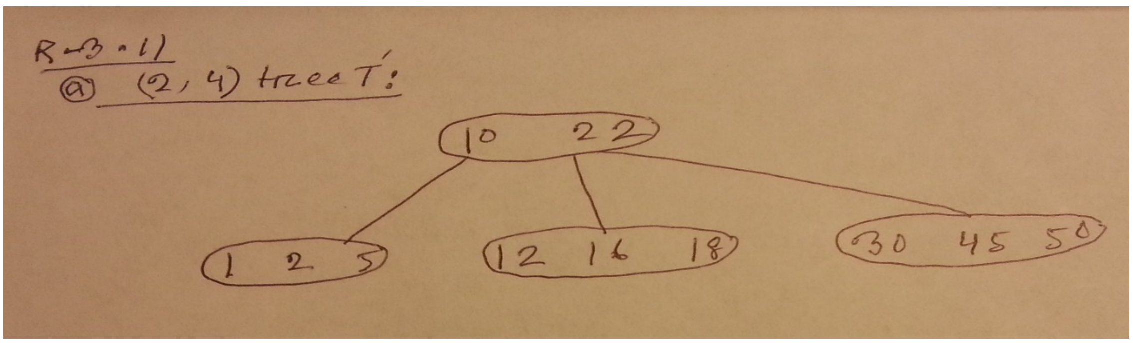
Name: Pheakdey Luk **Assignment 9**
ID: 986591

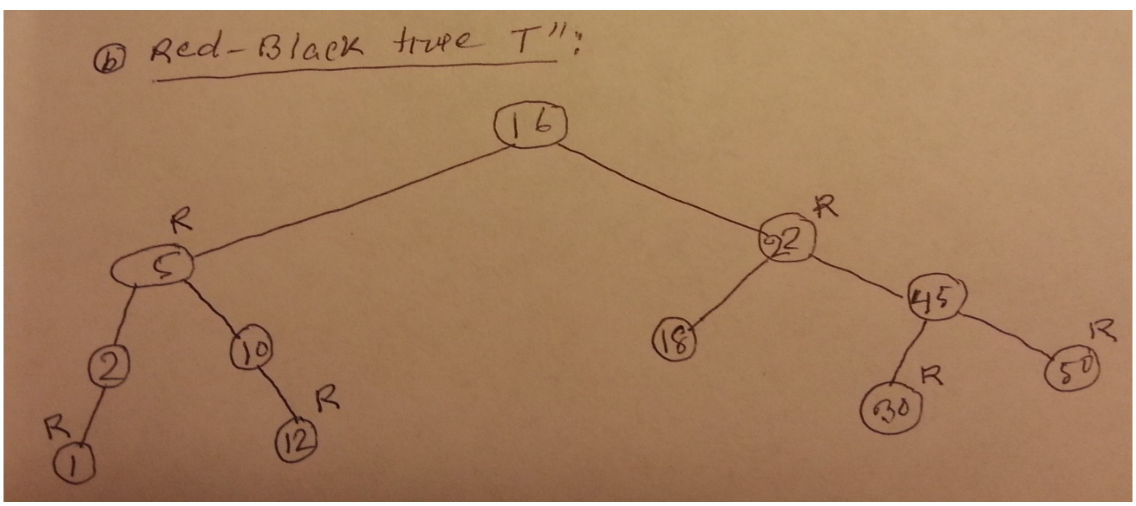
R-3.11 Consider the following sequence of keys:
(5, 16, 22, 45, 2, 10, 18, 30, 50, 12, 1)

Consider the insertion of items with this set of keys, in the order given, into: a. an initially empty (2,4) tree T'.
b. an initially empty red-black tree T''.

Draw T' and T'' after each insertion.

Answer:





R-3.14 For each of the following statements about red-black trees, determine whether it is true or false. If you think if it is true, provide a justification. If you think it is false, give a counterexample.

a	False. Reason: To be a red-black root has to be black but there is no guarantee root of subtree will be black. It might be red or black.
b	True. Reason: if by another external node it is a black external nodes, because there's a rule that all external nodes are black. And also it can be red, because when we insert, or doing recolor, or restructure, the node will still be red.
c	False. Reason: Every red-black tree can become (2,4) tree, and the other way around. But we can make the tree with the same red black tree and produce a different (2,4) tree, even though it's result will be the same. So basically, it's not unique.
d	False. Reason: Every (2,4) tree can become red-black tree, and the other way around. But we can make the tree with the same (2,4) tree and produce a different red black tree, even though it's result will be the same. So basically, it's not unique.

a. a subtree of a red-black tree is itself a red-black tree.
b. the sibling of an external node is either external or it is red.
c. given a red-black tree T, there is an unique (2,4) tree T' associated with T. d. given a (2,4) tree T, there is an unique red-black tree T' associated with T.

Design a pseudo code algorithm **isValidAVL(T)** that decides whether or not a binary tree is a valid AVL tree. For this problem, we define valid to mean that the height of the left and right sub-trees of every node do not differ by more than one.

What is the time complexity of your algorithm?

Design an algorithm, **isPermutation(A,B)** that takes two sequences A and B and determines whether or not they are permutations of each other, i.e., they contain same elements but possibly occurring in a different order. Assume the elements in A and B cannot be sorted. **Hint**: A and B may contain duplicates. Same problem as in previous homework, but this time use a dictionary to solve the problem.

What is the worst case time complexity of your algorithm? Justify your answer.

C-3.10 Let D be an ordered dictionary with n items implemented by means of an AVL tree (or a Red-Black tree). Show how to implement the following operation on D in time O(log n + s), where s is the size of the iterator returned:

FindAllInRange(k1, k2):
Return an iterator of all the elements in D with key k such that k1 < k < k2.

Answer:

```
Algorithm findAllInRange(k1,k2)
    Input: key k1, k2
    Ouput: return iterator for all
the elements in D within the range of
k1 and k2
    T<- tree of D
    S<-findElements(T,T.root(),k1,k2)
    return S.iterator()
```

```
Algorithm findElements(T,p,k1,k2)
    Input: Tree T, position of a node
p, key k1, k2
    Output: Sequence S with all the
elements between the range of k1 and
k2 inclusive.
    S<-new Sequence
    k <- T.key(p)
    if k1 <= k ^ k <= k2 then

S.insertLast(D.findElement(k))

findElements(T,T.leftChild(p),k1,k2)

findElements(T,T.rightChild(p),k1,k2)
    else if k < k1 then
        return
findElements(T,T.leftChild(p),k1,k2)
    return S
```