Data Structures (CS 102), Professor Yap – Fall 2019

HOMEWORK

December 2, 2019

Homework 7: Due on Wed Nov 11, 2019

- Carefully read the Homework Policy in the class wiki. Pay attention to the rules for submitting programs in a zip file (not rar or other formats).
- This will be our last homework. We will provide Homework 8 later, but it will not be graded.
- This homework is larger than usual, and worth about two homeworks. So please start immediately even though you have a bit more time!

PART A: WRITTEN ASSIGNMENT

A.1 (3+3 Points)
Question R-9.10 and R-9.11 of Text (p.395). Possible positions the 3rd smallest key and the largest key in a min-heap.

A.2 (12 Points)
Question R-9.13 of Text (p.395). Simulation of in-place Heap Sort on the input sequence:

$$(2, 5, 16, 4, 10, 23, 39, 18, 26, 15).$$

How do you show your simulation? Instead of showing an array, always show the complete binary tree at each stage (at the end of each `upheap` or `downheap` (see p.377 code). *Note the heap size keeps changing, but always show the entire complete binary tree!*
Remark: THERE ARE TWO VERSIONS of `heapify`, i.e., the construction of a heap from an initial array. The "bottom up" version is described in Section 9.3.4 (p.380). But we ask you to use the simpler (but less efficient) method of growing the heap from left to right.

A.3 (6 Points) AVL simulation
Problem R-11.8 (inserting 52 into an AVL tree in Figure 11.13b, p.484).

A.4 (10 Points)
Write the java code with this header

```
int ht(Bnode2 u);
```

for computing the height of the BST at `u`. If `u=null`, then `ht(u)` returns `-1`. Please use recursion; you may write helper methods if you like.

A.5 (10 Points) Updating Balance information of AVL
In an AVL node `u`, we do not store the height of `u`. Instead, we store a "balance" information, which is defined to be

`bal(u) = ht(u.left)-ht(u.right)`.

Since it is AVL, `bal(u)` is either $-1, 0$ or $+1$. Suppose that the height of `u` just got incremented as a result of adding a new node. Describe in pseudo-code how the balance of `v=u.parent` should be updated. NOTE: you will have to implement this in the programming part.

A.6 (4+4+4 Points) Reconstruction of BST
(a) Here is the list of keys from post-order traversal of a BST:

$$2, 1, 3, 7, 10, 8, 5, 13, 15, 14, 12$$

Draw this binary search tree.
(b) Rotate the node with key 8. Draw the resulting BST.
(c) List the keys in a post-order traversal of (b).

A.7 (15 + 15 Points – programming and written part)
Given a singly-linked list we want to split off the list into two lists of roughly the same length (i.e., the lengths of the two lists differ by at most one). For example, if the list is initially $(3, 1, 4, 1, 5, 9)$, then after the split, the two lists will be $(3, 4, 5)$ and $(1, 1, 9)$. As usual, a list may be represented by its head node. Write the pseudo-code for the following a java method with this header

```
Node split(Node u);
```

The argument `u` represents the input list. After the split, `u` represents one of the two lists, and the other lists is returned.
NOTE: DO NOT write actual Java code (though you can put snippets of Java code when useful). Your pseudo-code must be implemented in the programming part of this homework.

PART B: PROGRAMMING (70 Points)

You have three programs to write: `Split.java`, `MyBST.java` and `AVL.java`. The latter two are based on the provided `BST.java`.

B.1 (20 Points)// Implement your `split` method in the provided code `src/Split.java`. The main method has already been provided for testing.

B.2 (100 Points)// We have provided a file `src/MyBST.java` to implement an subclass extension of the `BST` class (used in the previous homework). Recall the "parenthesized BST rep(resentation)" from the previous homework. Note: if a left or right child is null, just ignore it; DO NOT output empty pairs like `"()"` or `"[]"`. In `MyBST` we gave the various methods you need to implement.

```
String myString(Bnode2 u);
     E.g., myString(u) might return "5 (2) [7 [9]]"
Bnode2 fromString(String str);
     E.g., fromString("7 [9])") returns a BST rooted at 7
        with rightchild 9.
void storeBST(String ofilename) throws Exception;
     E.g., storeBST("bst.txt") will create a file "bst.txt"
        to store the parenthesized BST of the root
Bnode2 readBST(String ifilename) throws Exception;
     E.g., readBST("bst.txt") will read from file "bst.txt"
        and return the BST stored there
void showLevels();
     E.g., showLevel() will print the keys of the root BST,
        with each output line representing a level.
```

(a) The methods `myString(u)` and `fromString(str)` are inverses of each other: they inter-convert between the parenthesized BST rep and a BST.

(b) Similarly, `storeBST(ofile)` and `readBST(ifile)` are inverses of each other. They do the same thing, except through the intermediary of a file. (E.g., my graders test our own BSTs by using `storeBST(ofile)` to from a file.)

(c) Finally, `showLevels()` will list the keys of a BST level by level.

See our sample output file `MyBST-output.txt`.

B.3 (100 Points)// We have provided a file `src/AVL.java` which is a subclass extension of the `BST` class. You only need to implement the `add` method; The outline is provided. Note that the `Bnode2` class has a new member to store `balance` information. You basically have to implement one method: `rebalanceAdd()`. Or course, you need also `rotate(u)`.

DO NOT IMPLEMENT the `remove` or `rebalanceRemove()` methods.