1.**In the ordinary BST, let us have an additional variable to be stored at the node, which is the height of the subtree rooted there. Thus every node x has the following attributes: left, right, value, height. Write pseudo-code for insert and delete in BST with the updation of height. Be careful with delete.**

Insertion of a Node

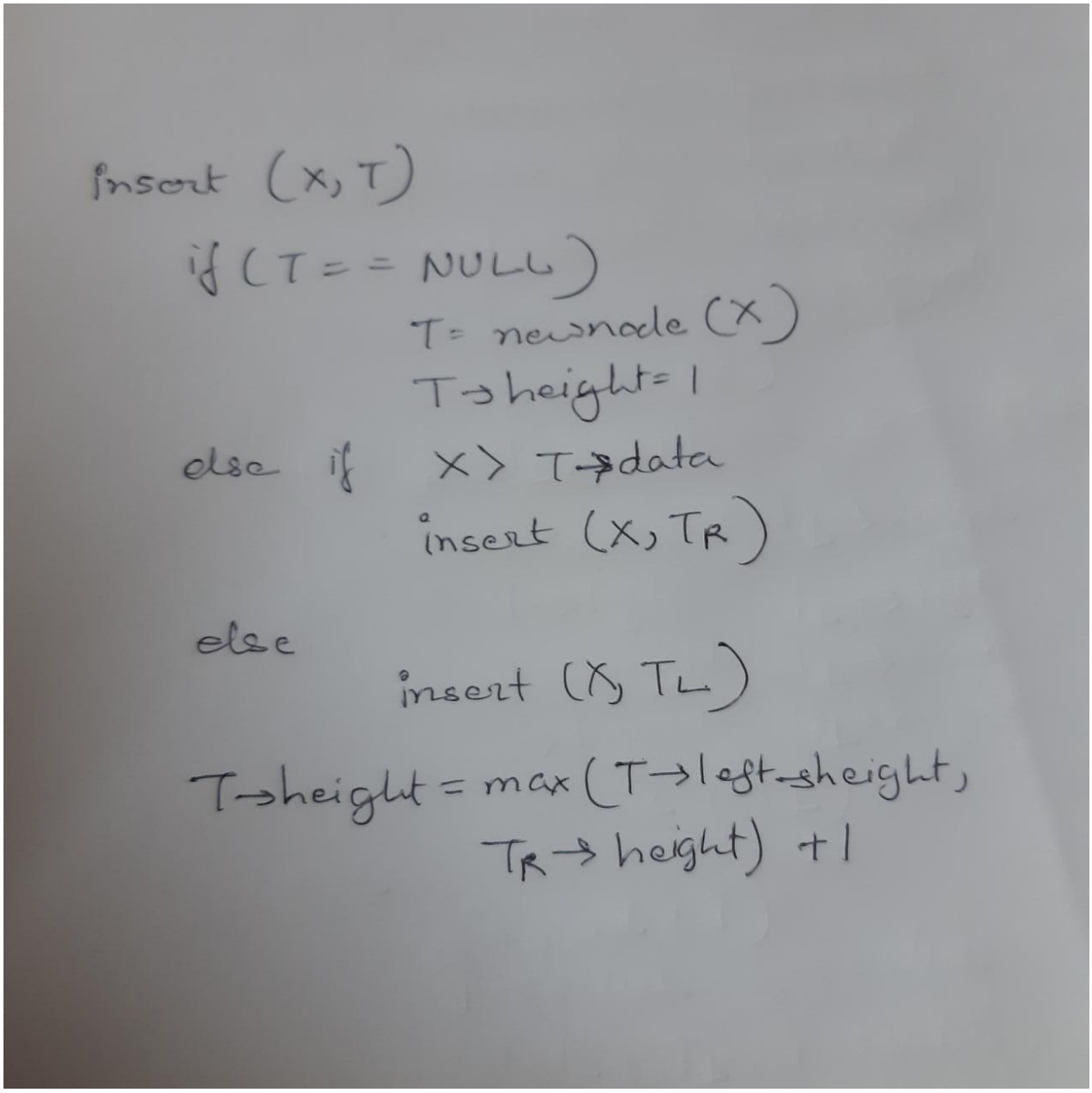
The height of the ancestors of the newly added node will change.

