Question 1

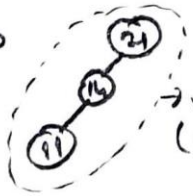
Question 1

Utku Boran Torun

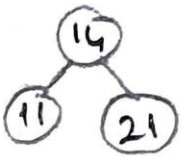
a) Insert 21

=> Insert 14

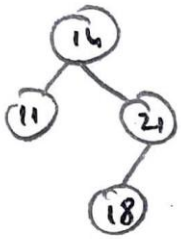
=> Insert 11



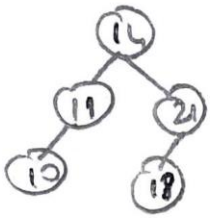
=>



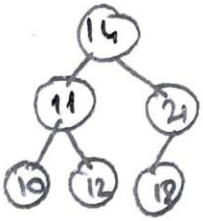
=> Insert 18



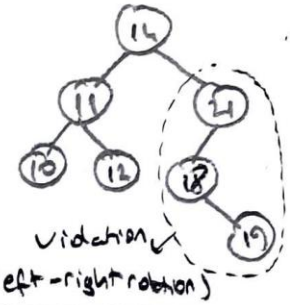
=> Insert 10



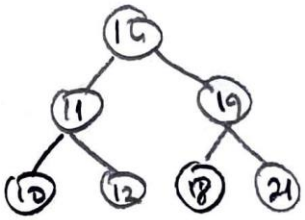
=> Insert 12



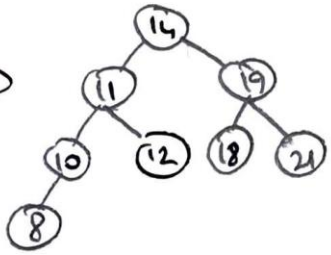
=> Insert 19



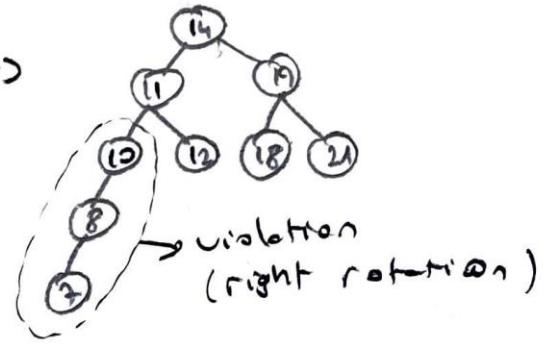
=>



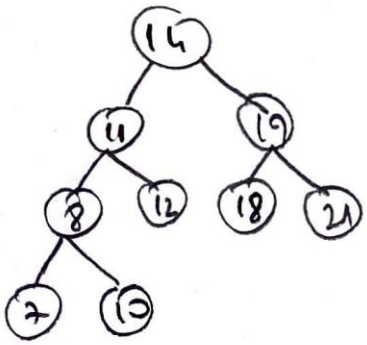
=> Insert 8



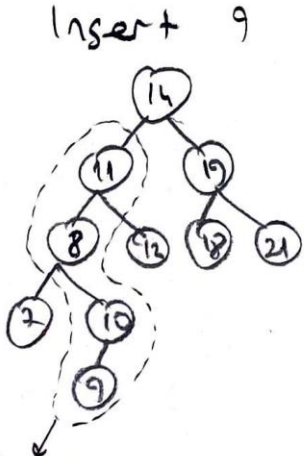
=> Insert 7



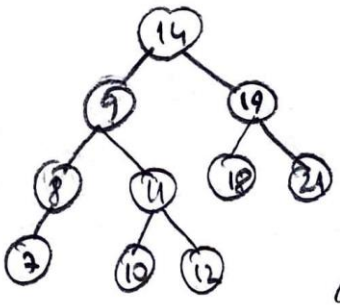
=>

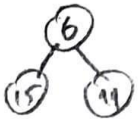
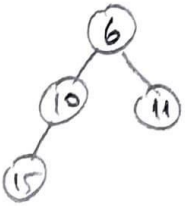


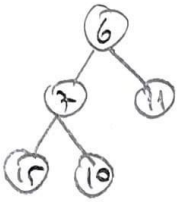
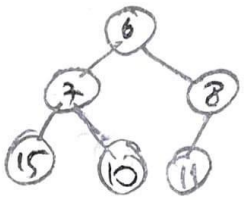
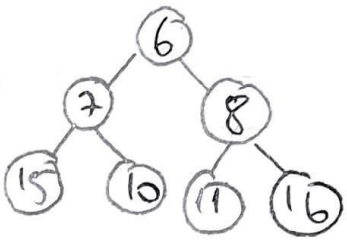
=> Insert 9

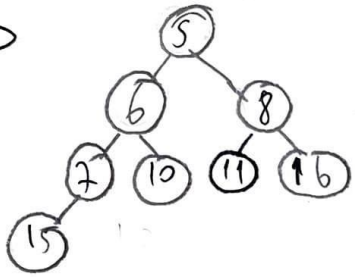
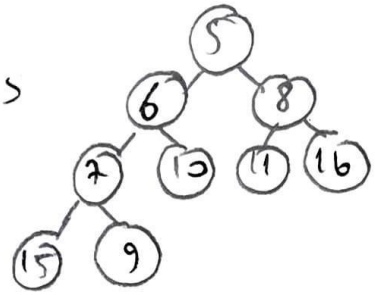
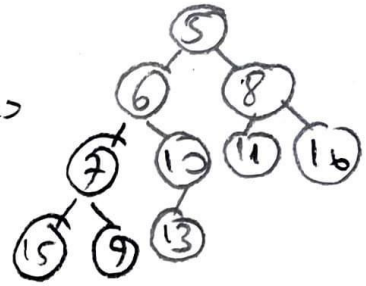


