

HOMEWORK 03

Part a)

insert 13:

13

insert 6:

13
/
6

insert 3:

6
/ \
3 13

insert 7:

6
/ \
3 13
/
7

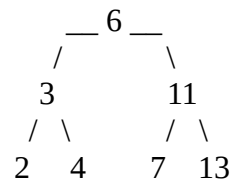
insert 2:

6
/ \
3 13
/ \
2 7

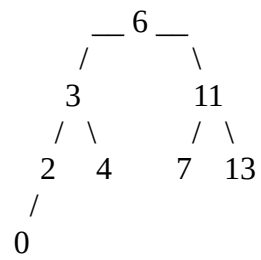
insert 4:

6
— / \
3 13
/ \
2 4 7

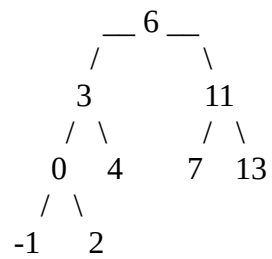
insert 11:



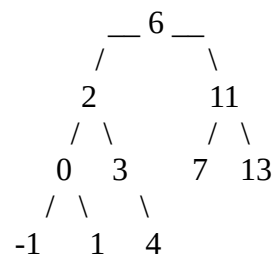
insert 0:



insert -1:



insert 1:

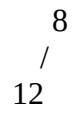


Part b)

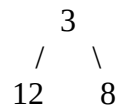
insert 12:

12

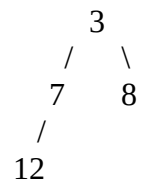
insert 8:



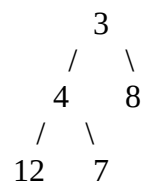
insert 3:



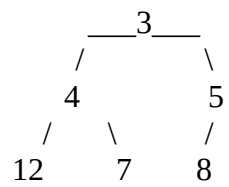
insert 7:



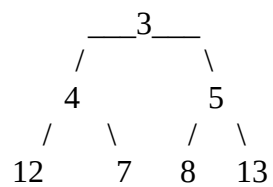
insert 4:



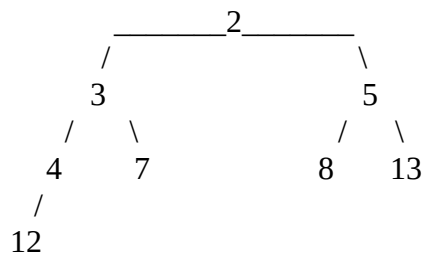
insert 5:



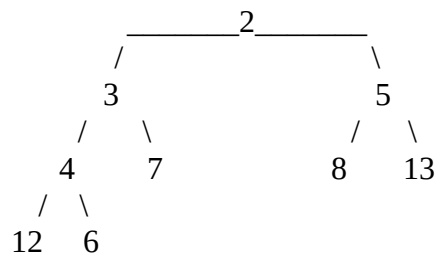
insert 13:



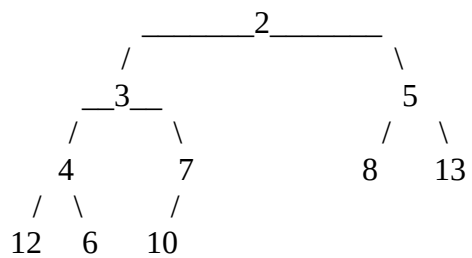
insert 2:



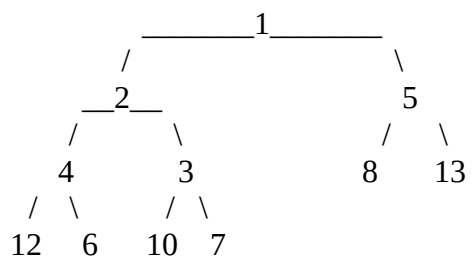
insert 6:



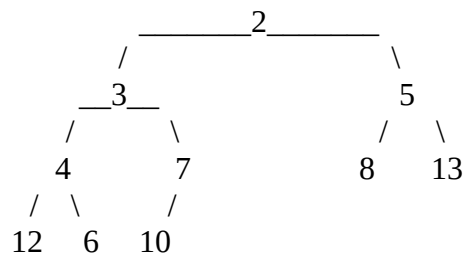
insert 10:



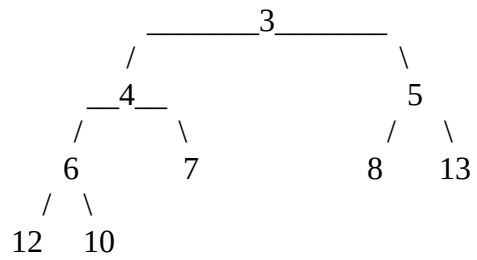
insert 1:



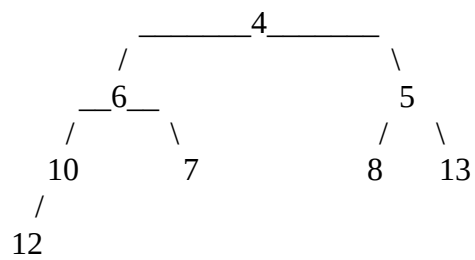
1st deleteMin:



