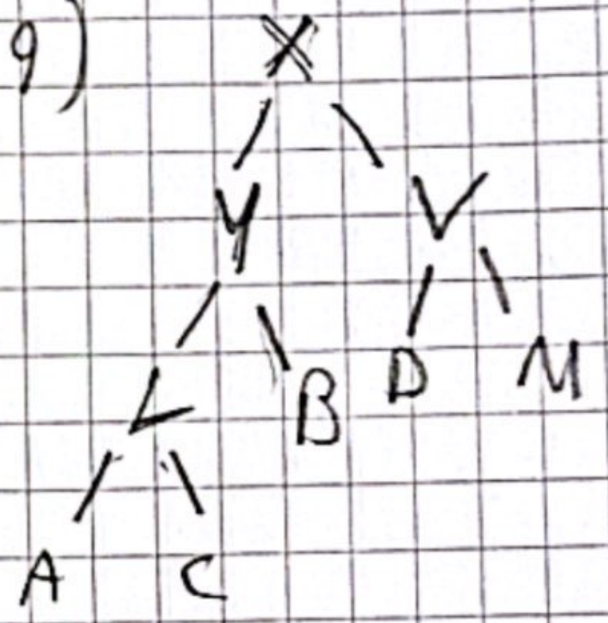
7)



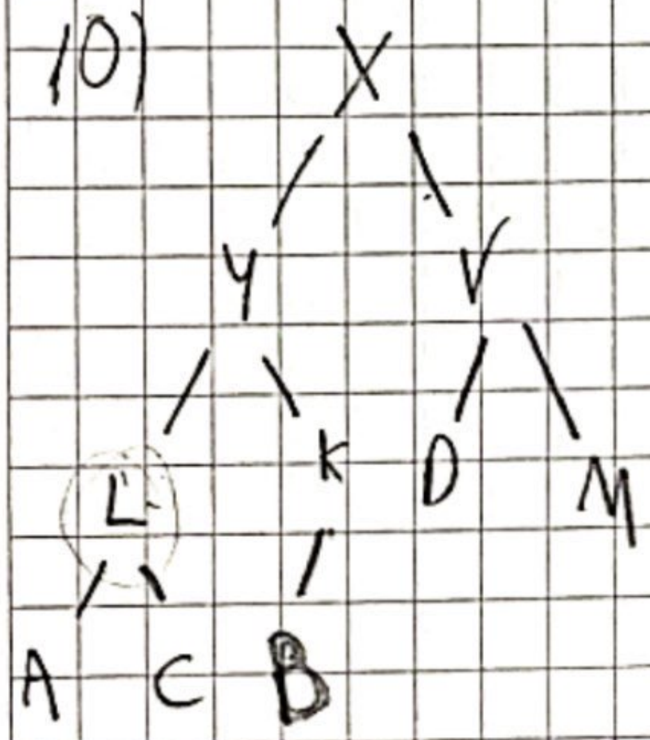
8)



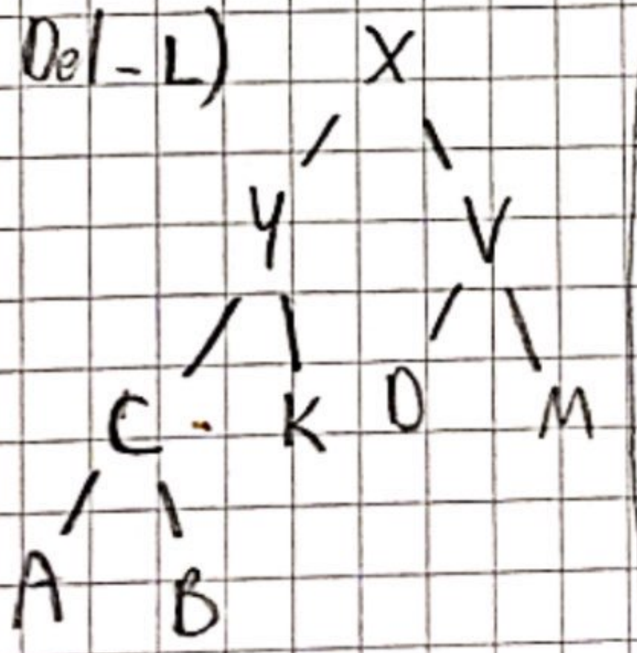
9)



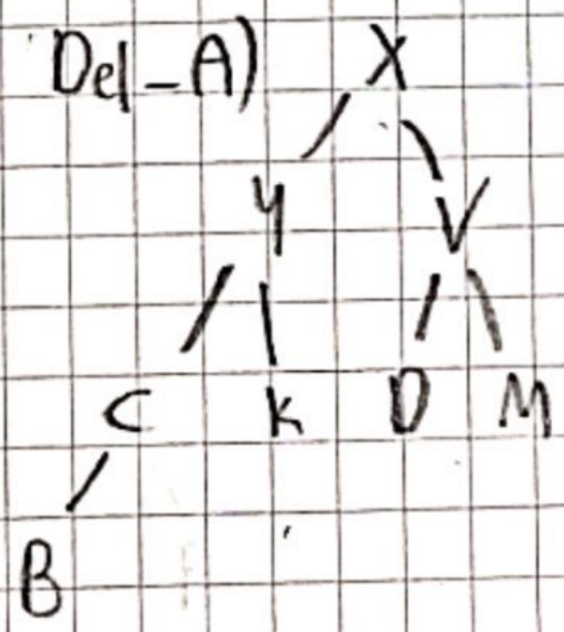
10)



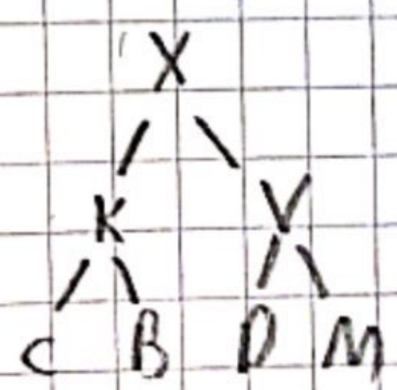
Del_L)



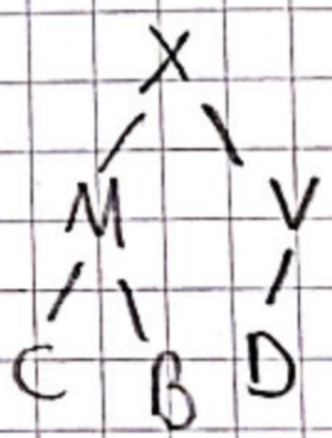
Del_A)



Del_Y)



Del_k)



c) a) Finding max element: Does not matter because in sorted array max is arr[size-1] and it takes O(1). In max-heap, max element is the root. O(1) time, it takes again.

b) finding min element = sorted array; In max-heap, we only know that the child element is less than the parent. So we need to traverse the heap in order to find the min. However, in the sorted array, there is an ordering and that's why it is more efficient.

c) finding the median

If we want to find the median by considering its location (I.e. the median of {1,1,1,2,3} is 1) Sorted array is more efficient because it will take O(1) since we are looking for the element in the arr[n/2] unrelated with the duplicateexistence

d) deleting an element: If want-to-delete element is the max-element. Heap takes O(log n) time, array takes O(n) time. However, if the want-to-delete element is just a random element both would take O(n) time. Overall by considering the best case(deleting the max element), I would choose heap.

e) forming the structure quickly: Heap rebuild takes O(log n) time. So creating a structure means inserting a new item for n times. Here we can compare the insertion complexity. Insertion in heap takes O(log n) times, insertion in sorted array takes O(n) times. Overall, I would choose heap.