Step1: If the root is NULL create a new node with the value, set height=1 and return the node.

Step2: If the value is greater than the root->data the call the insert function recursively with root->right

Step3: Else insert function is called with root->left

Step4:
root->height=max(root->left->height,root->right->height)+1



Deletion of a Node

Step1:If root is NULL then return

Step2:If key is less than root->data call the delete function recursively with `root->left = deleteNode(root->left, key)`

Step3:Else if key is greater than root->data then

```
root->left = deleteNode(root->left, key)
```

Step4:Else

Step i:If the node is leaf node then return NULL

Step ii: If the node contain single leaf then replace the deleted node witht the leaf node and update the height of the replaced node and all its subtree nodes by decreasing the height by 1

Step iii: If the node has 2 children replace the node with the inoder successor or predecessor and update the height of the replaced node with the deleted node

Step5: Now outside the else we will perform root->height=max(root->left->height,root->right->height)+1

```
delete (x, T)
    a = T → value
    if x < a
        delete (x, T_L)
        T → ht = max (T_L → ht, T_R → ht) + 1

    else if x > a
        delete (X, T_R)
        T → ht = max (T_L → ht, T_R → ht) + 1

    else
        if T_L == NULL && T_R ≠ NULL
            T → value = T_R → value
            T → left = T_R → left
            T → right = T_R → right
            T → height = T_R → height
            dispose (T_R)
        else if T_L ≠ NULL && T_R == NULL
            T → value = T_L → value
            T → left = T_L → left
            T → right = T_L → right
            T → height = T_L → height
            dispose (T_L)
        else
            Q = pred (T)
            T → value = Q → value
```

2.This is to ensure that you understand the analysis of the "random permutation". In Quicksort, we had said that if [a(1),a(2),...,a(n)] is a random permutation, then the probability that a(1)=i is precisely 1/n. Now, for a permutation as above, let b=a(a(1)). In other words, if a=[2,3,1,4,5] then b=3, but if a=[5,4,3,2,1] then b=1 and so on. If I were to select a random permutation, then what is the probability p(i) that b=i. Warning: p(i) is not the same for every i.

$p(i) = P(a(a(1)) = i)$ $\quad (i \in \{1,2,\dots,n\})$

$= \sum_{k=1}^{n} P(a(1) = k \wedge a(k) = i)$

$\underline{i=1}$: $p(i) = \sum_{k=1}^{n} P(a(1) = k \wedge a(k) = 1)$

$= P(a(1) = 1) + \sum_{k=2}^{n} P(a(1) = k \wedge a(k) = 1)$

$= \frac{1}{n} + \sum_{k=2}^{n} \frac{(n-2)!}{n!}$

$= \frac{1}{n} + \frac{n-1}{n(n-1)} = \frac{2}{n}$

$\underline{\underline{i \neq 1}}$: $p(i) = \sum_{k=1}^{n} P(a(1) = k \wedge a(k) = i)$

