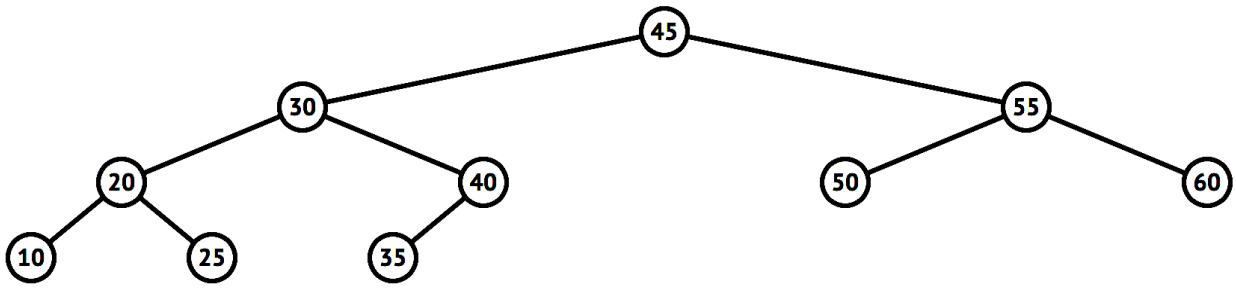
CS 202

Homework 3

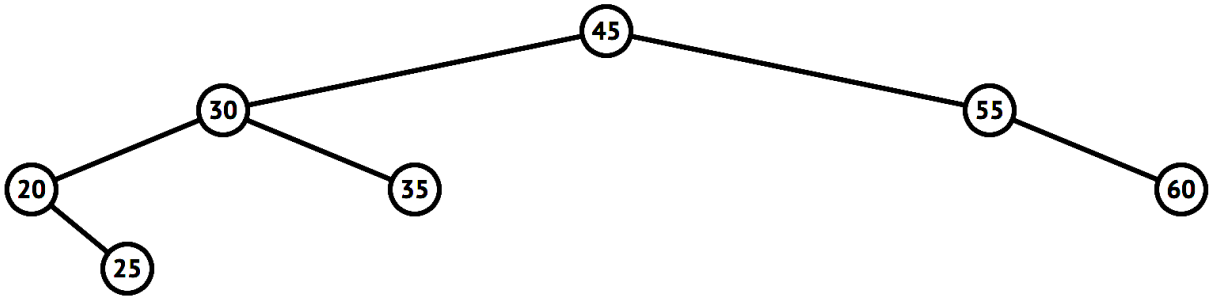
Kerem Ayöz
21501569 Section: 1

Question 1

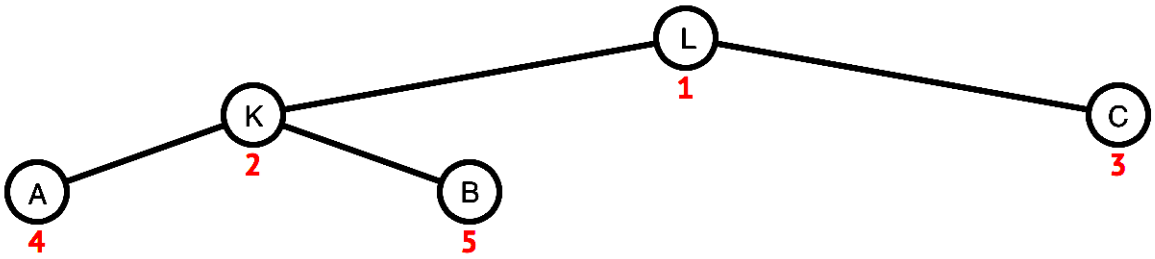
a) After the insertions of 40, 50, 45, 30, 60, 55, 20, 35, 10, 25 into the AVL Tree



After the deletions of 10, 40, 50 from the AVL Tree



b) After insertions of *M, L, X, A, B, D, V, C, Y, K* to max-heap and deletions of *Y, X, V, M* from the max-heap



c) Decisions of which data structure is the most suitable in the different cases:

1. To find the maximum element quickly, it does not matter which of these we use, both data structures have same time complexity of *O(1)* to find maximum element.

2. To find the minimum element quickly, we need to use sorted array in order to make it efficient. We just need to access the first or last element of the array and it takes *O(1)* time. In the max-heap this takes *O(n)* time.

3. To find the median of the elements, sorted array is better if we know its size this operation takes O(1) time, if we do not know the size it does not matter which data structure we used.