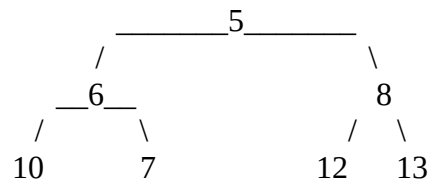
2nd deleteMin:



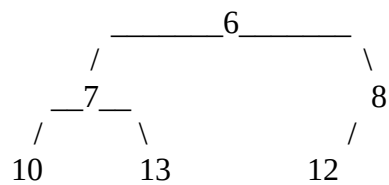
3rd deleteMin:



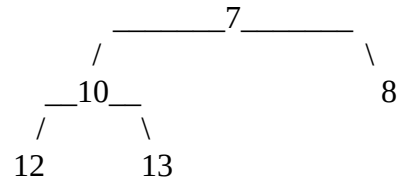
4th deleteMin:



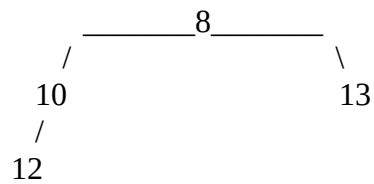
5th deleteMin:



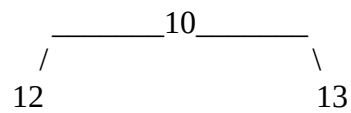
6th deleteMin:



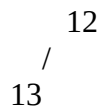
7th deleteMin:



8th deleteMin:



9th deleteMin:



10th deleteMin:

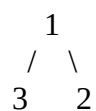


final deleteMin:



Part c)

Consider the following tree:



this tree is a heap. However all of the possible outputs from its pre-order, in-order or post-order traversal results in an unsorted way.

Pre-order traversal : 1 3 2
In-order traversal: 3 1 2
Post-order traversal: 2 3 1

Therefore, just by knowing that the tree is a heap, we can't conclude that the result of its traversals should be in sorted.

Part d)

Q1- Give a precise expression for the minimum number of nodes in an AVL tree of height h .

Solution:

Let $N(h)$ be the function that returns the minimum number of nodes in an AVL tree of height h .

If $h \leq 0$, we know that $N(h) = 0$ since a tree with non-positive height does not exist (so do nodes).

If $h = 1$, we know that $N(h) = 1$.

If $h = 2$, we also know that $N(h) = 2$

However after $h \geq 2$, we can use the recurrence relation $N(h) = N(h-1) + N(h-2) + 1$. Here, +1 comes in order to add the current node to the number of nodes, and without loss of generality we can assume that $N(h-1)$ and $N(h-2)$ denotes the number of nodes in the left and right childs with height $h-1$ and $h-2$ (if their height would be the same than the result would not be minimal, therefore they should be differ by 1).

Now considering that $F_0 = 1, F_1 = 1$ and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$ (In essence F denotes the fibonacci series). We have the following relation that can easily proved by mathematical induction method:

$$F_h = N(h-1) + 1 \text{ or equivalently } N(h) = F_{h+1} - 1$$

Q2- What is the minimum number of nodes in an AVL tree of height 15?

Solution:

$$N(15) = F_{16} - 1 = 1597 - 1 = 1596$$

Part e)

Pseudo-code for determining whether a tree is min-heap:

```
If the tree is empty:
    The tree is a min-heap.
Else if the tree is a complete tree:
    If the node's key value is smaller than node's childrens' key value:
        If left and right subtrees are both min-heap:
            The tree is a min-heap
        Else:
            The tree is not a min-heap
Else:
    The tree is not a min-heap
```

Pseudo-code for determining whether a tree is a complete:

```
If the tree is empty:
    Consider the tree as complete
Else:
    consider the index of the root node as 0
    For each node:
        consider the index as index of the current node
        consider the index of the left node (if there is any) as  $2 * \text{index} + 1$ 
        consider the index of the right node (if there is any) as  $2 * \text{index} + 2$ 

    Consider the tree as complete by default
    For each node:
        if index of the node is larger or equal to number of total nodes in the tree:
            Consider the tree as non-comlete
```

Report

For each data input given, we have the following number of key comparisons with respect to the size:

For size = 1000, we have 19036 key comparisons
For size = 2000, we have 42093 key comparisons
For size = 3000, we have 66646 key comparisons
For size = 4000, we have 92029 key comparisons
For size = 5000, we have 118377 key comparisons

The ratio of key comparison / size for each data test is

test 1: $19036 / 1000 = 19.03$
test 2: $42093 / 2000 = 21.04$

test 3: $66646 / 3000 = 22.21$

test 4: $92029 / 4000 = 23$

test 5: $118377 / 5000 = 23.6$

Since the comparison key / n ratio seems to be growing at $\log n$ scale, we can conclude that the growth rate for comparison key number seems to be $n \cdot \log n$ which actually also matches our theoretical expectations.