For $k=1$: $a(1)=1 \wedge a(1)=i$, $i \neq 1$ $\Rightarrow$ Prob. $= 0$

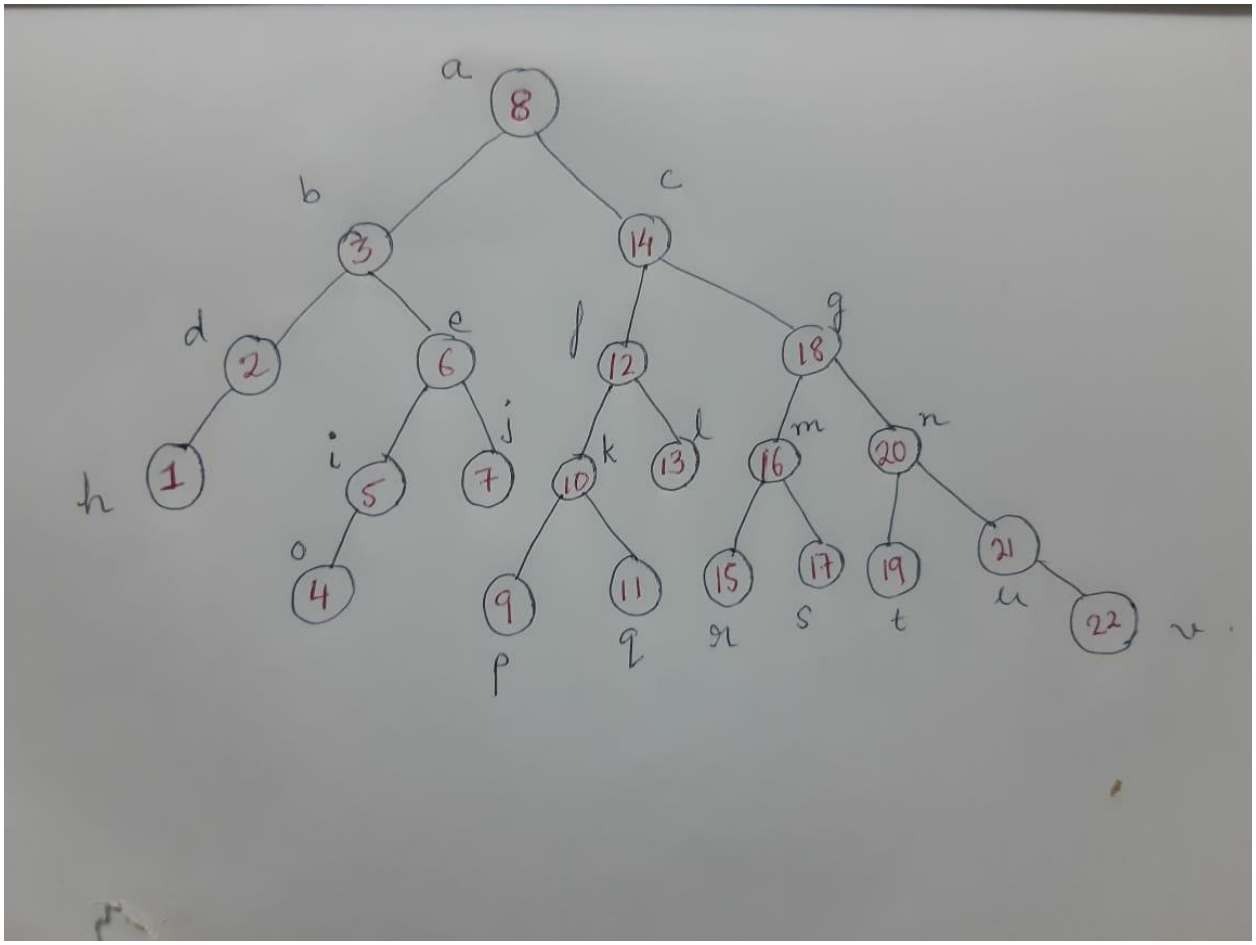
For $k=i$: $a(1)=i \wedge a(i)=i$, $i \neq 1$ $\Rightarrow$ Prob. $= 0$

