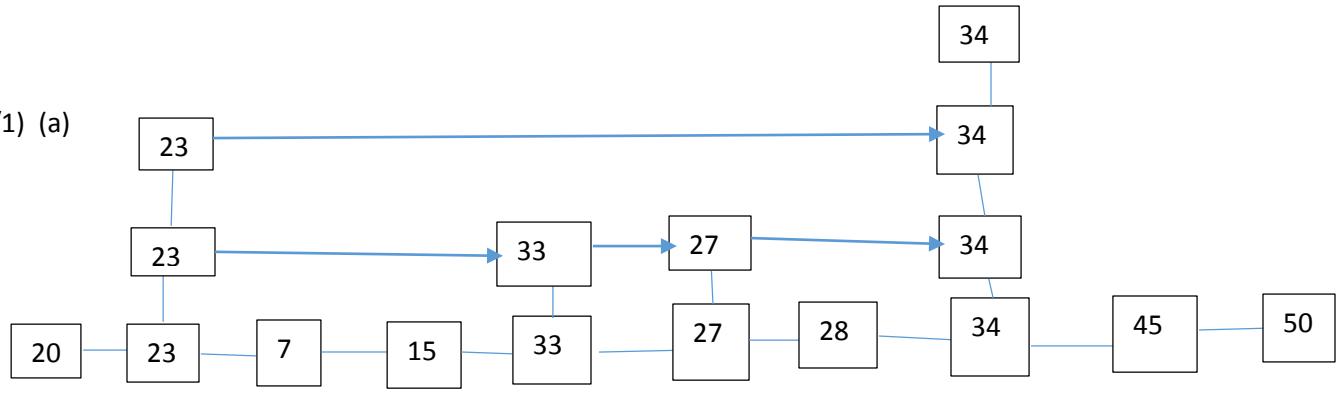
3/1) (a)

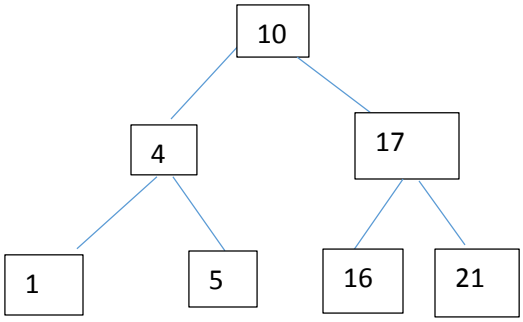


(c) pseudo code of the function atRank(nod,i) is as follows:

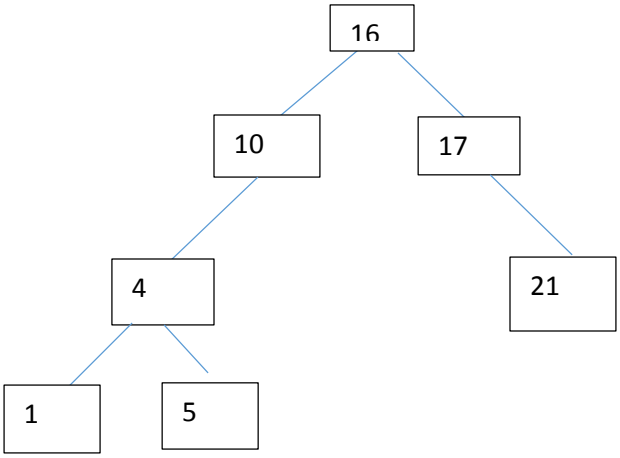
```
if x!=None:
    y=x
    while y!=None:
        y=y.next
    if x.rank<i:
        while x.rank<i and x.next!=None:
            x=x.next
        if x.rank==i:
            print x.da
        elif x.up!=None:
            atRank(x.up,i)
    elif x.rank==i:
        print x.da
    else:
        while x.rank>i and x.pre!=None:
            x=x.pre
        if x.rank==None:
            print x.da
        elif x.up!=None:
            atRank(x.up,i)
```

3/2)

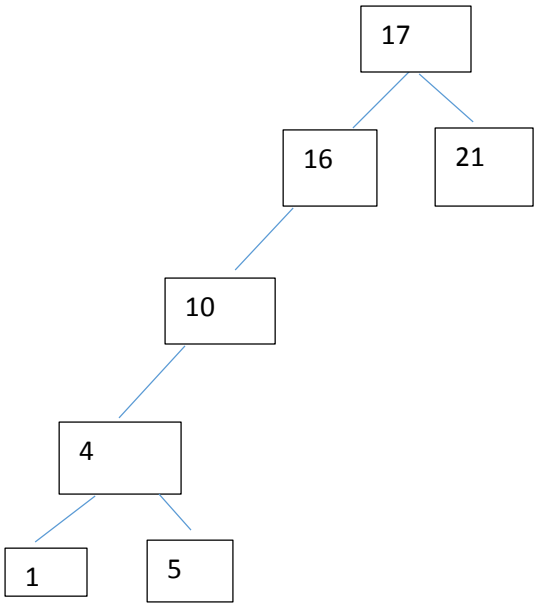
Binary Tree of height 2:



