(1) Exercise 14.1-1 on Page 344 [20 points]
Show how OS-SELECT(T.root,10) operates on the red-black tree T of Figure 14.1.
Answer:
OS-SELECT(T.root, 10): r=12+1=13 > i=10
OS-SELECT(T.root.left, 10): r=7+1=8 < i=10, i-r=2
OS-SELECT(T.root.left.right, 2): r=2+1=3 > i=2
OS-SELECT(T.root.left.right.left.right, 2): r=0+1=1 < i=2, i-r=1
OS-SELECT(T.root.left.right.left.right, 1): r=0+1=1 = i
return T.root.left.right.left.right -> 20

(2) Exercise 14.1-2 on Page 344 [20 points]
Show how OS-RANK(T, x) operates on the red-black tree T of Figure 14.1 and the node x with x.key = 35.
Answer:
x.key=35
r=0+1=1, y=x->35
r=1, y=y.p->38
r=1+1+1=3, y=y.p->30
r=3, y=y.p->41
r=3+12+1=16, y=y.p->26
y=T.root
return r=16

(3) Exercise 14.1-5 on Page 344 [30 points]
Given an element x in an n-node order-statistic tree and a natural number i, how can we determine the i th successor of x in the linear order of the tree in O(lgn) time?
Answer:
Given an element x in an n-node order-statistic tree T and a natural number i, the following procedure retrieves the i th successor of x in the linear order of T:
OS-SUCCESSOR(T,x,i)
   r = OS-RANK(T,x)
   s = r + i
   return OS-SELECT(T.root, s)
Since OS-RANK and OS-SELECT each take O(lgn) time, so does the procedure OS-SUCCESSOR.
Exercise 14.3-3 on Page 353 [30 points]

Describe an efficient algorithm that, given an interval i, returns an interval overlapping i that has the minimum low endpoint, or T.nil if no such interval exists.

Answer:

As it travels down the tree, INTERVAL-SEARCH first checks whether current node x overlaps the query interval i and, if it does not, goes down to either the left or right child. If node x overlaps i, and some node in the right subtree overlaps i, but no node in the left subtree overlaps i, then because the keys are low endpoints, this order of checking (first x, then one child) will return the overlapping interval with the minimum low endpoint. On the other hand, if there is an interval that overlaps i in the left subtree of x, then checking x before the left subtree might cause the procedure to return an interval whose low endpoint is not the minimum of those that overlap i. Therefore, if there is a possibility that the left subtree might contain an interval that overlaps i, we need to check the left subtree first. If there is no overlap in the left subtree but node x overlaps i, then we return x. We check the right subtree under the same conditions as in INTERVAL-SEARCH: the left subtree cannot contain an interval that overlaps i, and node x does not overlap i, either.

Because we might search the left subtree first, it is easier to write the pseudocode to use a recursive procedure MIN-INTERVAL-SEARCH-FROM(T, x, i), which returns the node overlapping i with the minimum low endpoint in the subtree rooted at x, or T.nil if there is no such node.

MIN-INTERVAL-SEARCH(T, i)
  return MIN-INTERVAL-SEARCH-FROM(T, T.root, i)

MIN-INTERVAL-SEARCH-FROM(T, T.root, i)
  if x.left ≠ T.nil and x.left.max ≥ i.low
    y = MIN-INTERVAL-SEARCH-FROM(T, x.left, i)
    if y ≠ T.nil
      return y
    elseif i overlaps x.int
      return x
    else return T.nil
  elseif i overlaps x.int
    return x
  else return MIN-INTERVAL-SEARCH-FROM(T, x.right, i)

The call MIN-INTERVAL-SEARCH(T, i) takes O(\text{lg} n) time, since each recursive call of MIN-INTERVAL-SEARCH-FROM goes one node lower in the tree, and the height of the tree is O(\text{lg} n).