$= \sum_{k \in \{1,\dots,n\} \setminus \{1,i\}} P(a(1) = k \wedge a(k) = i)$

$= (n-2) \times \frac{(n-2)!}{n!} = \frac{n-2}{n(n-1)}$

3. Consider the structure below. Is the tree structure AVL? If we wish to store the set

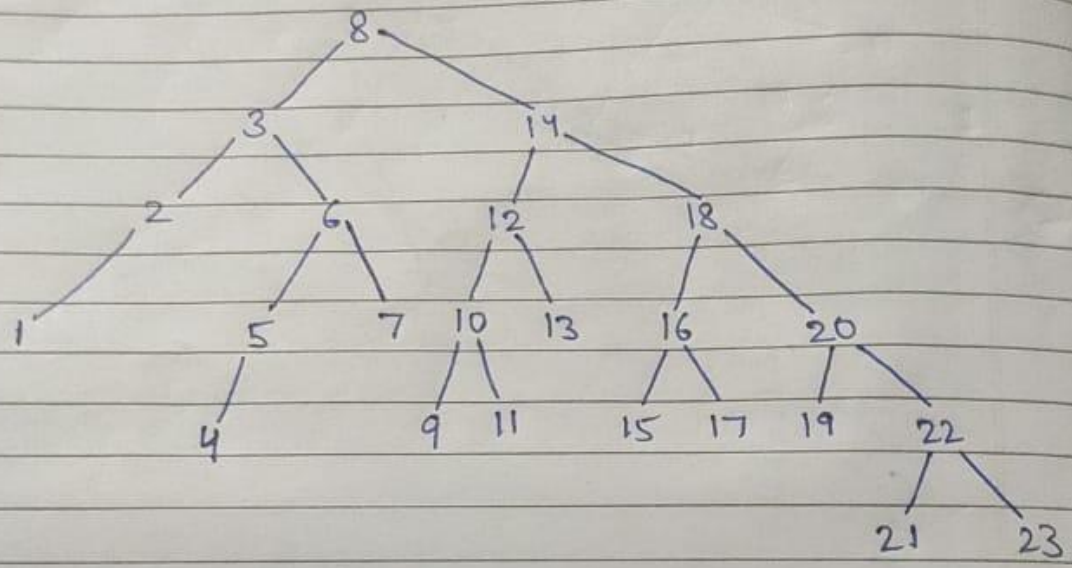
1, . . . , 22, label each node with the correct number.



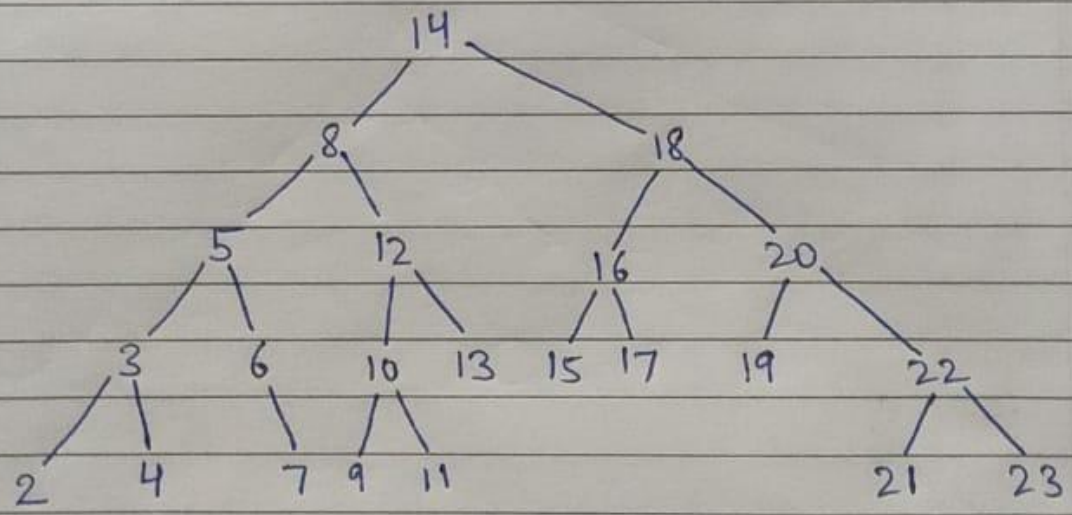
Now add 23 to the set and then delete 1. Also do the same in the reverse order. Are the answers the same? When will the answers be the same?