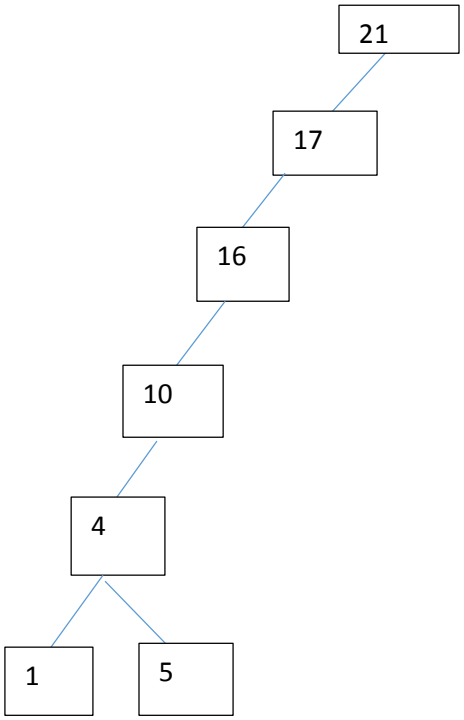
Binary Search Tree of height 3:



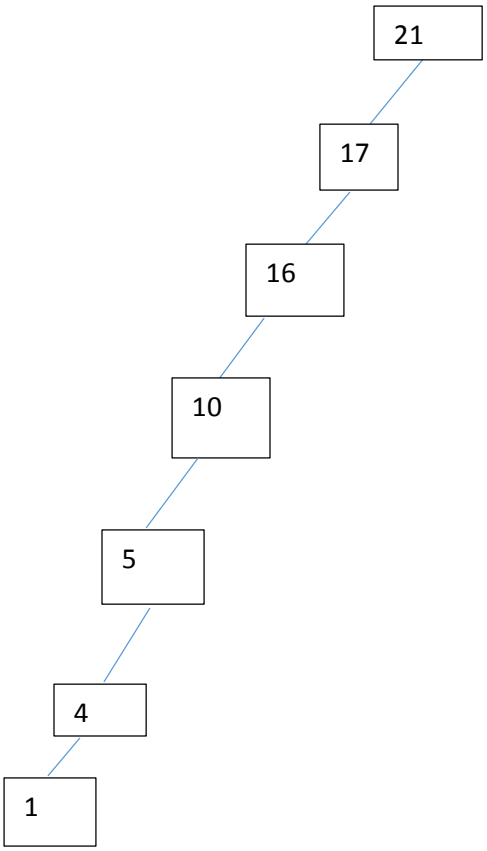
Binary Search Tree of height 4:



Binary Search Tree of height 5:



Binary Search Tree of height 6:



3/3) c.) 925,202,911,240,912,245,363

e.) 935,278,347,621,299,392,358,363

3/4) Let us suppose that there are total of n number of entries and left and right pointers of every leaf and also the parent of root will point to the sentinel whose id is set to 0. Initially id's of every node except sentinel are set to 1.

```
X=T.root
While n>0
    if x.id==1 and x.left.id==1
        X=x.left
    Elsif x.id==1 and x.left.id==0
        x.id=0
        print x.value
        n=n-1
        if x.right.id==1
            x=x.right
        else
            x=x.parent
    elsif x.id==0
        x=x.parent
```

3/5)In **worst case** the tree will be skewed one and hence the worst case running time will be $O(n^2)$

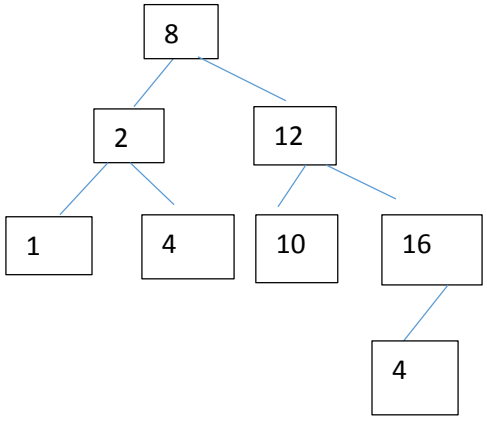
In **best case ,**the tree will be full binary tree and hence running time will be $\Omega(nlgn)$.

3/6).

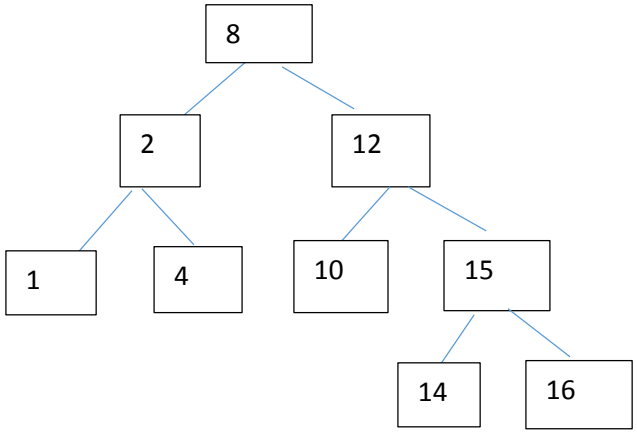
a)Yes, it an AVL tree.

b)

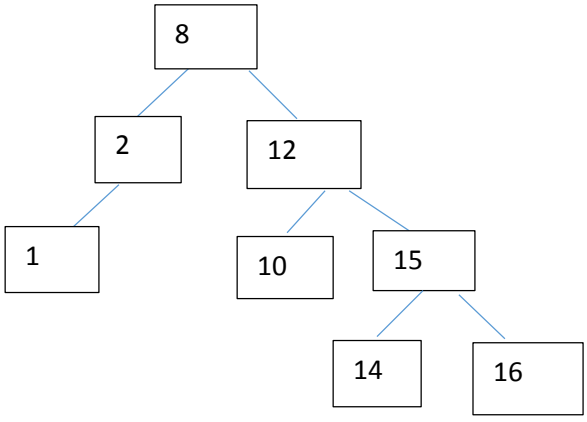
insert(1)



