for node **v**, and **bf** is the balance factor of **v**. For example, in the Red-Black tree of Figure 1 below, the height of the node containing 13 is 1 and the balance factor is 0. Similarly, the height of the node containing 10 is 3 and the balance factor is 0 whereas, the height of the node containing 15 is 4 and the balance factor is +1 (since the height of its left subtree is one more than the height of its right subtree, i.e., 3 - 2 = 1).

(b) [5 points] What is the time complexity of your algorithm if the tree is a red-black tree? What if the tree is neither a red-black tree nor an AVL tree? Justify your answers.

Sequence ADT:
first(), last(), before(p), after(p), replaceElement(p, o),
swapElements(p, q), insertBefore(p, o), insertAfter(p, o),
insertFirst(o), insertLast(o), remove(p), size(), isEmpty(),
elemAtRank(r), replaceAtRank(r, o), insertAtRank(r, o), removeAtRank(r),
atRank(r), rankOf(p)

Dictionary ADT
findElement(k), insertItem(k, e), removeElement(k), items(), keys(), elements()

OrderedDictionary
findElement(k), insertItem(k, e), removeElement(k), items(),
closestKeyBefore(k), closestKeyAfter(k),
closestElemBefore(k), closestElemAfter(k)

PriorityQueue
removeMin(), minKey(), minElement(k), insertItem(k, e)

BinaryTree ADT:
root(), parent(v), children(v), leftChild(v), rightChild(v), sibling(v),
isInternal(v), isExternal(v), isRoot(v), size(), elements(), positions(),
swapElements(v, w), replaceElement(v, e)

2-4 and Red-Black Trees:

38. [5 points] Draw the corresponding 2-4 tree for the red-black tree in Figure 1.

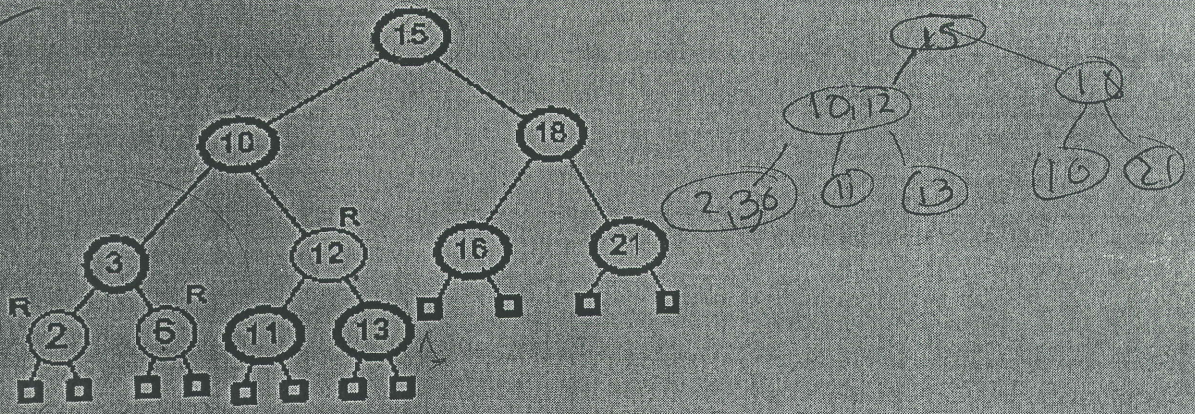

Figure 1