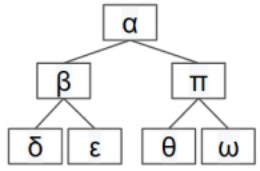
CS 61B Spring 2018

Heaps and Trees

Exam Prep 10: March 19, 2018

1 An Operational Understanding (Spring 2017 MT2 Q2)

Consider the tree on the left where greek letters represent numerical values. In the boxes to the right, shade all values that might match the text. Assume all values are unique. For BSTs, assume left items are less than.



	□α	□β	□π	□δ	□ε	□θ	□ω	
MinHeap, largest item	□α	□β	□π	□δ	□ε	□θ	□ω	
MinHeap, smallest item	□α	□β	□π	□δ	□ε	□θ	□ω	
BST, largest item	□α	□β	□π	□δ	□ε	□θ	□ω	
BST, smallest item	□α	□β	□π	□δ	□ε	□θ	□ω	
MinHeap, median item	□α	□β	□π	□δ	□ε	□θ	□ω	
BST, median item	□α	□β	□π	□δ	□ε	□θ	□ω	
MinHeap, new root after deleteMin	□α	□β	□π	□δ	□ε	□θ	□ω	
BST, new root after Hibbard[1] deletion of α	□α	□β	□π	□δ	□ε	□θ	□ω	
MinHeap, root after inserting new item φ	□α	□β	□π	□δ	□ε	□θ	□ω	□φ
BST, root after inserting new item φ	□α	□β	□π	□δ	□ε	□θ	□ω	□φ

hibbard deletion means swapping the node to be deleted with its min right node

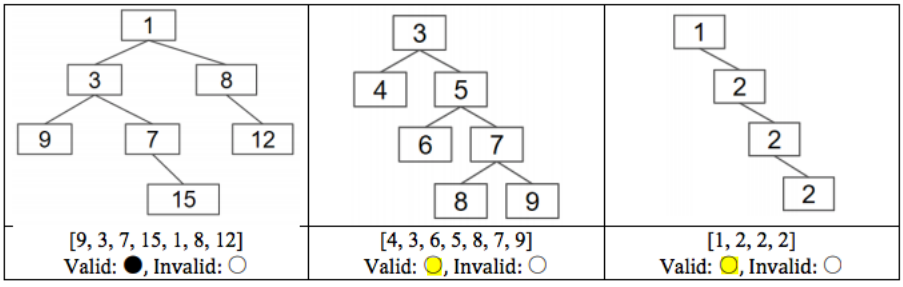
but here it seems that swapping with its max left node is also considered hibbard

2 Xelha Trees (Spring 2017 MT2)

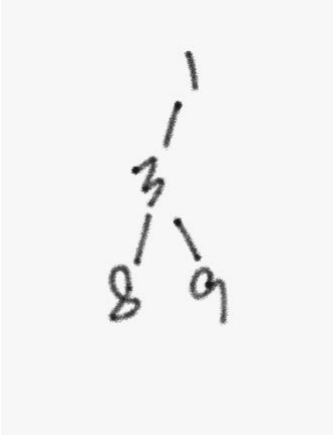
Given a list of numbers X, a XelhaTree for that list obeys the following:

1. The XelhaTree has the min-heap property (i.e. every value is less than or equal to its children).
2. An inorder traversal of the XelhaTree visits the nodes in the same order as the list. which is level order

(a) Which of the following are valid XelhaTrees for the given sequences? The first is done for you.



(b) Draw a valid XelhaTree corresponding to the sequence [8, 3, 9, 1].



3 Verifying Xelha Trees (Spring 2017 Final Q10)

Write a function `validXelhaTree` which takes an `IntTree` and a `List` and returns true if the `IntTree` is a `XelhaTree` for the list. You may not need all lines. A `XelhaTree` is valid if it obeys the min heap property, and if an in-order traversal of the `XelhaTree` yields the list of items passed to `createXelhaTree` (in the same order). One line if statements with on the same line are fine. You may not need all the blanks. Assume there are no duplicates.

```
public class TestXelhaTree {
    public static class IntTree {
        public int item;
        public IntTree left, right;
    }

    public static IntTree createXelhaTree(List<Integer> x) { ... }

    /** If x is null, returns largest possible integer 2147483647 */
    private static int getItem(IntTree x) {
        if (x == null) { return Integer.MAX_VALUE; }
        return x.item;
    }

    public static boolean isAHeap(IntTree xt) {
        // Bug1: the input xt may be null
        if (xt == null) {
            return true;
        }
        IntTree[] toCheck = {xt.left, xt.right};
        for (IntTree t : toCheck) {
            if (t != null && (!isAHeap(t) || getItem(xt) > getItem(t))) {
                return false;
            }
        }
        return true;                                                    eeValues) {
    }

    3 usages
    public static void getTreeValues(IntTree xt, List<Integer> treeValues) {
        if (xt == null) {
            return;
        }
        getTreeValues(xt.left, treeValues);
        treeValues.add(getItem(xt));
        getTreeValues(xt.right, treeValues);                            > vals) {
    }

    no usages
    public static boolean validXelhaTree(IntTree xt, List<Integer> vals) {
        List<Integer> treeValues = new ArrayList<Integer>();
        /* getTreeValues adds all values in xt to treeValues */
        getTreeValues(xt, treeValues);
        return isAHeap(xt) && treeValues.equals(vals);                   }
    }
}
```

4 Reconstructing Trees from Traversals

Given two lists of integers, where one represents the preorder traversal of a binary tree, and the other represents the inorder traversal of a binary tree, come up with a high-level implementation to reconstruct a binary tree given this information. In other words, you are coming up with an implementation for the following method:

```
public IntTree constructTree(int[] preorder, int[] inorder)
```

You may assume that all the elements in the lists are distinct.

The main idea is simple but the code can be complex:
1. we mark the 0th element of preorder as root
2. we traverse inorder to get the index of root
3. we devide inorder into two sublists using that index, the first is for left subtree, the second is for right subtree
4. we use recursion to construct IntTree left and IntTree right
5. we combine the two subtree and root together to become the whole tree