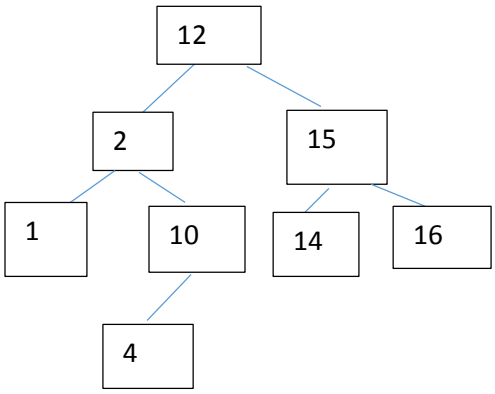
Insert(15)



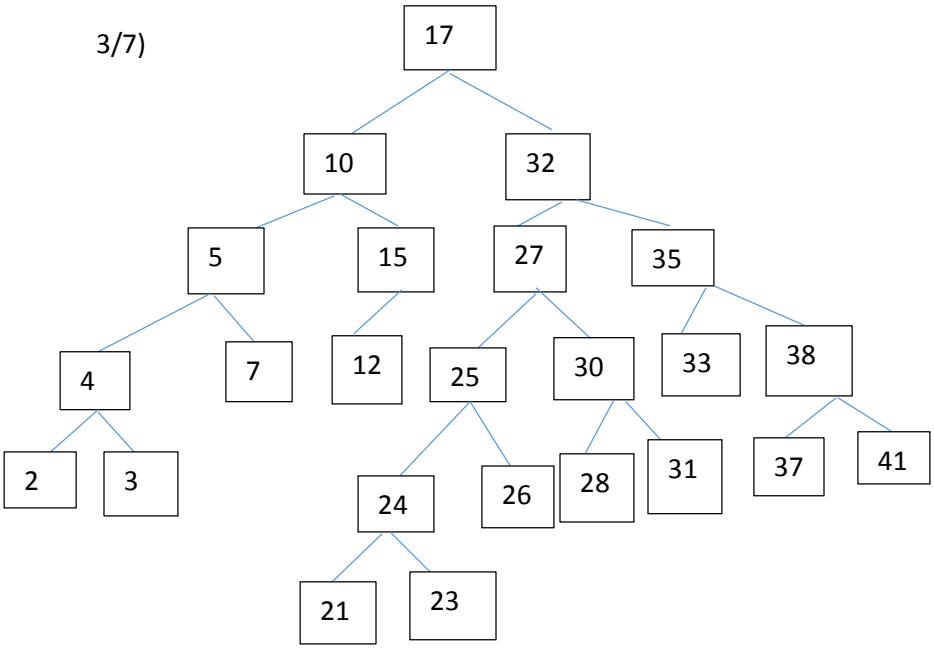
Delete(4)



Delete(8)

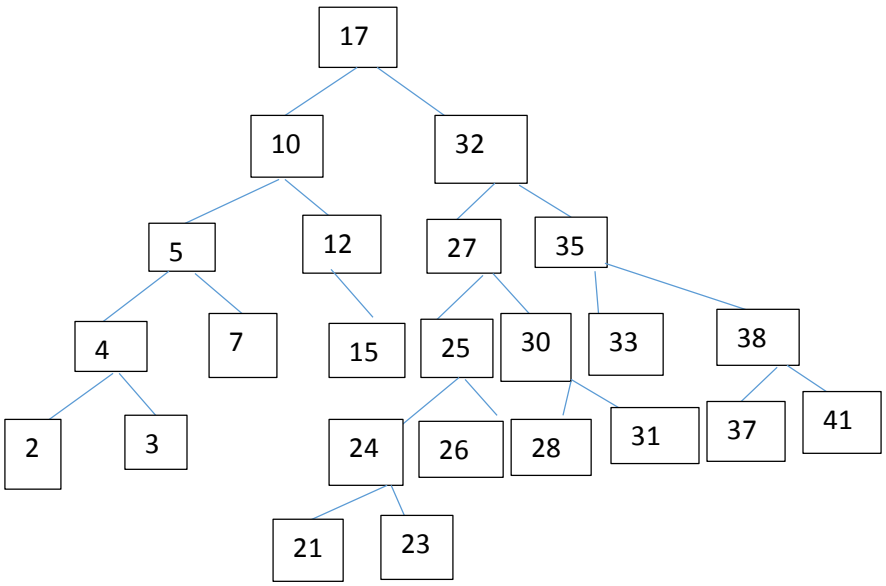


3/7)

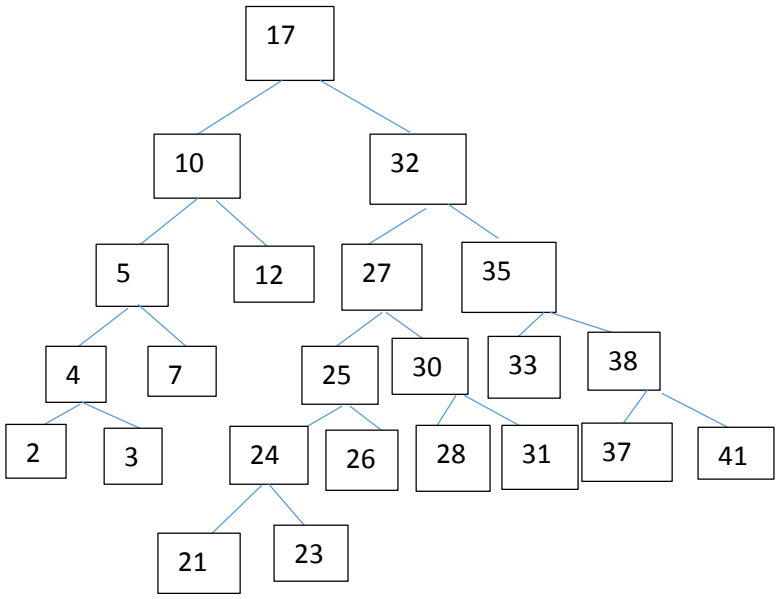


Let us apply delete operation on 15.