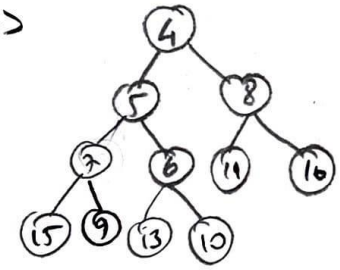
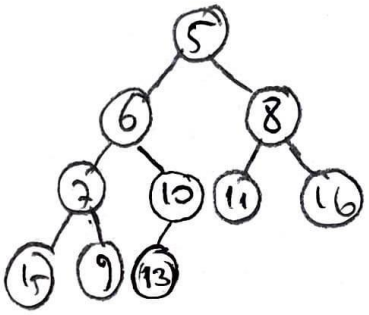
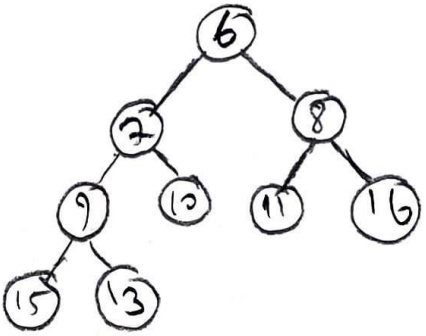
=>



b) Insert 15  => Insert 11  => Insert 6  => Insert 10 

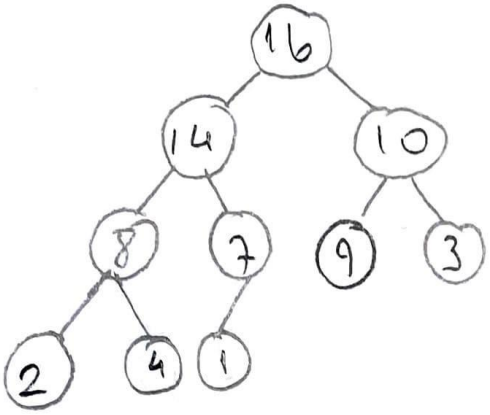
=> Insert 7  => Insert 8  => Insert 16 

=> Insert 5  => Insert 9  => Insert 13 

=> Insert 4  => First deleteMin  => Second deleteMin 

c) Consider heap that occurs Heap Powerpoint slide page 20.

Utku Boran Torun



Preorder Traversal: 16-14-8-2-4-7-1-10-9-3 => Not Sorted

Inorder Traversal: 2-8-4-14-1-7-16-9-10-3 => Not sorted

Postorder Traversal: 2-4-8-1-7-14-9-3-10-16 => Not sorted

---

d) If we write base case and reccurence relation for minimum number of nodes in an AVL tree of height h,

$\{N(1) = 1,\ N(h) = N(h-1) + N(h-2) + 1$

Minimum number of nodes in an AVL tree of height 20,

$N(2) = N(1) + N(0) + 1 = 2$

$N(3) = N(2) + N(1) + 1 = 4$

$N(4) = N(3) + N(2) + 1 = 7$

⋮ ⋮

$N(20) = N(19) + N(18) + 1 = 17710$

Minimum # of nodes in an AVL tree of height 20 = 17710 //

e)

Utku Baran Torun

```
bool isMinHeap(TreeNode* root)
{
    if( isEmpty() ) // tree is empty root = NULL;
        return false;

    else if ( root's both left and right children aren't NULL)
    {
        if( root's item < left child's item && root's item < right child's item)
            return isMinHeap( root->leftChild) && isMinHeap(root->rightChild);

        else // one of the child is less than parent it violates min-heap
            return false;
    }

    else if (root's left child is not NULL, but right child is NULL)
    {
        if ( root's left child is leaf node)
            return root->item < root->leftChild->item;

        else
            return false;
    }

    else
        return false;

} // end of isMinHeap
```

Question 2

Chart for each data input (Heapsort):

	Data 1	Data 2	Data 3	Data 4	Data 5
Size (Number of Data Points)	1000	2000	3000	4000	5000
Comparison Count	20937	46640	74499	103463	133346

As we can see from the chart that when size gets bigger our comparison count increases as well. As we know heapSort's worst case time complexity is O(n logn) and if we look our chart our experimental results match with theoritical numbers.

Heap Data Structure and Heapsort

In this homework we used max-heap data structure. This structure's essential property is parent node's item is always greater than child node's item. In heap structure there are insert, delete, rebuild and isEmpty functions. In addition to these functions we added maximum, popMaximum functions as well. Data is holded in array for heap. Root is always at 0th index, root's left child is at 1st index and this fashion goes on. Maximum function returns item at 0th index (which is the greatest item in heap) and popMaximum just pops the maximum from heap (deleting root and putting in a container variable). Heapsort is a function that sorts the array by using heap data structure.