Question 1

A)

9 and 12 inserted normal. Tree is still balanced.

After element 10 inserted, tree became unbalanced at element 9 which is current root, double right-left rotation happened at unbalanced element(case 3 in our lecture slide avltree, slide number 13).

5 inserted normal. Tree is still balanced.

After element 1 inserted, tree became unbalanced at element 9 which is the left child of current root, single right rotation happened at unbalanced element(case 1 in our lecture slide avltree, slide number 11).

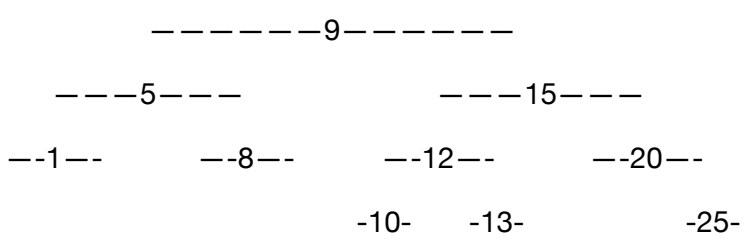
After element 8 inserted, tree became unbalanced at element 10 which is current root, double left-right rotation happened at unbalanced element(case 4 in our lecture slide avltree, slide number 14).

After element 20 inserted, tree became unbalanced at element 10 which is the right child of current root, single left rotation happened at unbalanced element(case 2 in our lecture slide avltree, slide number 12).

15 inserted normal. Tree is still balanced.

After element 13 inserted, tree became unbalanced at element 20 which is right child of right child of current root, single right rotation happened at unbalanced element(case 1 in our lecture slide avltree, slide number 11).

After element 25 inserted, tree became unbalanced at element 12 which is right child of current root, single left rotation happened at unbalanced element(case 2 in our lecture slide avltree, slide number 12).



B)

arr = {41,75,63,33,49} is my initial array. Size = n = 5 note my second parameter calculated as n/2 -1

After heapRebuild(arr, 1, n); arr = {41,75,63,33,49}

After heapRebuild(arr,0, n); arr = {75,49,63,33,41}

So my heap is builded from my array now. So lets swap first and last element of unsorted. “|” after this sign right side represents sorted part.

Swap arr[0] and arr[n - 1] arr = {41,49,63,33,|71}

Now we should rebuild this part of array{41,49,63,33}

After heapRebuild(arr, 1, n-1) arr = {41,49,63,33,|71}

After heapRebuild(arr, 0, n-1) arr = {63,49,41,33,|71}

Again my heap is builded so lets swap first and last element of unsorted.

Swap arr[0] and arr[n-2] arr ={33,49,41,|63,71}

Now we should rebuild this part of array {33,49,41}

After heapRebuild(arr, 0, n-2) arr = {49,33,41,|63,71}

Again my heap is builded so lets swap first and last element of unsorted.

Swap arr[0] and arr[n-3] arr ={33,41,|49,63,71}

Now we should rebuild this part of array {33,41}

After heapRebuild(arr, 0, n-3) arr = {41,33,|49,63,71}

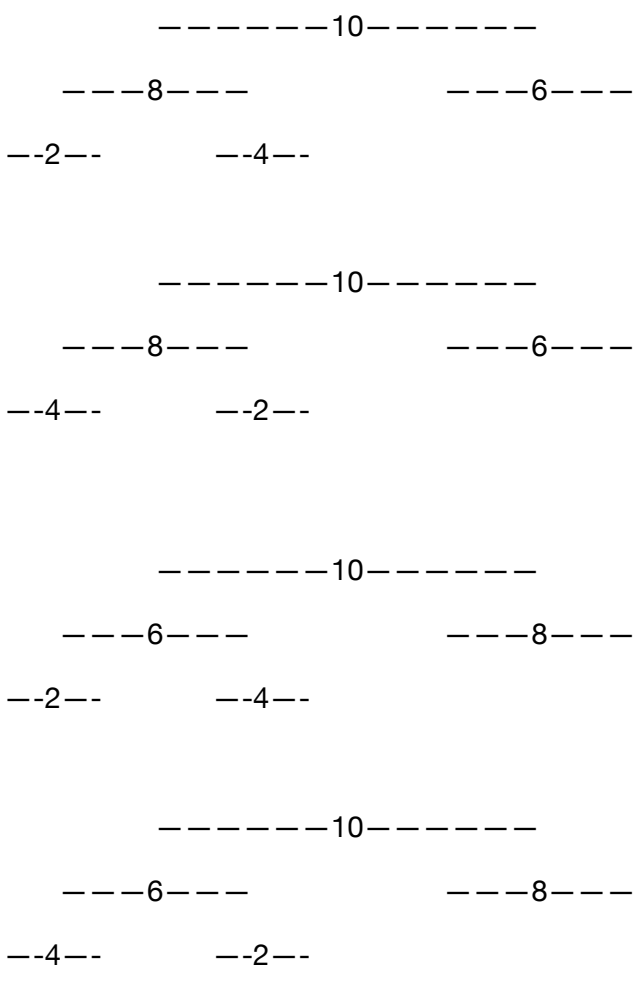
Again my heap is builded so lets swap first and last element of unsorted.

