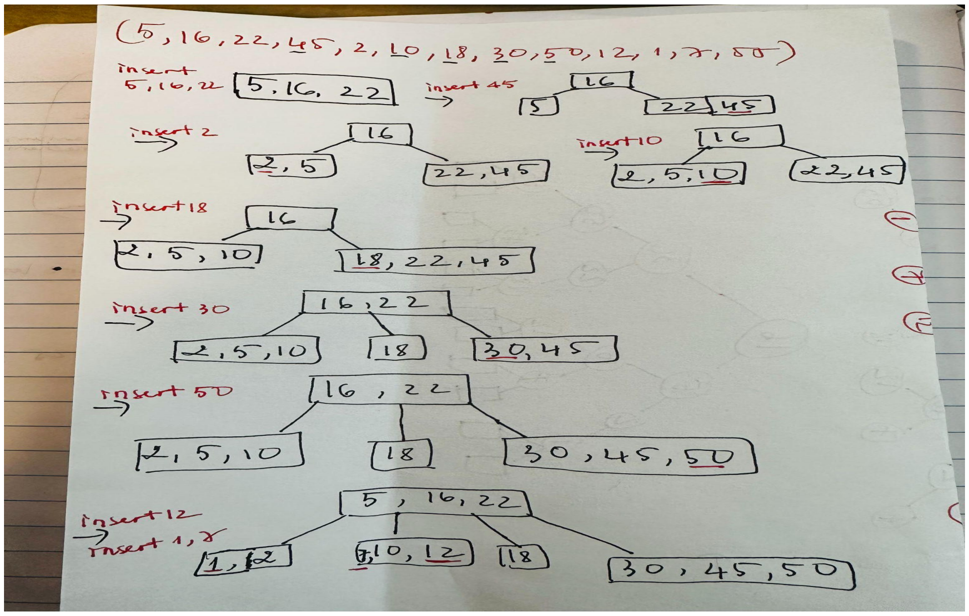
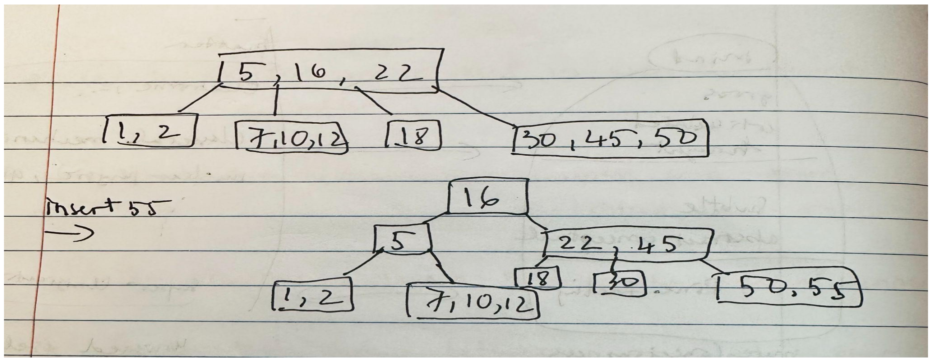
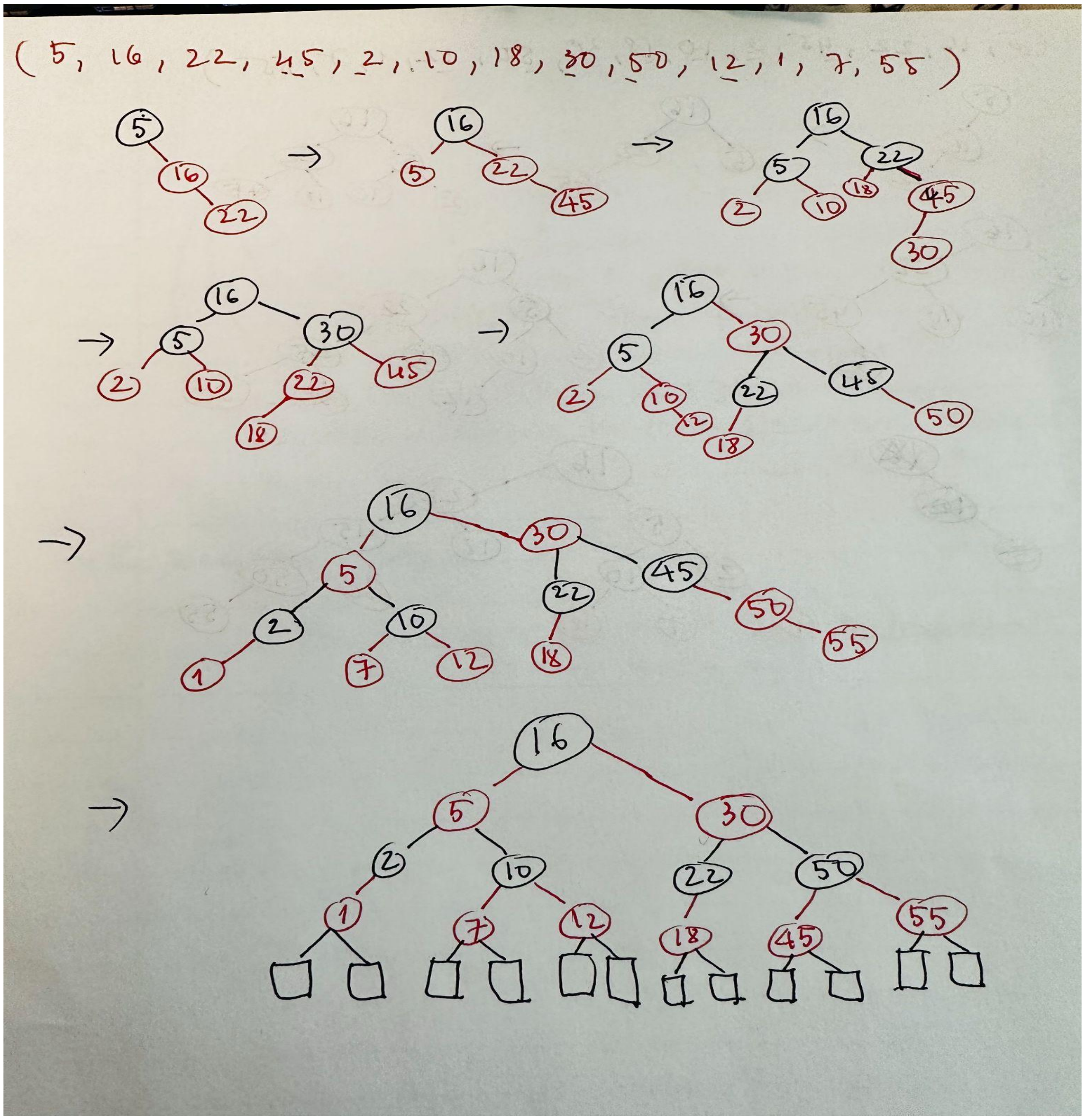
Assignment 14

