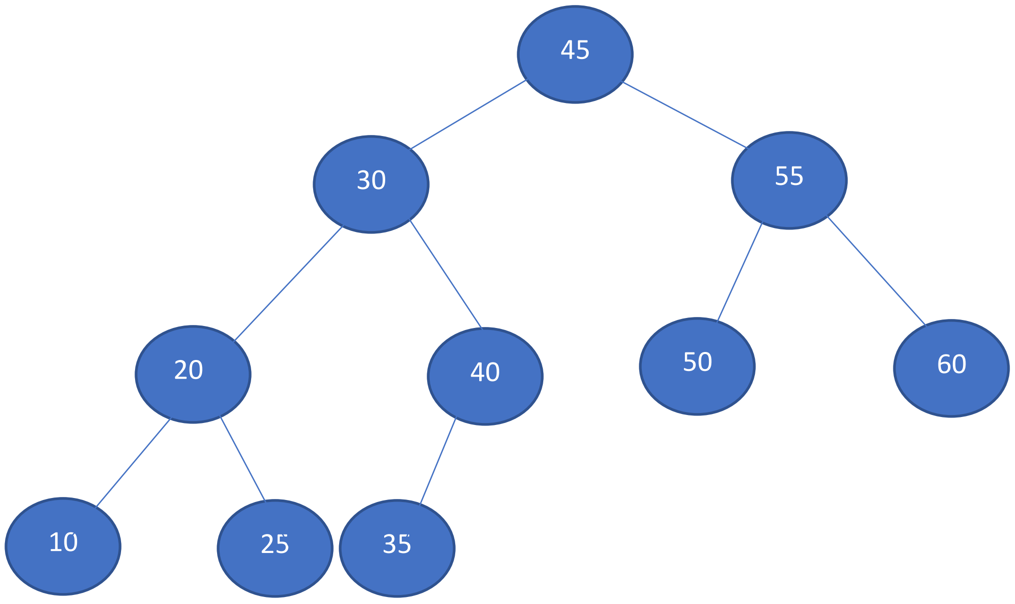
/**
* Title : Heaps and AVL Trees
* Author : Faaiz Ul Haque
* ID: 21503527
* Section : 2
* Assignment : 3
* Description : Report for question 1 & 3
*/

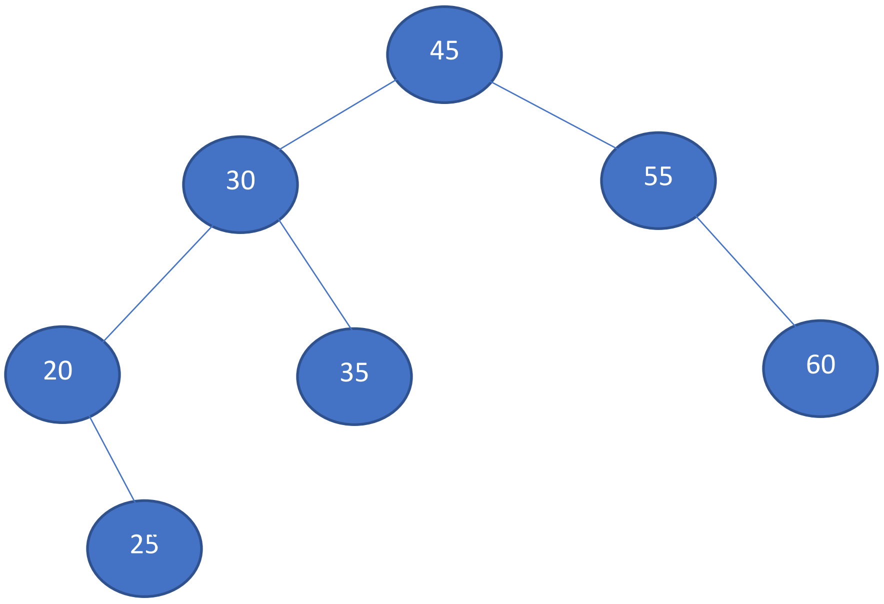
Question 1 – 20 points

(a) [5 points] Insert 40, 50, 45, 30, 60, 55, 20, 35, 10, 25 to an empty AVL tree. Show only the final tree after all insertions. Then, delete 10, 40, 50 in given order. Show only the final tree after all deletion operations. Use the exact algorithms shown in the lectures