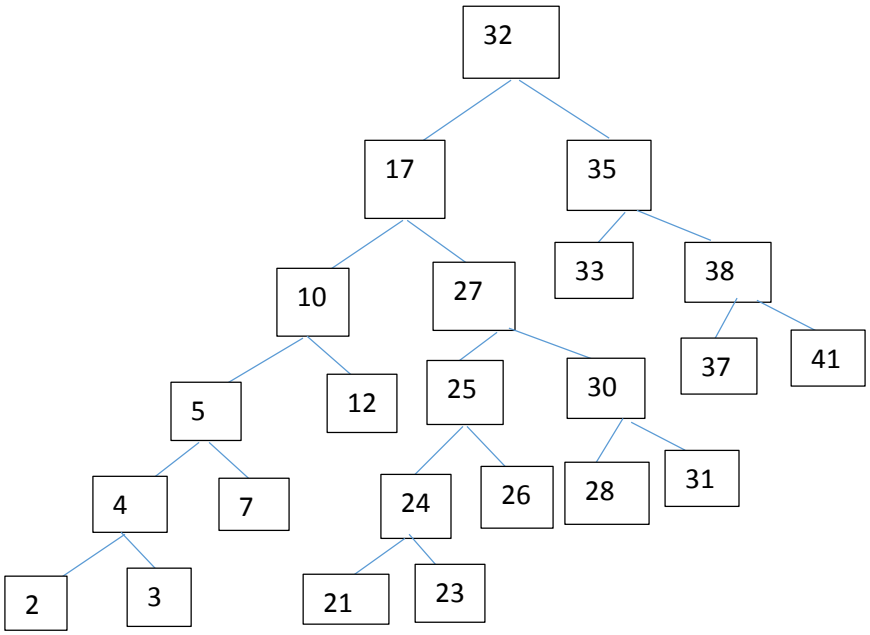
Applying right rotation



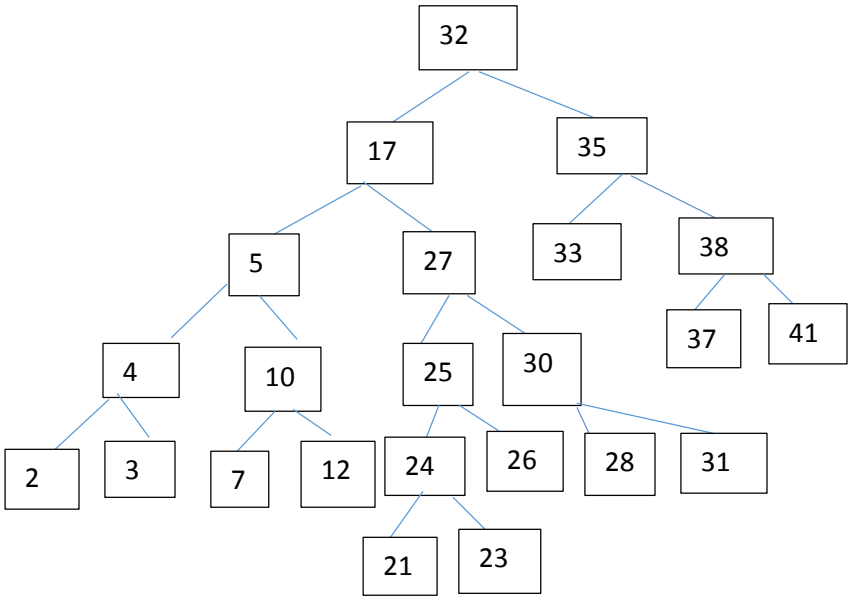
Delete(15)



