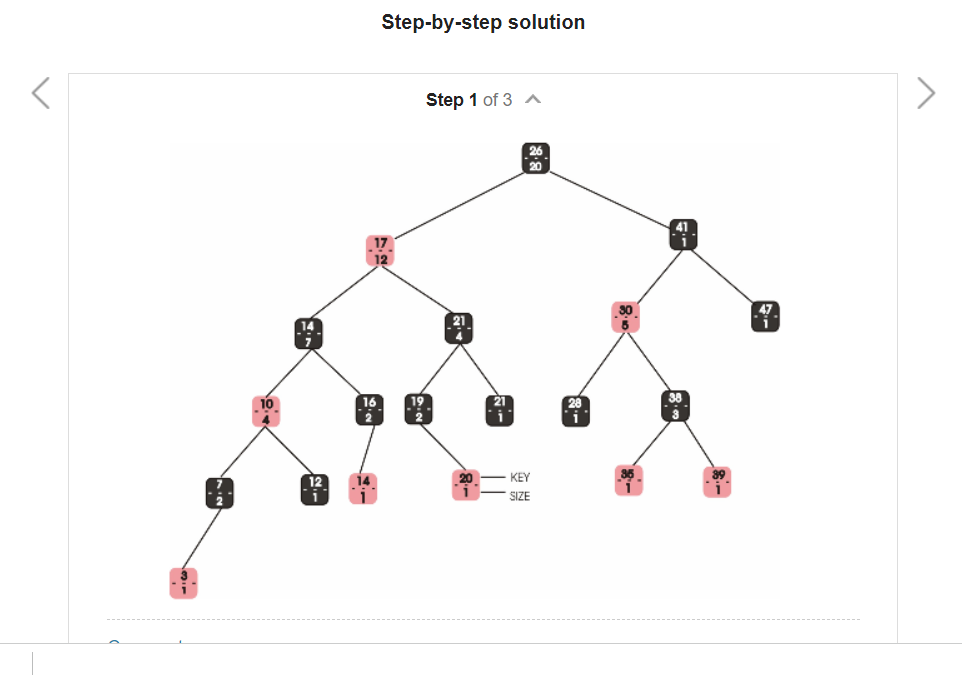
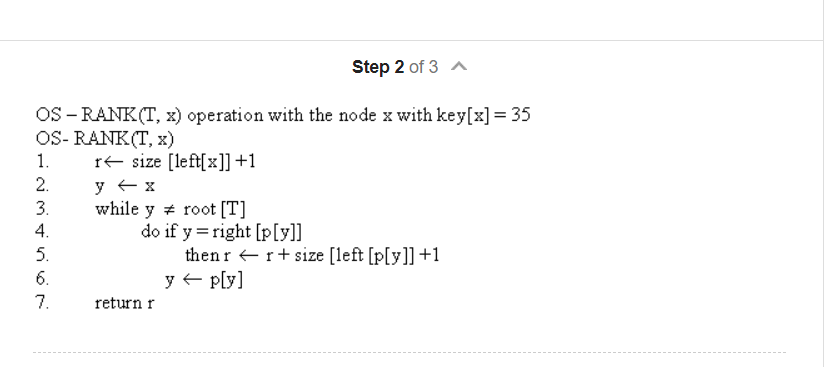
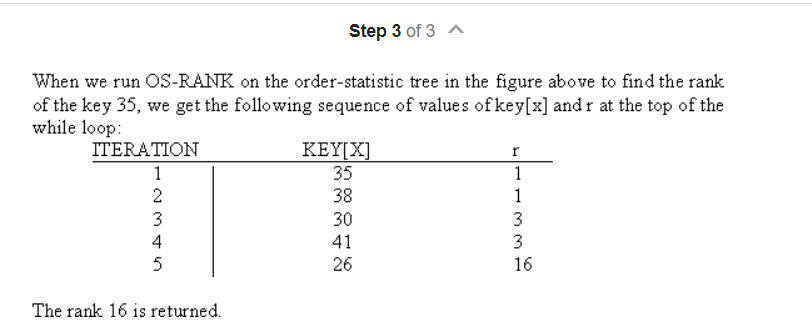
14.1-1 Show how OS-SELECT.T:root; 10/ operates on the red-black tree T of Figure 14.1.?

Ans)







14.1-2)

**1) We begin with an operation that retrieves an element with a given rank. The procedure OS-SELECT(x, i) returns a pointer to the node containing the ith smallest key in the subtree rooted at x. To find the ith smallest key in an order-statistic tree T, we call OS-SELECT(root[T], i).**

OS-SELECT(x,i)

1 r size[left[x]] + 1

2 **if** i = r

3 **then return** x

4 **elseif** i < r

5 **then return** OS-SELECT(left[x],i)

6 **else return** OS-SELECT(right[x],i - r)

The value of size[left[x]] is the number of nodes that come before x in an inorder tree walk of the subtree rooted at x. Thus, size[left[x]] + 1 is the rank of x within the subtree rooted at x.