R-3.11 Consider the following sequence of keys:
(5, 16, 22, 45, 2, 10, 18, 30, 50, 12, 1, 7, 55)
Consider the insertion of items with this set of keys, in the order given, into:
a. an initially empty (2,4) tree T'.Draw T' after each insertion.





b. an initially empty red-black tree T".Draw T" after each insertion.



R-3.14 For each of the following statements about red-black trees, determine whether it is true or false. If you think it is true, provide a justification. If you think it is false, give a counterexample.
a. a subtree of a red-black tree is itself a red-black tree.
False
Counterexample: There is a Red-Black Tree with the root is Black and left child and right child both are red. Therefore, there are 2 left and right subtrees with their roots are red, those subtrees are not the Red-Black Tree.

b. the sibling of an external node is either external or it is red.
True. If an external node had a black internal sibling, then these two siblings would not have the same black depth, which would contradict the property of red-black trees that all leaves must have the same black depth.

c. given a red-black tree T, there is an unique (2,4) tree T' associated with T.
True. Merge each black node with its red children (either 0 or 1 or 2 red children) to form either a 2-node or a 3-node or a 4-node, respectively.

d. given a (2,4) tree T, there is an unique red-black tree T' associated with T.
False. Each 3-node can be represented in either of two distinct ways. It becomes a black node with one red child, which can be either the left child or the right child.