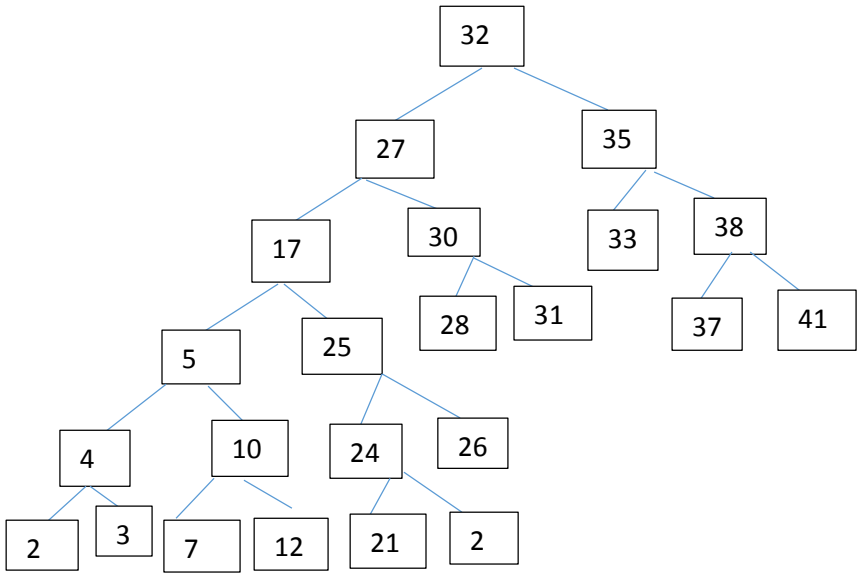
Applying left rotation on 17 and 32



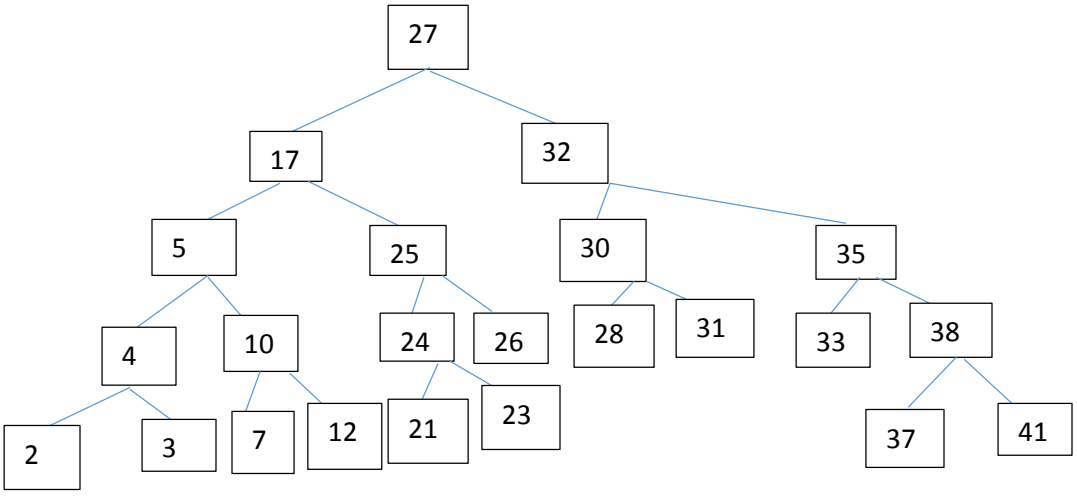
Applying right rotation on 5 and 10



Applying left rotation on 17 and 27

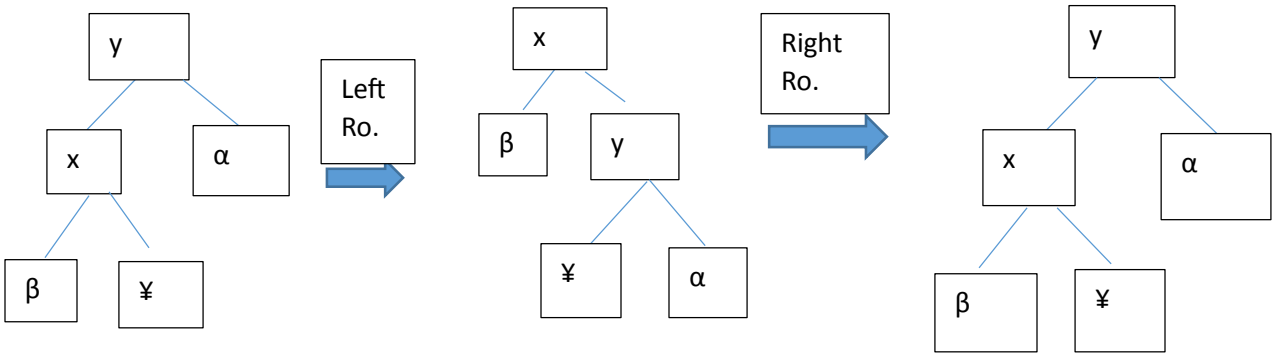


Applying right rotation on 27 and 32



3/8)My algorithm has upper and lower bounds both $2^{n.}$

3/9) Let us consider an example to prove that left and right rotations are inverse of each other.



From the above three figures, it is clear that if we consider two nodes x and y and first apply left rotation on x then we will get figure 2. After that if we apply right rotation on y, then we will get figure 3. But, wait a minute, figure 1 and 3 are exactly same. Therefore left and right rotations are inverse of each other.

No, double rotations are not inverse. Because in splay tress, most of the time we use double rotations but every time we yield a different tree structure.

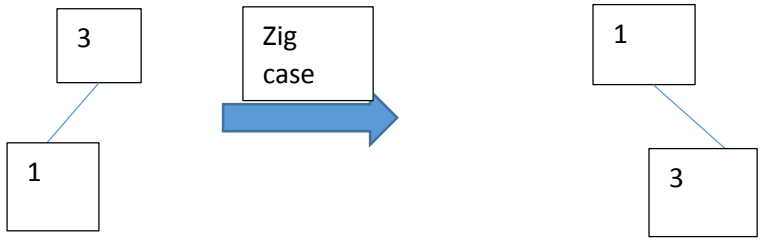
3/10)

(b) I am pointing the left pointer of a leaf to its predecessor in an inorder tree traversal.