In line 1 of OS-SELECT, we compute r, the rank of node x within the subtree rooted at x. If i = r, then node xis the ith smallest element, so we return x in line 3. If i < r, then the ith smallest element is in x's left subtree, so we recurse on left[x] in line 5. If i > r, then the ith smallest element is in x's right subtree. Since there are relements in the subtree rooted at x that come before x's right subtree in an inorder tree walk, the ith smallest element in the subtree rooted at x is the (i - r)th smallest element in the subtree rooted at right[x]. This element is determined recursively in line 6.

**Example:**To see how OS-SELECT operates, consider a search for the 17th smallest element in the order-statistic tree of above algorithm. We begin with x as the root, whose key is 26, and with i = 9. Since the size of 26's left subtree is 12, its rank is 13. Thus, we know that the node with rank 13 is greater than i=9. After the recursive call, x is the node with key 17, and i = 9. Since the size of 17's left subtree is 7, its rank within its subtree is 8. Thus, we know that the node with rank 8 is less than i=9. Thus,we recurse once again to find the 9-8=1 s t smallest element in subtree rooted at the node with key 21. We now find that its left subtree has size 2, which means it is the third smallest element.

After the recursive call,x is the node with key 19.It's rank is 1 and i=1.

Thus, a pointer to the **node with key 19** is returned by the procedure.

**2) Given a pointer to a node x in an order-statistic tree T, the procedure OS-RANK returns the position ofx in the linear order determined by an inorder tree walk of T.**

OS-RANK(T,x)

1 r size[left[x]] + 1

2 y x

3 **while** y root[T]

4 **do if**y = right[p[y]]

5 **then** r r + size[left[p[y]]] + 1

6 y p[y]

7 **return**r

The procedure works as follows. The rank of x can be viewed as the number of nodes preceding x in an inorder tree walk, plus 1 for x itself. The following invariant is maintained: at the top of the **while** loop of lines 3-6, r is the rank of key[x] in the subtree rooted at node y. We maintain this invariant as follows. In line 1, we set r to be the rank of key[x] within the subtree rooted at x. Setting y x in line 2 makes the invariant true the first time the test in line 3 executes. In each iteration of the **while** loop, we consider the subtree rooted at p[y]. We have already counted the number of nodes in the subtree rooted at node y that precede x in an inorder walk, so we must add the nodes in the subtree rooted at y's sibling that precede x in an inorder walk, plus 1 for p[y] if it, too, precedes x. If y is a left child, then neither p[y] nor any node in p[y]'s right subtree precedes x, so we leave r alone. Otherwise, y is a right child and all the nodes in p[y]'s left subtree precede x, as does p[y] itself. Thus, in line 5, we add size[left[y]] + 1 to the current value of r Setting y p[y] makes the invariant true for the next iteration. When y = root[T], the procedure returns the value of r, which is now the rank of key[x].

**Example explanation:**

As an example, when we run OS-RANK on the order-statistic tree of algorithm to find the rank of the node with key 35, we get the following sequence of values of key[y] and r at the top of the **while** loop:

iteration **key**[**y**] **r**

--------------------

1 35 1

2 38 1

3 30 3

4 41 3

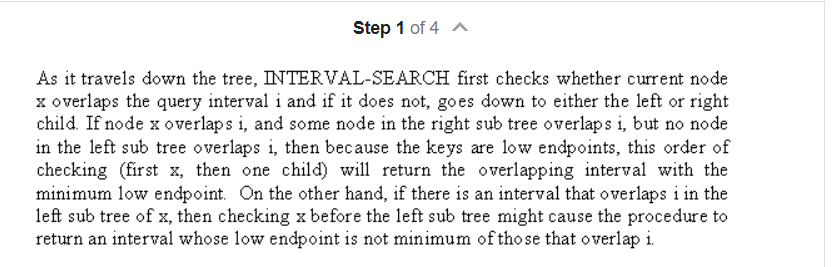
5 26 16

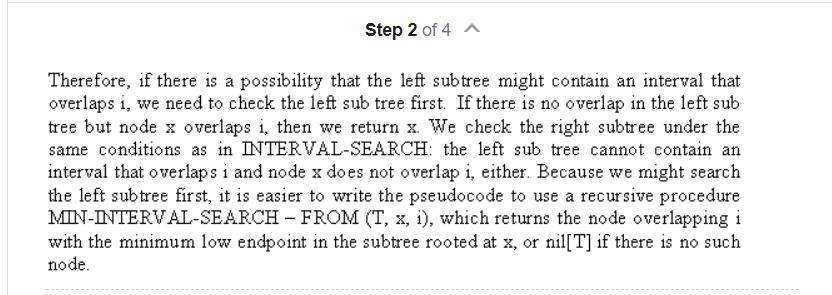
**The rank 16 is returned**

**14.1.5)**

For this our data structure should have these 2 operations:  
1) Get(i)-which gives the key at ith position of the total order of keys  
2) Rank(x)- which gives the position of x in total order of keys  
Now we work on this Get(Rank(x) + i)  
In an order statistic tree, each and every node x keeps the record of the number of nodes contained in the subtree rooted in x.  
Using these 2 operations will keep the track of no. of nodes lie to the left of our path

14.3.3)





MIN-INTERVAL-SEARCH(T,i):

x = T.root

while x != T.nil:

if x.left != T.nil and i.low <= x.left.max:

x = x.left

elif x.int overlaps i:

break

else

x = x.right

return x

