Ch 14. Augmenting Data Structures

14.1 Dynamic Order Statistics

- Recall the problem of finding the i^{th}-smallest element in a set of elements.

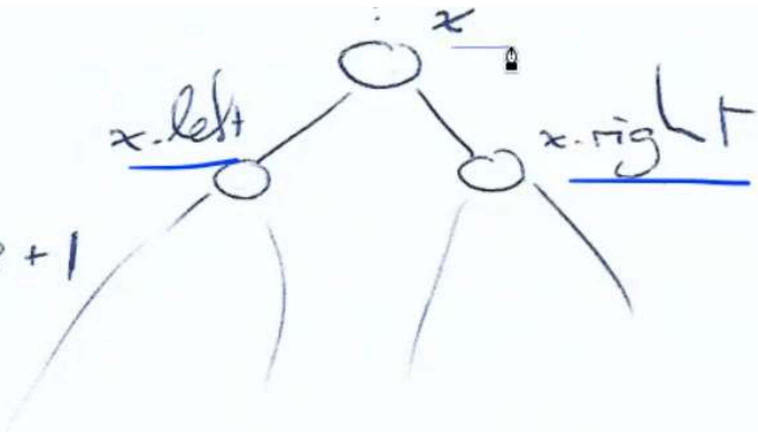
- Here, we consider how to find the i^{th}-smallest element in a set of keys stored as a BST or RB-tree.

- We will need to augment RB-trees with an additional attribute

By an Order-Statistic Tree we mean a RB-tree in which every node has an attribute size that indicates the number of nodes in the subtree at that node.

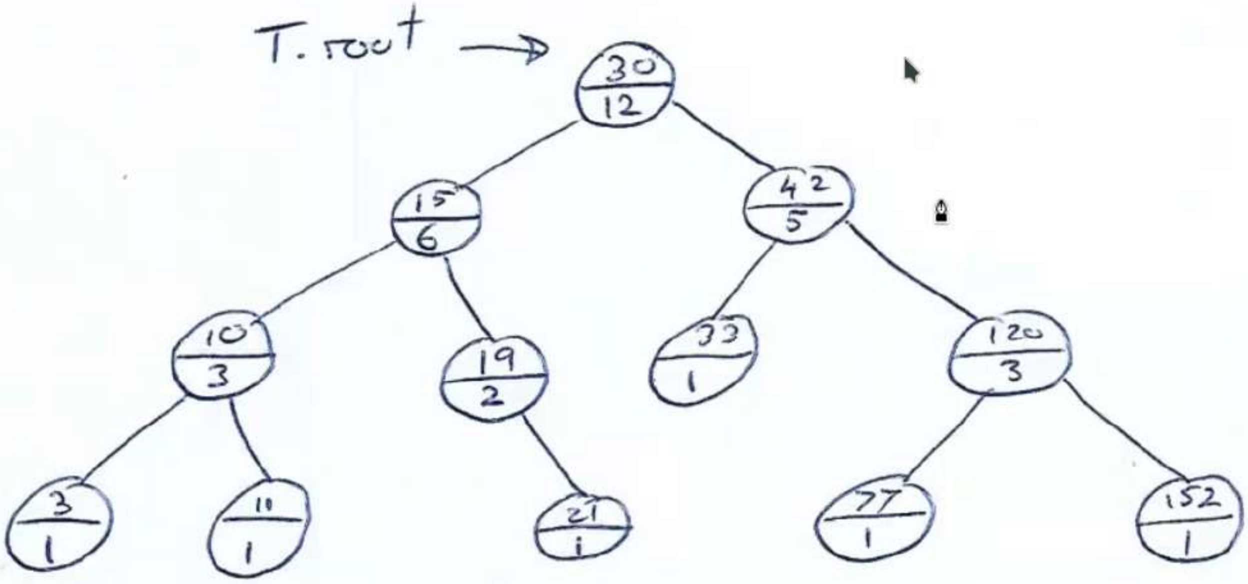
x.size

$= x.left.size + x.right.size + 1$



```
OS-Select (x, i)
r = x.left.size + 1
if i = r
    return x
else if i < r
    return OS-Select(x.left, i)
else return OS-Select (x.right, i-r)
```

$$= O(h) = O(\lg n) \text{ run-time}$$



os-select(T.root, 9)

$r = 6 + 1 = 7$ The root is the 7^{th} smallest key.

$i \geq r$ $(9 \geq 7)$

os-select(T.root.right, 2)

