Assignment 9

R.3.11) 5, 16, 22, 45, 2, 10, 18, 30, 50, 12, 13, 33

1) 5, 16, 22

2) 16 / 5 / 22 45

3) 16 / 2,5,10 / 18,22,45

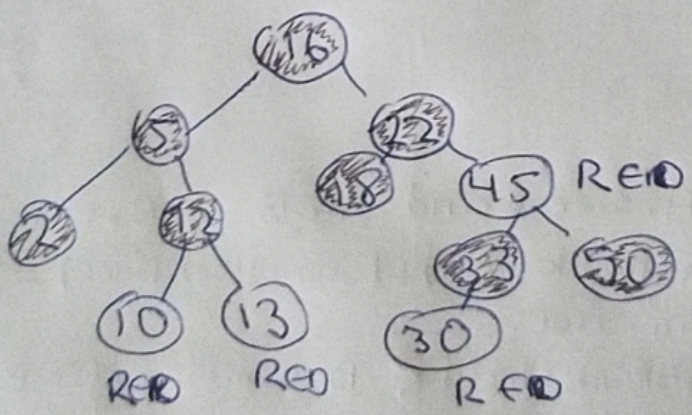
4) 16, 22 / 2,5,10 / 18 / 30,45,50

5) 5,16,22 / 2 / 10,12 / 18 / 30,45,50

5, 16, 22 / 2 / 10,12,13 / 18 / 30,45,50

16 / 5 / 22,45 / 2 / 10,12,13 / 18 / 30,33 / 50

- RED - BLACK



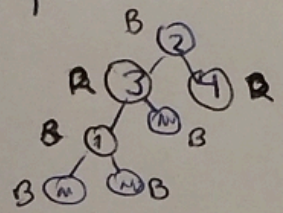
R.3_14.

- a Subtree of a red-black tree is itself a red-black tree.

true, because satisfy this
  - Every Node is either red or Black
  - Every leaf (NIL) is Black
  - if a node is red, then both its children are black
  - for each node, all simple path from the node to any descendant leaves contain the same number of black nodes

- the sibling of an external node is either external or it is red

False, because an external node can be either external or black



- Given a red-black tree T, there is a unique (2,4) Tree T' associated T

true, because (2,4) tree is a binary search tree that satisfy:
  - Every node has at least two children and at most four children.
  - All leaves are at the same level.
  - the keys in all left subtrees are less than the key in the root.
  - the keys in all right subtree are greather than the key in the root.

- Given (2,4) tree T, there is a unique red-black tree T' associated with T.

False, because there are multiple ways to color the nodes of T red or Black such that the resulting tree is a red-black tree,

```
Algorithm isValidAVL(ROOT)
    input is a node ROOT
    if root == null then
        return true
    leftHeight := getHeight(root.left)
    rightHeight := getHeight(root.right)
    if ( Abs (leftHeight - rightHeight) >1 ) then
        return false
    return isValidAVL(root.left) and isValidAVL(root.right)

Algorithm getHeight(root)
    input root is a node
    output Integer
    if root == null then
        return 0
    return 1 + Math.max(getHeight(root.left), getHeight(root.right))
```

```
Algorithm isPermutation(A, B)
Input A,B are Sequences.

1   if A.size != B.size then
        return false

1   D := new Dictionary

n   for ( i := 0 to A.size do
n       if D.findItem(A.elemAtRank(i)) == null then
n           D.insertItem(A.elemAtRank(i), 1)
        else
n           D.insertItem(A.elemAtRank(i), D.findValue(A.elemAtRank(i)) + 1)
n   for i := 0 to B.size do
n       count = D.findValue(B.elemAtRank(i))
n       if count = null or count = NO_SUCH_KEY then
            return false
n       D.insertItem(B.elemAtRank(i), count - 1)

1   return true.
```

```
Algorithm findAllInRange (k1, k2)
    Input k1, k2 are range
    S := New Sequence

    findHelper(T, k1, k2, T.root(), S)

    return S.

Algorithm findHelper(T, k1, k2, N, S)

    if T.isExternal(p) then
        e := p.element(); return
    if k1 > e then
    findHelper(T, k1, k2, T.leftChild(p), S)

    e := p.element()

    if k1 ≤ e ∧ e ≤ k2 then
        S.insertLast(e)
    if e > k2 then
    findHelper(T, k1, k2, T.rightChild(p), S)
```