After insertions:



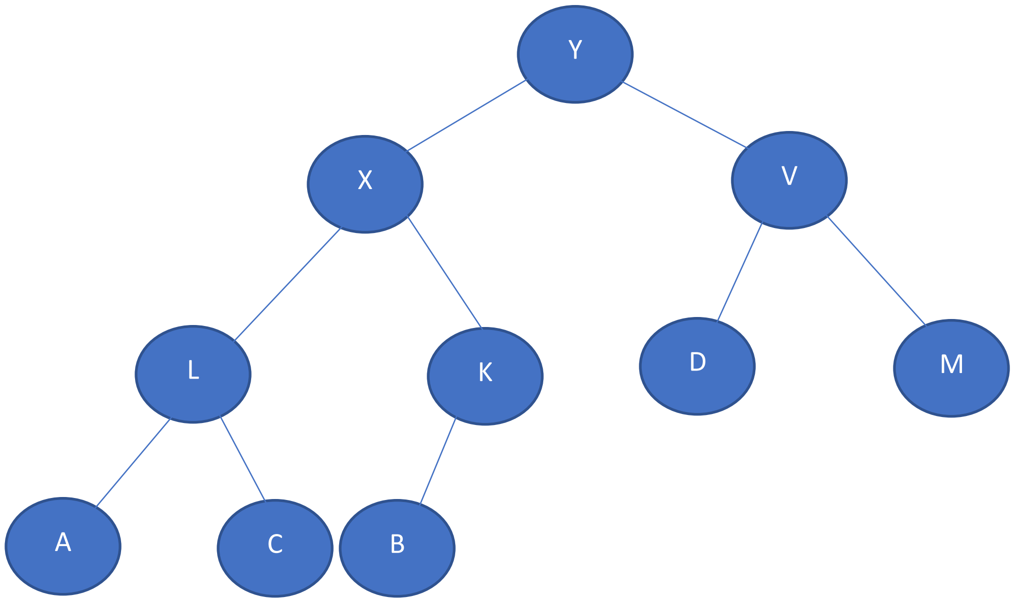
After deletions:



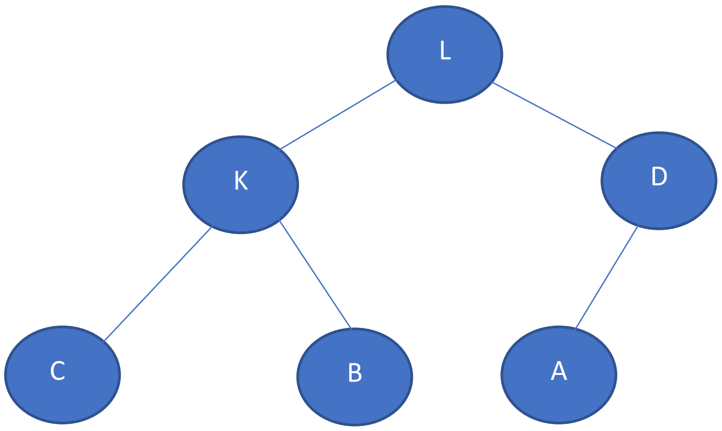
```
/**
* Title : Heaps and AVL Trees
* Author : Faaiz Ul Haque
* ID: 21503527
* Section : 2
* Assignment : 3
* Description : Report for question 1 & 3
*/
```

(b) [5 points] Insert M, L, X, A, B, D, V, C, Y, K, into an empty max-heap. Show only the final heap as a tree (not as an array) after all insertions. Then, delete L, A, Y, K in given order. Show only the final heap as a tree after all deletion operations. Use the exact algorithms shown in the lectures.