Page No.
Date

Insert 23 :

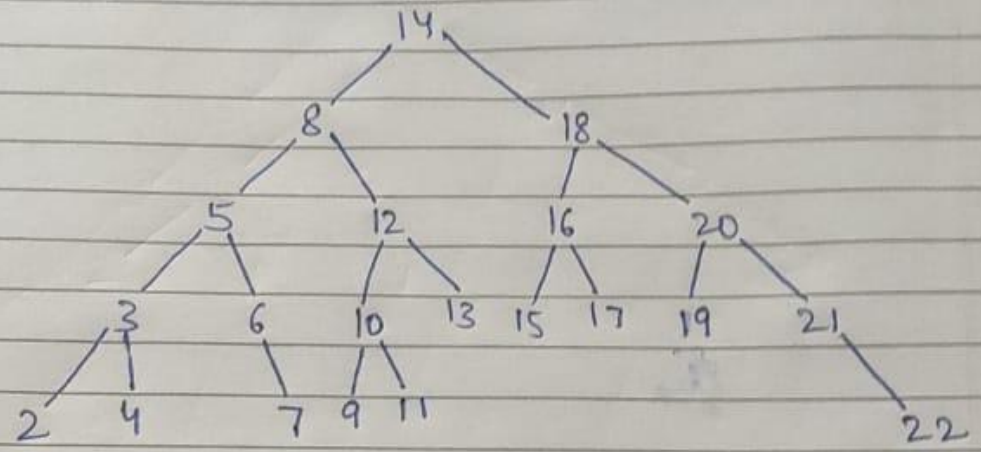


Delete 1 :

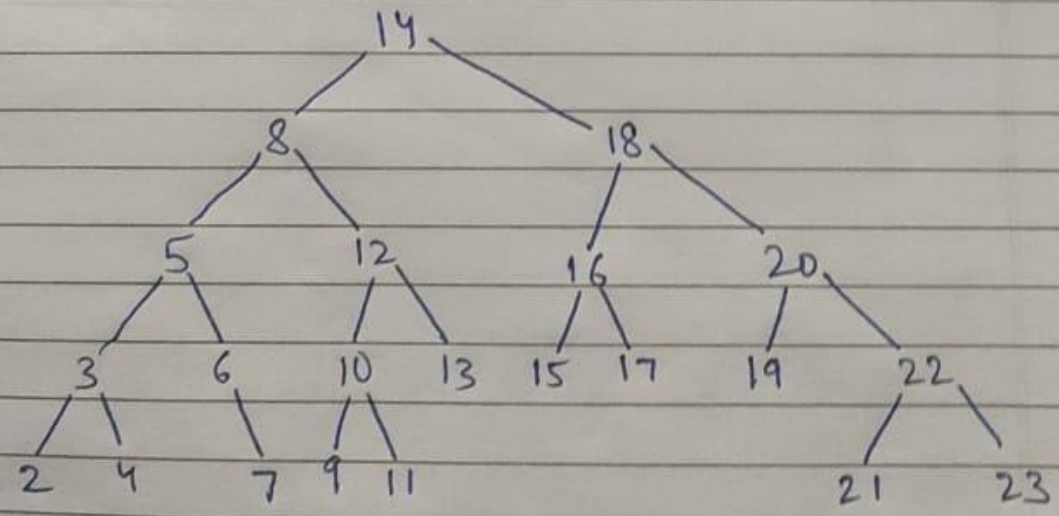


Page No.
Date

Delete 1 :



Insert 23 :



So the results are same irrespective of the order of insertion and deletion.

Why is it that in insert, a single or double rotation is sufficient to balance the tree, but not in deletion?

Ans: In insertion only one node looses its balance so single or double rotation is sufficient.
Once you have deleted, now you have to move up the tree to find the first place where the
imbalance occurs. Having found that you do a rotation, that rotation may or may not solve your
problem. If it does not solve the problem of height balanced, it does not restore height balance
then you might have to continue up from that from that node. And may be once again perform a
rotation, if that is also the problem then you stop otherwise you will have to continue up.