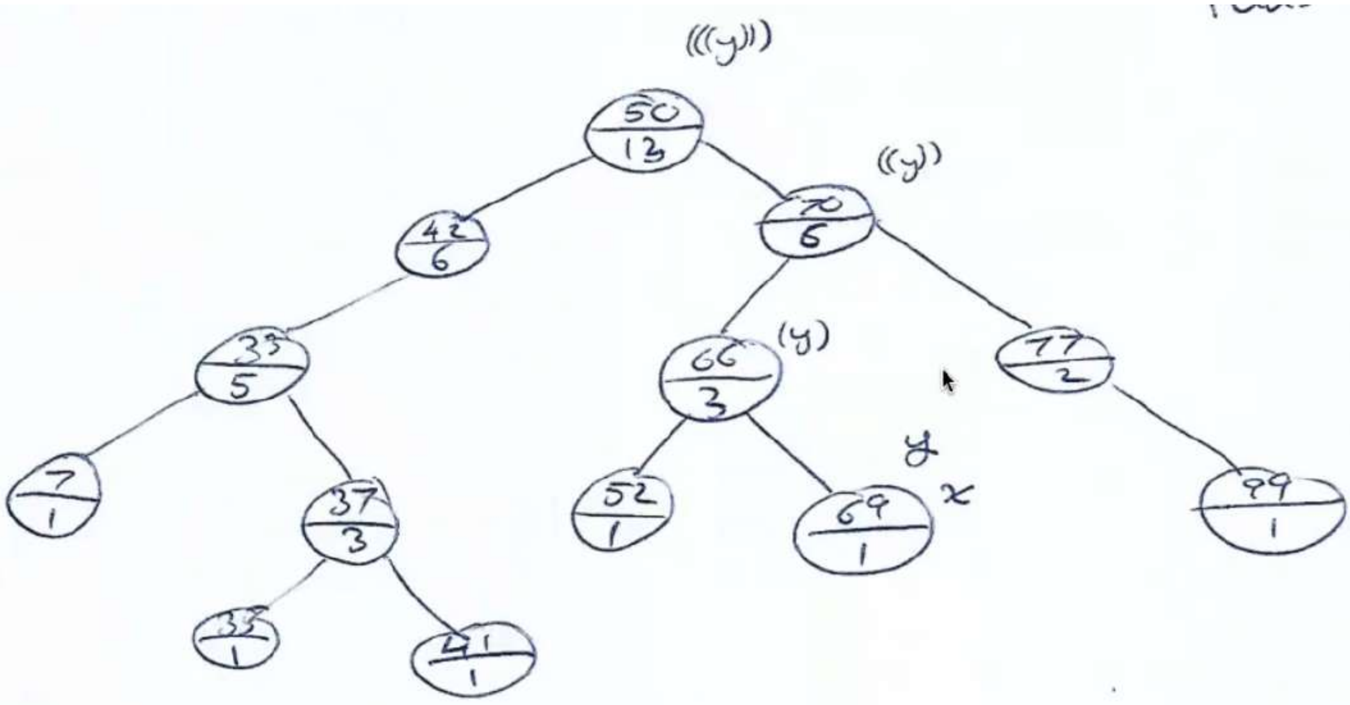
$r = 1 + 1 = 2$

return node with key = 42

OS-Rank (T, x) — Returns the rank of node x in T.

```
r = x.left.size + 1
y = x
while y ≠ T.root
    if y = y.p.right
        r = r + y.p.left.size + 1
    y = y.p
return r
```

$O(h) = O(\log n)$ run-time



(((y)))
50
13
((y))
42
6
70
6
33
5
(y)
66
3
77
2
y
x
7
1
37
3
52
1
69
1
99
1
1
41
1

$$r = \underline{0} + \underline{1} = 1$$

$$r = \underline{1} + \underline{1} + \underline{1} = \underline{3}$$

$$r = 3$$

$$\underline{r = 3 + 6 + 1 = 10}$$

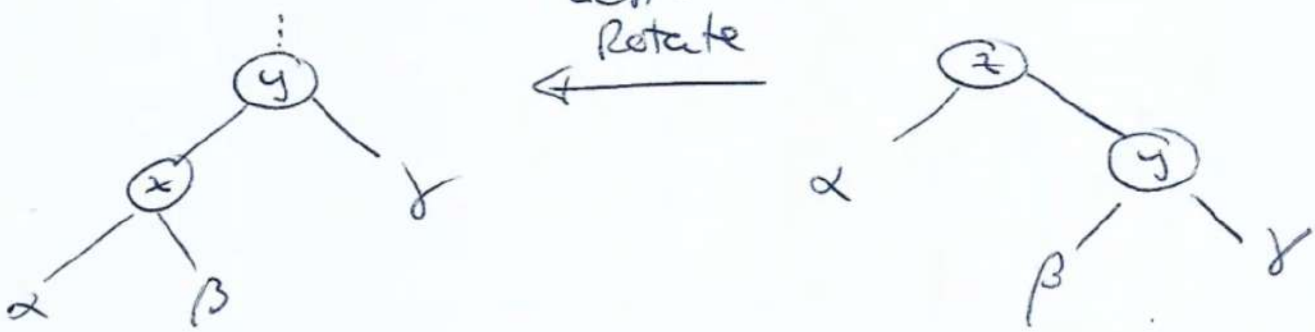
Maintaining the size attributes in an order-statistic Tree after Insert or Delete

<u>Insert</u>: New nodes are inserted at leaves. so the new node will have size =

Then trace a path from the new node to the root and add 1 to the size of each node on the path.

- Can also do this when traversing down to the insertion position.

What about Red-Black-Fixup Step -



$$y.size = x.size$$

$$x.size = x.left.size + x.right.size + 1$$

Similar for Right-Rotate.

Delete : Similar to Insert -

14.2 How to augment a data structure

In Order-Statistic Trees we added the .size attribute which allowed us to do OS-Select and OS-Rank in $O(\log n)$ run-time.

We could augment nodes with other attributes for different applications. what's important is that these attributes can be maintained after Insert or Delete operations.

Let's suppose we have attribute f.

<u>Theorem</u>: If $x.f$ depends only on information at x, $x.left$ and $x.right$, then Insert and Delete can be adapted to maintain f in $O(\log n)$ time.

[Recall: $x.size = x.left.size + x.right.size + 1$]