After insertions:



After deletions:



```
/**
* Title : Heaps and AVL Trees
* Author : Faaiz Ul Haque
* ID: 21503527
* Section : 2
* Assignment : 3
* Description : Report for question 1 & 3
*/
```

(c) [10 points] You would like to store a set of numbers either in a max-heap or a sorted array. For the following applications, explain which data structure is better, or discuss if it does not matter which one is used. Answers without explanation will get 0 points.

(a) finding maximum element quickly

Does not matter. In a sorted array the max element is always in first location or last location depending on if its ascending or descending, and in a max-heap the max element is always in the first location so we can directly find the max element in O(1) time in both cases.

(b) finding minimum element quickly

Sorted array. In a sorted array finding the minimum element can be done in O(1) time as the minimum is either in the start or end of the array depending on if its ascending or descending. In a max-heap the minimum element does not have to be in the last element, it is in the last level of the tree but this could take a while to find if the heap size is very large.

(c) finding median of elements quickly

Sorted array. We can directly find the median by dividing the size of the array by 2 and selecting the median as the middle or an average of the two middle elements. We do not have to go through the entire array, and the operation can be done in O(1) time. In a heap we have to again check the middle level or the two middle levels of the heap and if the heap is again large this may take a long time, greater than O(1)

(d) deleting an element quickly

Max-Heap. Deleting an element from a max-heap takes O(logN) time at average and worse case, since when we delete we swap the element we want to delete with the last element in the array which is O(1) operation. Then we restructure the heap, and we only have to swap elements in one level so at worst this is still logN. However in a sorted array, at worst case we have to delete the first element and in this case we should move all the other items which is an O(N) time operation.

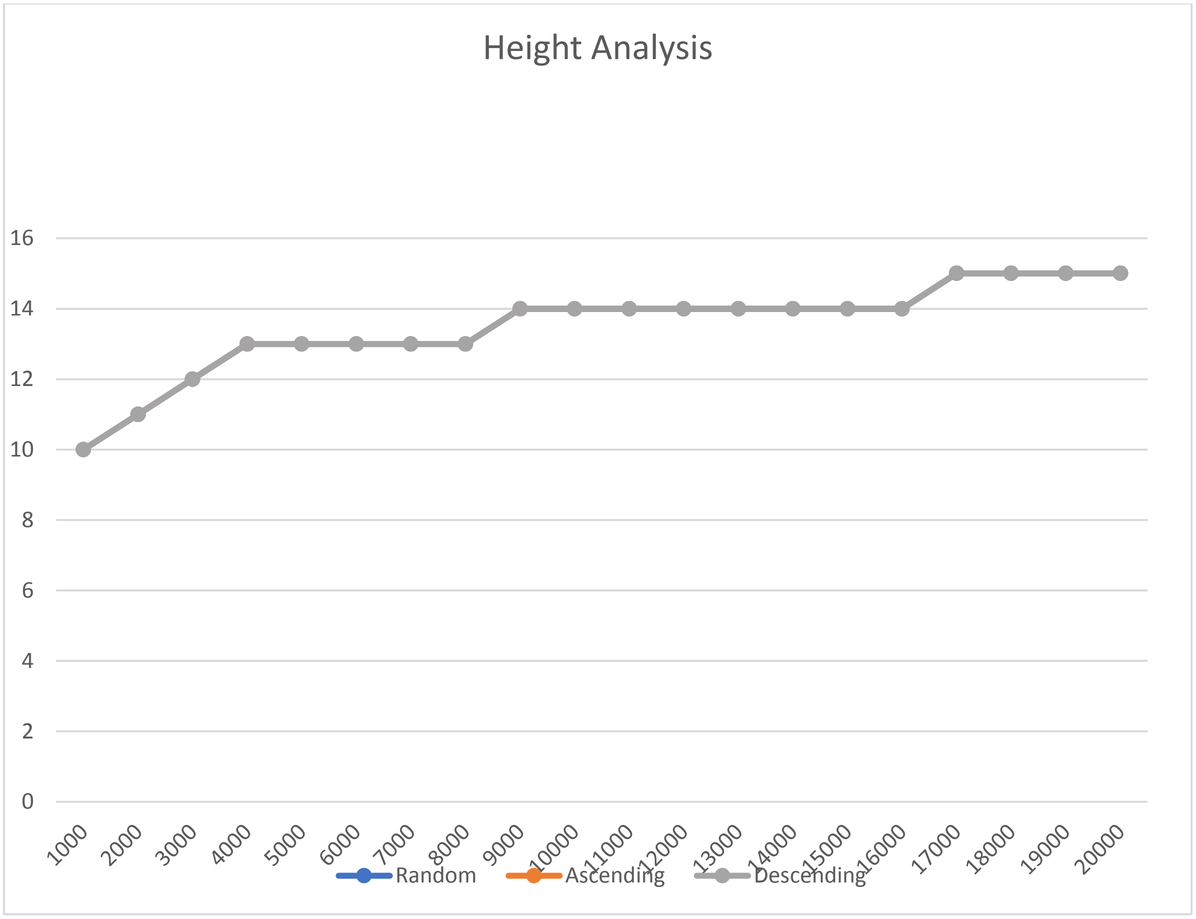
/**
* Title : Heaps and AVL Trees
* Author : Faaiz Ul Haque
* ID: 21503527
* Section : 2
* Assignment : 3
* Description : Report for question 1 & 3
*/
(e) forming the structure quickly