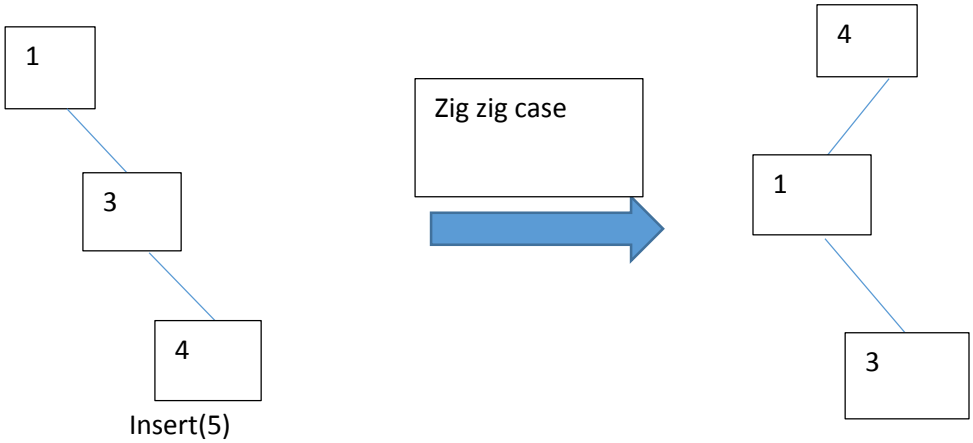
3/11)

a)insert(3)

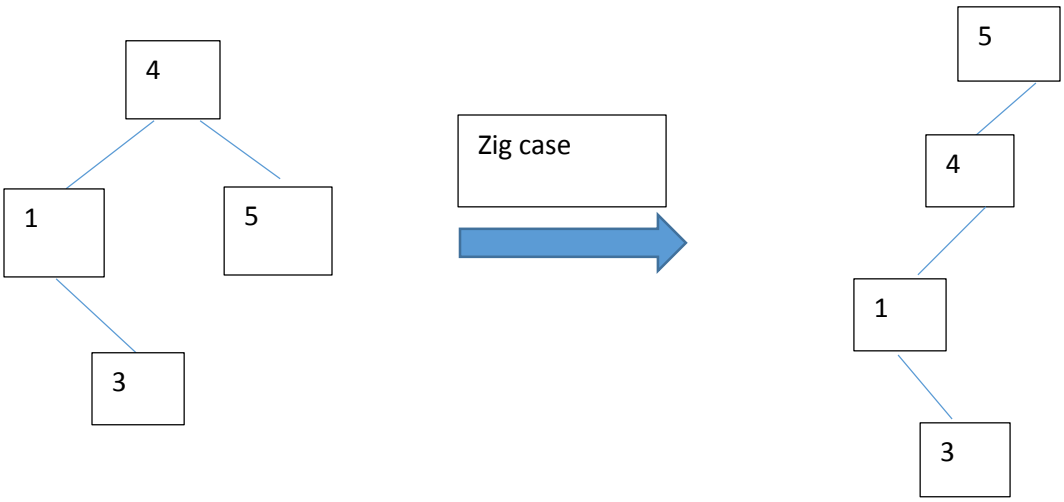
insert(1)



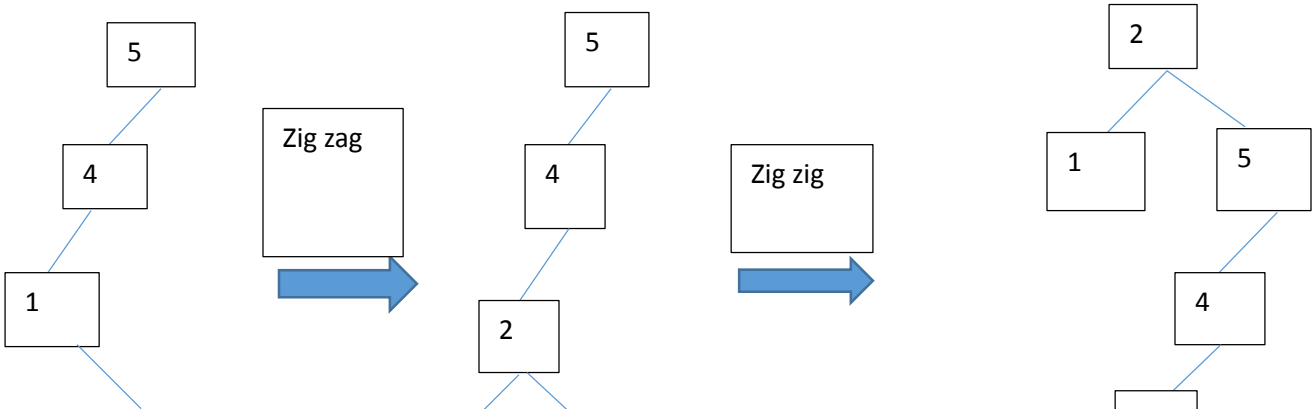
Insert(4)

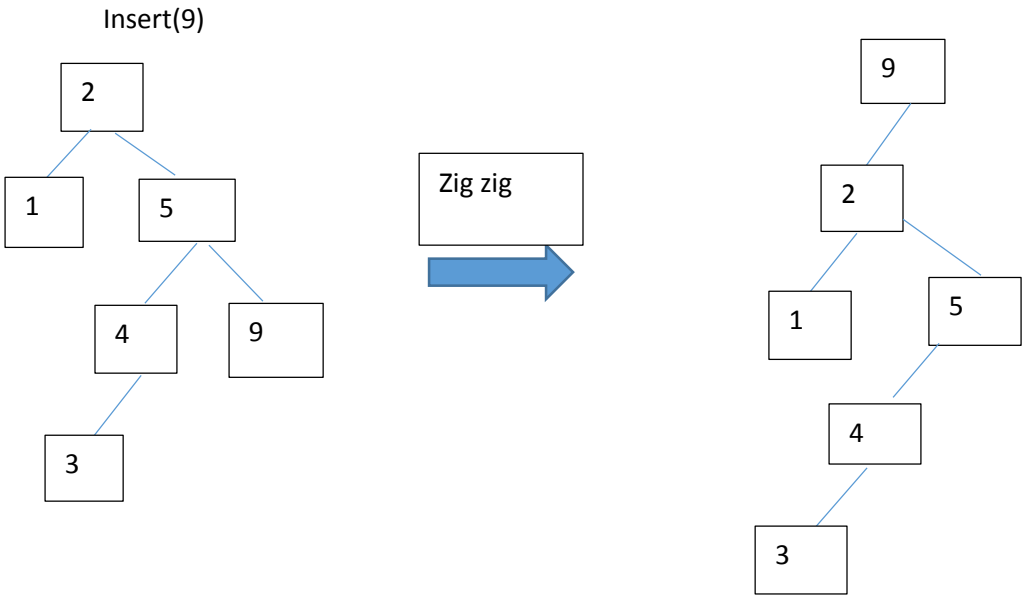


Insert(5)