4. Deleting the any given element takes *O(n)* time in both data structures so it does not matter which of them we use.

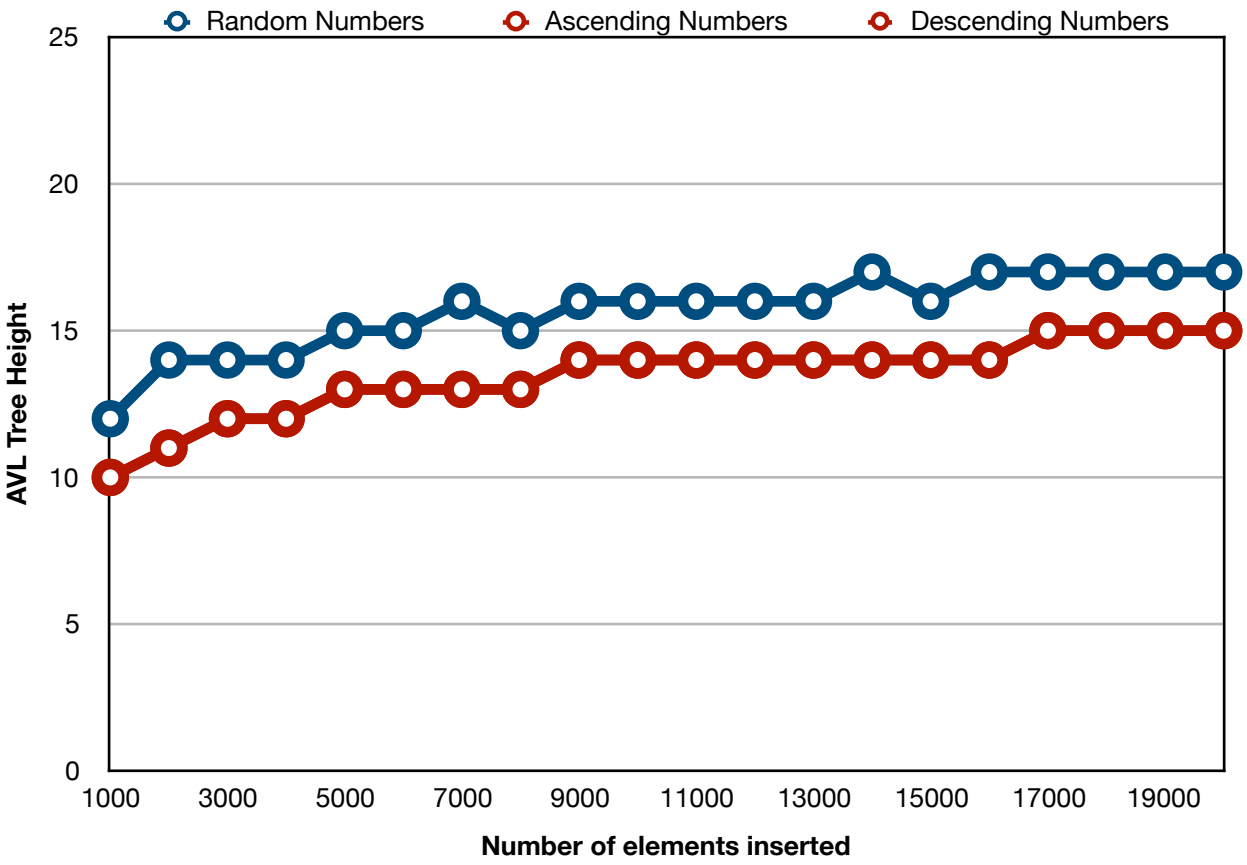
5. We can form the structure by using a max-heap in *O(nlogn)* time because insertion takes O(logn) time in max-heap also forming the structure with array that is sorted depends on the algorithm that we use but in the best case it takes *O(nlogn)* time. So it does not matter which one we use in terms of efficiency.

Question 3

AVL Trees are the balanced binary search trees so that it keeps its balance by doing rotations while the new element is inserting. Therefore it avoids the worst-case scenarios in the binary search trees such that insertion of ascending or descending integers to binary search trees give different results than the random integer insertion in terms of height, which means efficiency in binary trees.

Different patterns did not affect the height of the tree as like binary search trees which are not balanced. AVL Trees guarantees even in the insertion stage that height difference between subtrees are no more than 1. This makes elements are inserted nearly perfect. And also whatever the case is nearly the same results are occurred in the graph.

Figure 1: Height of the AVL Tree after different type of insertions with different number of elements.



If we have used Binary Search tree in that experiment, difference of the height growths will be much more huge between the cases. Random insertion to the both AVL Trees and Binary Search Trees may be closer to each other, however the AVL Tree is much more guaranteed because of the obscurity of the randomness. Ascending and descending integer insertions will make the Binary Search Tree linear like linked list, height gets as larger as the number of elements, but we obtained the result that height did not change that much while inserting the huge number of integers to AVL Tree.

As a consequence, using balanced binary search trees guarantee less height than binary search trees whatever the order of the numbers that are inserted. If our main aim is to build an efficient binary tree we should use balanced binary search trees such as AVL Trees which forms the structure more like full binary tree. Getting closer to full binary tree and storing the elements as binary search tree structure makes all the operations to execute in efficient times and AVL Tree has that features.