Max-heap. Forming a heap is a linear operation where-as forming a sorted array is in its best worst case nlogn.

Question 1 – 20 points

Array Size	Random	Ascending	Descending
1000	10	10	10
2000	11	11	11
3000	12	12	12
4000	13	13	13
5000	13	13	13
6000	13	13	13
7000	13	13	13
8000	13	13	13
9000	14	14	14
10000	14	14	14
11000	14	14	14
12000	14	14	14
13000	14	14	14
14000	14	14	14
15000	14	14	14
16000	14	14	14
17000	15	15	15
18000	15	15	15
19000	15	15	15
20000	15	15	15

```
/**
* Title : Heaps and AVL Trees
* Author : Faaiz Ul Haque
* ID: 21503527
* Section : 2
* Assignment : 3
* Description : Report for question 1 & 3
*/
```



Do your findings related to average height of AVL tree agree with the theoretical results?

Average height of AVL Tree is O(logN) this does correspond to the results obtained above for all cases of arrays.

Do the different patterns of insertion affect the height of AVL tree? If so, explain how. If not, explain why not.

No they do not, since we keep rotating upon receiving an imbalance. So it doesn't matter how we insert we try to occupy the most space as soon as a violation of 1 level different occurs. This makes AVL tree's operations very efficient and it is why it is preferred. The random ascending or descending do not have any effect on the height of the AVL Tree. They will all obtain the same heights in the end.

How would the result be if you used regular Binary Search Tree instead of AVL tree?

```
/**
* Title : Heaps and AVL Trees
* Author : Faaiz Ul Haque
* ID: 21503527
* Section : 2
* Assignment : 3
* Description : Report for question 1 & 3
*/
```

Ascending and descending would have the largest heights since it would show up in the worst case. Their heights would be O(N) while the random generated array would have a height of O(logN) which is the average case. But still we are interested in the worst cases which is why AVL Trees are preferred.

We can clearly see from the graph only one curve in one colour this is because the curves are exactly identical so unfortunately in this graphing tool only one is visible but the 2 other curves are directly underneath that one.