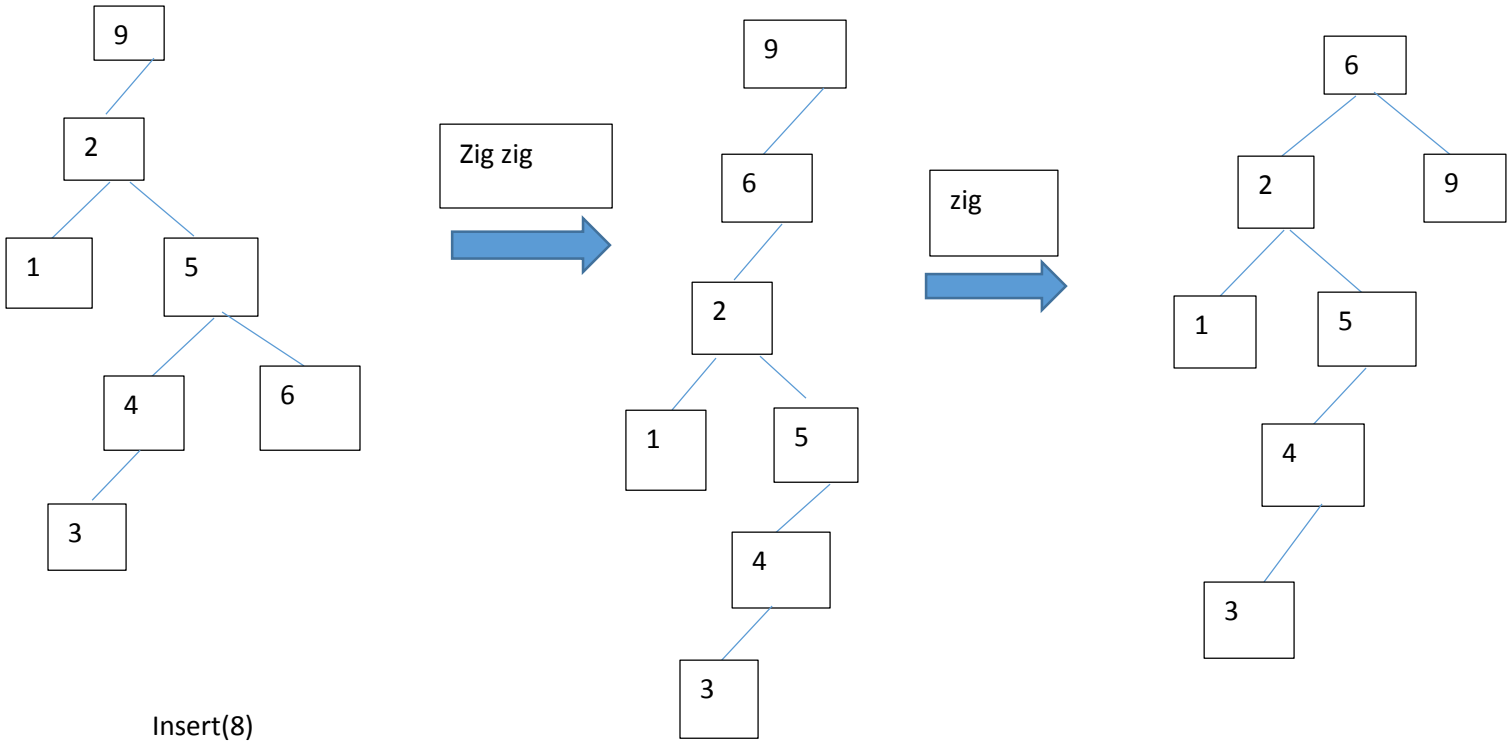


Insert(2)

