1. Height-biased Leftist trees.

Consider the following height-biased max leftist tree. Show the modified tree under each of the following operations. (Note: The two operations (a) and (b) are independent. Each of them starts from the given tree.)

```
14
 /\ 
12 10
 /\ /\ 
8 4 6 9
 / / 
2 5
```

a. Perform a put (7). Label each node of the resulting tree with its $s$-value ($\text{shortest}(x)$).
b. Perform a RemoveMax () operation on the given tree above. Show each step.

2. Tournament trees.

A k-tournament is a tree in which the winner of each match moves up to occupy also the parent position. In a pointer-based implementation, we have for each cell, pointers left [.] and right [.] to left and right children, and a value [.] Design an efficient algorithm for merging two k-tournament trees into a (k+1)-tournament tree. Justify your algorithm’s time complexity.

3. Bin-packing problem

In contrast to the best-fit (BF) bin-packing strategy, the worst-fit (WF) strategy puts each object into the current bin that has the maximum residual capacity. If the object does not fit in any bin, a new bin is started. Write a Java program to implement the WF strategy. Refer to program code FirstFit.java in the applications folder.
/** worst fit bin packing */

package applications;

import dataStructures.*;

public class WorstFit {
    // top-level nested class
    public static class Bin implements Playable {
        // data members
        int unusedCapacity;

        // constructor
        public Bin(int theUnusedCapacity) {
            unusedCapacity = theUnusedCapacity;
        }

        // method of Playable
        public boolean winnerOf(Playable theBin) {
            // define for a max winner tree
            if (unusedCapacity >= ((Bin) theBin).unusedCapacity)
                return true;
            else
                return false;
        }
    }

    // output worst fit packing into bins of size binCapacity
    // *
    // * @param objectSize[1:objectSize.length-1]
    // * are the object sizes
    // */

    public static void WorstFitPack
        (int[] objectSize, int binCapacity) {
        // your code here, please output the results,
        // showing which objects are in each bin
    }

    //your help code here

    /** test program */
    public static void main(String[] args) {
        int[] objectSize = { 0, 4, 3, 6, 2, 7, 5, 9, 1 };
        // output the packing
        WorstFitPack(objectSize, 10);
    }
}
Hint: Unlike the BestFit strategy, the WorstFit strategy does not use a Binary Search Tree. Consider the code of FirstFit.java, which uses tournament tree to find the first bin with usable residual capacity. You can use a similar (but not identical) strategy to find the bin with the maximum residual capacity for the WorstFit strategy.

You MAY call any methods of class ExtendedCWTree. You may define any helper class.

a) Write Java code for the method

\[
\text{worstFitPack}(\text{int} \ [\text{objectSize}, \text{int} \ \text{binCapacity})
\]

b) Determine and justify your time complexity.