

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.

| | | | | | | | | | |
|--|--|---------------------------------------|---------------------------------------|---------------------------------------|---------------------------------------|---------------------------------------|---------------------------------------|---------------------------------------|---------------------------------------|
| | MinHeap, largest item | <input type="checkbox"/> α | <input type="checkbox"/> β | <input type="checkbox"/> π | <input checked="" type="checkbox"/> δ | <input checked="" type="checkbox"/> ε | <input checked="" type="checkbox"/> θ | <input checked="" type="checkbox"/> ω | |
| | MinHeap, smallest item | <input checked="" type="checkbox"/> α | <input type="checkbox"/> β | <input type="checkbox"/> π | <input type="checkbox"/> δ | <input type="checkbox"/> ε | <input type="checkbox"/> θ | <input type="checkbox"/> ω | |
| | BST, largest item | <input type="checkbox"/> α | <input type="checkbox"/> β | <input type="checkbox"/> π | <input type="checkbox"/> δ | <input type="checkbox"/> ε | <input type="checkbox"/> θ | <input checked="" type="checkbox"/> ω | |
| | BST, smallest item | <input type="checkbox"/> α | <input type="checkbox"/> β | <input type="checkbox"/> π | <input checked="" type="checkbox"/> δ | <input type="checkbox"/> ε | <input type="checkbox"/> θ | <input type="checkbox"/> ω | |
| | MinHeap, median item | <input type="checkbox"/> α | <input checked="" type="checkbox"/> β | <input checked="" type="checkbox"/> π | <input checked="" type="checkbox"/> δ | <input checked="" type="checkbox"/> ε | <input checked="" type="checkbox"/> θ | <input checked="" type="checkbox"/> ω | |
| | BST, median item | <input checked="" type="checkbox"/> α | <input type="checkbox"/> β | <input type="checkbox"/> π | <input type="checkbox"/> δ | <input type="checkbox"/> ε | <input type="checkbox"/> θ | <input type="checkbox"/> ω | |
| | MinHeap, new root after deleteMin | <input type="checkbox"/> α | <input checked="" type="checkbox"/> β | <input checked="" type="checkbox"/> π | <input type="checkbox"/> δ | <input type="checkbox"/> ε | <input type="checkbox"/> θ | <input type="checkbox"/> ω | |
| | BST, new root after Hibbard ¹ deletion of α | <input type="checkbox"/> α | <input type="checkbox"/> β | <input type="checkbox"/> π | <input type="checkbox"/> δ | <input checked="" type="checkbox"/> ε | <input checked="" type="checkbox"/> θ | <input type="checkbox"/> ω | |
| | MinHeap, root after inserting new item φ | <input checked="" type="checkbox"/> α | <input type="checkbox"/> β | <input type="checkbox"/> π | <input type="checkbox"/> δ | <input type="checkbox"/> ε | <input type="checkbox"/> θ | <input type="checkbox"/> ω | <input checked="" type="checkbox"/> φ |
| | BST, root after inserting new item φ | <input checked="" type="checkbox"/> α | <input type="checkbox"/> β | <input type="checkbox"/> π | <input type="checkbox"/> δ | <input type="checkbox"/> ε | <input type="checkbox"/> θ | <input type="checkbox"/> ω | <input type="checkbox"/> φ |

2 Xelha Trees (Spring 2017 MT2)

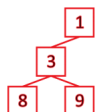
Given a list of numbers X, a XelhaTree for that list obeys the following:

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

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

| | | |
|---|---|--------------------------------------|
| | | |
| [9, 3, 7, 15, 1, 8, 12] Valid: ●, Invalid: ○ | [4, 3, 6, 5, 8, 7, 9] Valid: ●, Invalid: ○ | [1, 2, 2, 2] Valid: ●, Invalid: ○ |

(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.

```

1  public class TestXelhaTree {
2      public static class IntTree {
3          public int item;
4          public IntTree left, right;
5      }
6
7      public static IntTree createXelhaTree(List<Integer> x) { ... }
8
9      /** If x is null, returns largest possible integer 2147483647 */
10     private static int getItem(IntTree x) {
11         if (x == null) { return Integer.MAX_VALUE; }
12         return x.item;
13     }
14
15     public static boolean isAHeap(IntTree xt) {
16         if (xt == null) { return true; }
17         if (xt.item > getItem(xt.left)) { return false; }
18         if (xt.item > getItem(xt.right)) { return false; }
19         return isAHeap(xt.left) && isAHeap(xt.right);
20     }
21
22     public static void getTreeValues(IntTree xt, List<Integer> treeValues) {
23         if (xt == null) { return; }
24         getTreeValues(xt.left);
25         treeValues.add(xt.item);
26         getTreeValues(xt.right);
27     }
28
29     public static boolean validXelhaTree(IntTree xt, List<Integer> vals) {
30         List<Integer> treeValues = new ArrayList<Integer>();
31         /* getTreeValues adds all values in xt to treeValues */
32         getTreeValues(xt, treeValues);
33         return isAHeap(xt) && treeValues.equals(vals); }
34 }

```

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 to recognize that if we are given a list of elements from the preorder traversal, the first element must be the root, since preorder is visit root, left subtree, then right subtree. Since the inorder traversal is left subtree, visit root, then right subtree, and we know the location of the root because of the preorder, we can figure out from the inorder traversal list what all the elements of the left subtree are, same for the right subtree. Then because we know how many elements are in the left subtree, we know what sublist of the preorder list is for our left subtree. We can then build the left and right subtrees recursively.