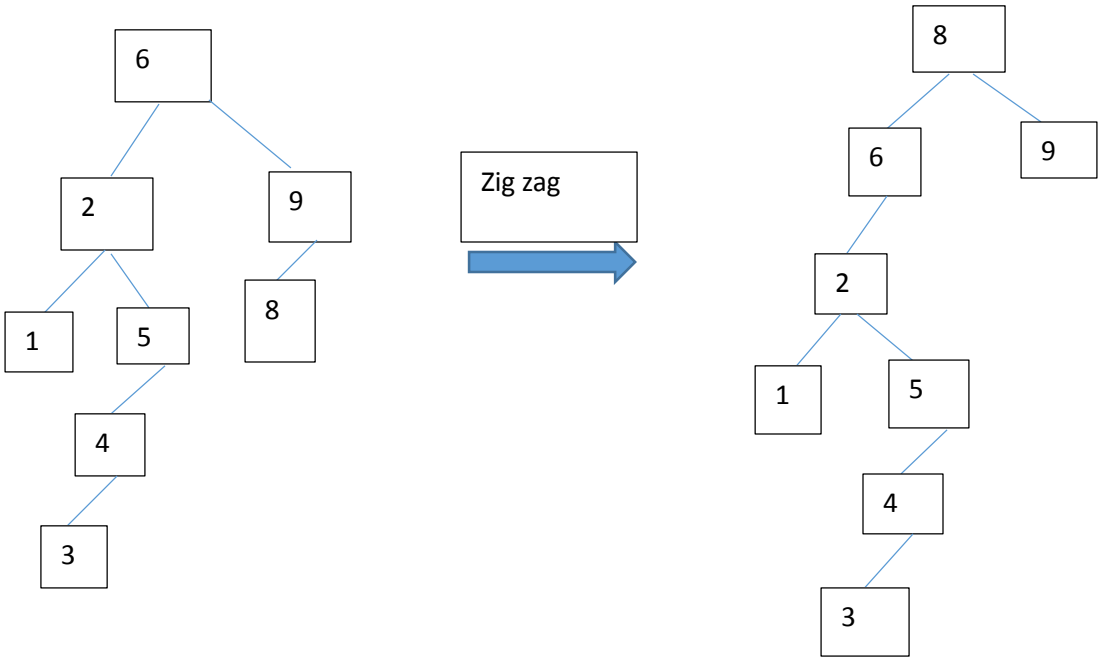


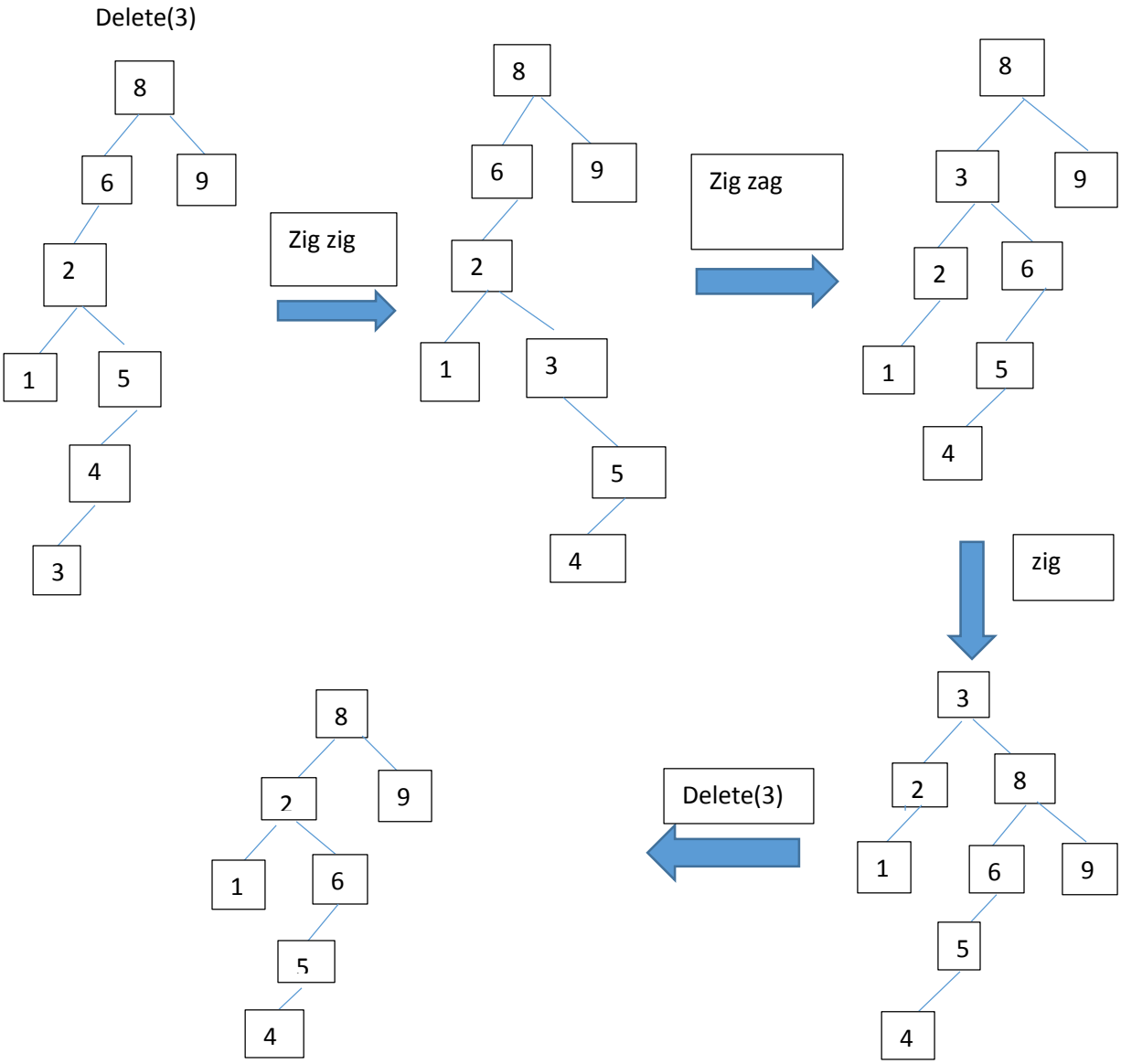

Insert(9)
2
1
5
4
9
3
Zig zig
9
2
1
5
4
3

Insert(6)



Insert(8)




Delete(3)
8
6
9
2
1
5
4
3
Zig zig
8
6
9
2
1
3
5
4
Zig zag
8
3
9
2
6
1
5
4
zig
3
2
8
1
6
9
5
4
Delete(3)
8
2
9
1
6
5
4