Swap arr[0] and arr[n-4] arr ={33,|41,49,63,71}

N = 1 so its already sorted

Arr = {|33,41,49,63,71} array is sorted

C)



Question 2

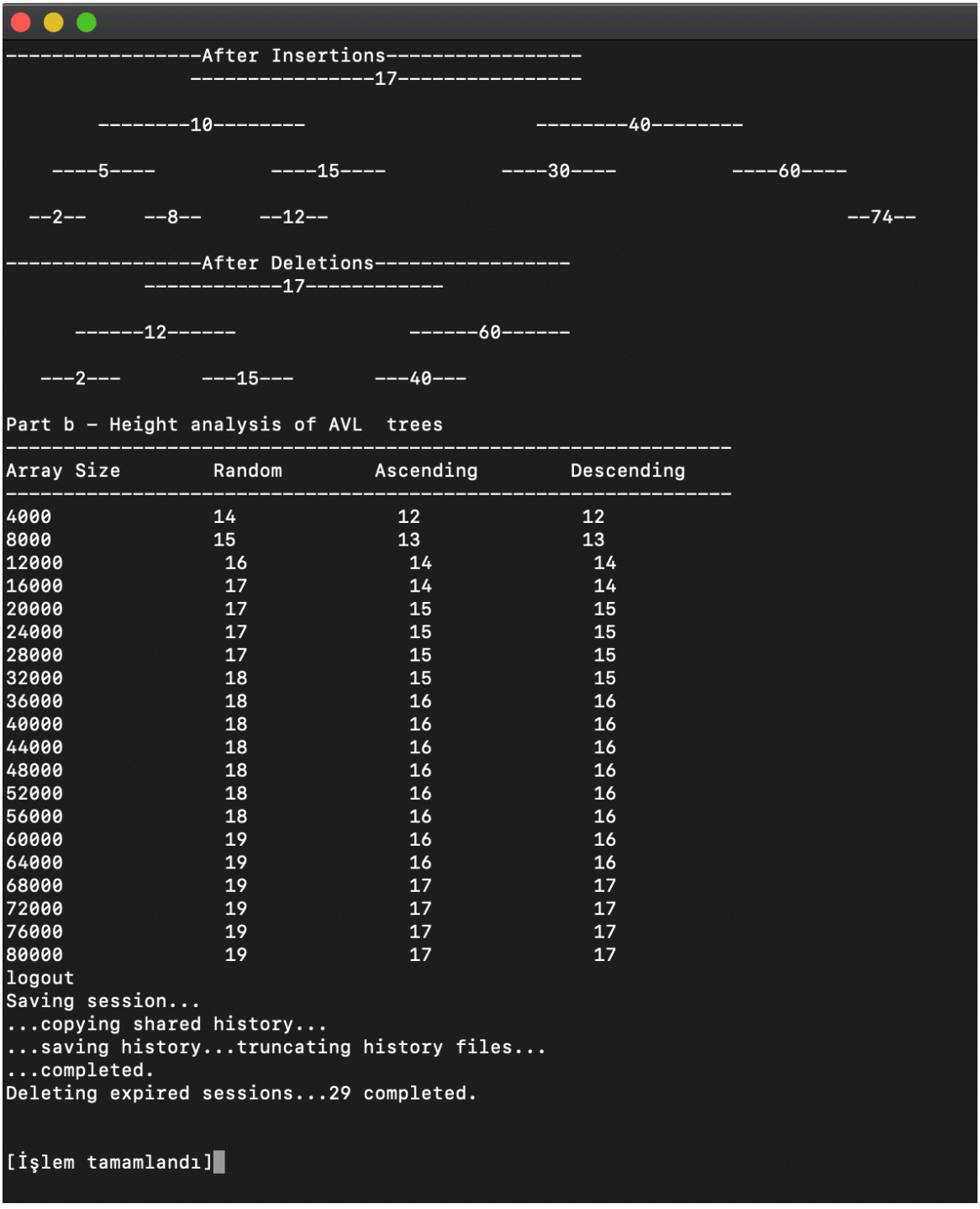


Figure 1.1

Question 3

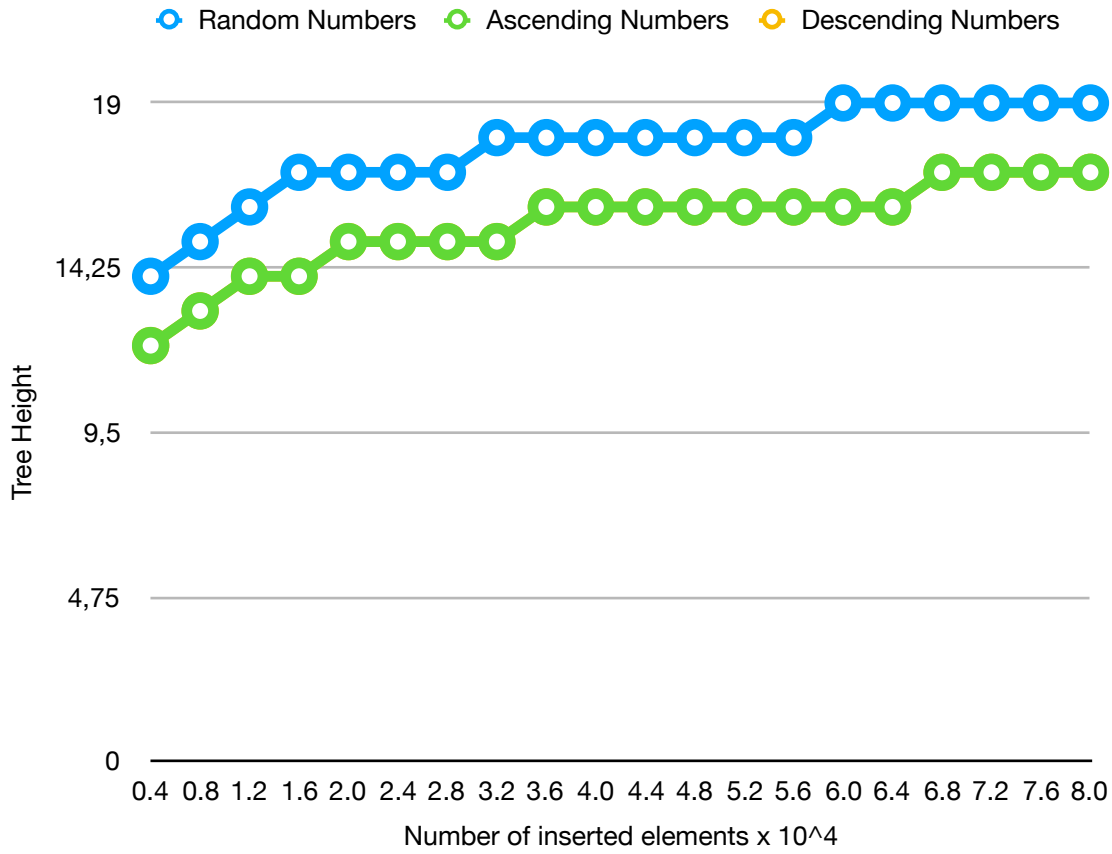


Figure 1.2

1) At figure 1.2 Descending Numbers are can not be seen because ascending and descending heights are always same when we investigate figure 1.1. Minimum number of nodes in AVL tree is minN(h) = min(h - 1) + min(h - 2) + 1 which h is height of the tree(AVLTree slide, slide number 28). So Worst case height of an AVLTree with n nodes is O(1.44 logn) which is O(logn)(AVLTree slide, slide number 28). Our results are correct when we investigate the worst case. Ascending and descending insertions gives the same height because while inserting elements, tree makes only single rotation and mirror rotations(For example ascending insertion makes only single left rotation, descending insertion makes only single right rotation). So their height is same. Random insertions gave bigger height because there is a double rotation possibility.

2) On both insertion and deletion operations, rotations and getting balance factor operations takes constant time. So the time complexity of both insertion and deletion operations of AVL tree is same as binary search tree's insertion and deletion operation which is O(h) where h is the height of the tree. AVL tree is always balanced so the height is O(logn). So time complexity of AVL tree insertion and deletion is O(logn) and worst case and best case of both operation always same because its a balanced tree and equal to O(logn).