1. Design a pseudo-code 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 the same elements but possibly occurring in a different order. Hint: A and B may contain duplicates. Same problem as in previous homework, but this time use a dictionary to solve the problem.

```
Algorithm isPermutation(A, B)
  if length of A is not equal to length of B
    return false
  create a dictionary D
  for each element x in A
    if x is not in D
      add x as a key to D with value 1
    else
      increment the value of x in D by 1
  for each element y in B
      if y is not in D
          return false
      else
          decrement the value of y in D by 1
      if the value of y in D is less than 0
          return false
  return true
```

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

The worst-case time complexity of the algorithm is O(n), where n is the length of the input sequences A and B.
The algorithm has two for loops, one for each sequence. The first for loop adds elements from sequence A to the dictionary D,
and the second for loop decrements the count of elements from sequence B.
Since each element is processed once in each for loop, the total number of operations is proportional to the sum of the lengths of
the two sequences, which is 2n. Therefore, the time complexity is O(n).
Note that the time complexity of dictionary operations (such as adding or checking if an element is in a dictionary,
or incrementing or decrementing the count of an element) is O(1) on average, but O(n) in the worst case when the dictionary has to resize itself.
However, since the dictionary is only used for counting elements, the size of the dictionary will be proportional to the
number of unique elements in the two sequences, which is at most n. Therefore, the time complexity of the dictionary operations will still
be O(1) on average, and O(n) in the worst case.

3. Design and solve this problem in four ways in JavaScript:
a. By sorting A and B
b. Using a Priority Queue
c. Using a Hash Table based Dictionary
d. Using a BST based Dictionary

4. Assume the elements in A and B cannot be sorted, i.e., there is no comparator. How would this restrict the way you would have to implement a solution to isPermutation(A,B), i.e., which of the above strategies could you use and which couldn't you use it?
If we use a Dictionary to check whether A and B are permuted(the algorithm above) , there is no restriction because we don't have to sort both A and B(no need for the comparator).

5. Which of the above strategies leaves the inputs A and B unchanged?

6. Are any of the approaches considered in-place?

7. Calculate the height of a Binary Tree. Implement your solution in the JavaScript file RBTree-HW.js that is provided. You are to do this both as a recursive function that traverses the tree and secondly using the Euler Tour template class (i.e., implement two different functions in JavaScript).

```
class Node {
  constructor(value, left = null, right = null) {
    this.value = value;
    this.left = left;
    this.right = right;
  }
}

function height(root) {
  if (!root) {
    return 0;
  }
  return 1 + Math.max(height(root.left), height(root.right));
}
```

8. Calculate the black height of each node of a Red-Black Tree. Implement your solution in the JavaScript file RBTree-HW.js that is provided.

```
const BT = require("./BinaryTree");

class Node{
    constructor (value, color, left=null, right = null) {
        this.value = value;
        this.color = color;
        this.left = left;
        this.right = right
    }
}

class RedBlackTree extends BT.BinaryTree {
    constructor () {
        super();
    }
    blackHeight(node) {
        if( !node ) return 0;
        if ( node.color === 'black')
            return 1 + Math.max(this.blackHeight(node.left),
this.blackHeight(node.right))
        else if (node.color === 'red')
            return  Math.min(this.blackHeight(node.left),
this.blackHeight(node.right))
    }
}

let rbTree = new RedBlackTree();
rbTree._root = new Node(16,'black')
rbTree._root.left = new Node(5, 'black');
rbTree._root.right = new Node(30, 'red');
rbTree._root.left.left = new Node(2, 'red');
rbTree._root.left.right = new Node(10, 'red');
rbTree._root.right.left = new Node(22, 'black');
rbTree._root.right.right = new Node(45, 'black');
rbTree._root.right.right.right = new Node(50, 'red');

console.log("The node value: ", rbTree._root.right.value + " the black
height: ", rbTree.blackHeight(rbTree._root.right));
```

You are to do this
both as a recursive function that traverses the tree and secondly using the Euler Tour template class (again two different functions). There are two methods on a Red-Black tree to determine the color of a node, i.e., T.isRed(p) and T.isBlack(p). The black height for each node corresponds to the height of that key in a 2-4

Tree. The definition of the black-height of a node p, denoted bh(p), is the number of black nodes from p to every external node in the subtree rooted at p, but not including node p. See the lecture notes for more details and examples.