CSE250, Spring 2022 "Virtual Assignment 7" Not Graded

Lectures and Reading:

The **Second Prelim Exam** will be on **Friday, April 29**. It will cover Chapters 12, 13, 15, and 16 (**not** 18), and focus on assignments through that date—*including this virtual one.*

Reading: This week will cover Chapter 18 on heaps, and maybe begin hash tables on Wednesday. That is to say, the order of coverage for the rest of term will be backward from the end: Chapter 22 on hash tables, then Chapter 21 on balanced binary trees, and finally as time permits, material from Chapter 19 on B-trees and maybe 20 on quadtrees and other spatial trees. (The latter two chapters will be over-the-horizon of assignments anyway, since the last weeks will focus on the final project. After Prelim II, there will be one more written assignment on examples of heaps and hash tables and balanced trees that will be graded for real, running in parallel time with the final course project.)

——— *Not a Graded Assignment, key will be released on Wednesday evening* ———

(1) Draw the binary search tree that results from inserting the words of this sentence in the order given, allowing duplicate keys. Use alphabetical order of lowercased words with the lower words at left. Then show the results of deleting all three occurrences of the word "the", one at a time. (It is OK to use either the inorder successor or predecessor for deletion, and putting an equal key left or right, but please show each step separately on the relevant part of the tree—you do not have to re-draw the whole tree each time. A virtual $12 + 9 = 21$ pts.)

(2) For each task below labeled 1.–8., say which of the following best describes its running time:

(a) Guaranteed $O(1)$ time.

(b) Amortized $O(1)$ time.

(c) Usually $O(1)$ time.

(d) Guaranteed $o(n)$ time.

(e) Usually $o(n)$ time.

(f) Guaranteed $O(n)$ time.

In all cases n denotes the number of items currently in the underlying data structure, and any other parameters are stated. Note that the answer "$o(n)$ time" can convey either $O(\log n)$ time or $O(\sqrt{n})$ time, or other *sub-linear* times. The variable `vec` stands for a vector, `dlist` for a doubly-linked list (not necessarily sorted), `ba` for a "`BALBOADLL`" data structure (assuming it has just been newly created or refreshed, no deletions), `arr` for an `ArrayBuffer` (or the `SortedArray` class in the ISR repository, same answers), `bst` for a BST—i.e. a basic binary search tree without balance guarantee, `itr` for an iterator of the appropriate kind, `comp` for a typical comparator function, and `item` for a typical item in the data structure. *Justifications* are not required, but might help for partial credit. The tasks are:

1. For a BST iterator `itr`, the call `itr.next()`

2. `dlist.pushRear(item);` (equivalently, `dlist.insert(item,dlist.end)`)

3. `dlist.remove(itr);`

4. `arr.remove(itr);` where `itr` points in the middle of `arr`

5. For a `BALBOADLL` iterator `itr`, the call `itr.next()`

6. `bst.find(item)`

7. Preorder traversal of a BST.

8. For a BALBOADLL object `ba` and index $j < n$, the call `ba(j)`. This supposes that indexing is simulated by running down the linked list and adding up the `length` of each constituent array until the sum goes over `j`, in which case the current array `arr` has the jth overall element. If `s` was the sum before adding the size of that array, so `s <= j`, then return `arr(j - s)`.

(A virtual $8 \times 3 = 24$ pts.)

(3) Show how a binary search tree `bst` can implement indexing `bst(j)` in $O(h)$ time, where h is the height of the tree. In particular, if the tree is *balanced*—which I specify to mean $h \leq 2\log_2 n$ for a tree of n nodes overall—then this means indexing is in $O(\log n)$ time. The needed wrinkle is that each `Node` has an extra field `mySize: Int` which maintains the number of nodes in the subtree rooted at that node. So, for instance, the leaves are exactly those nodes with `mySize == 1`, and that might be a quicker check than both `left` and `right` being `endSentinel` (or being `null` in the text). And of course `root.mySize` equals n itself—this would supersede the BST class needing to maintain a `_size` field.

The particular case $j = (n-1)/2$ when n is odd means that the middle element of such a tree can be found in $O(\log n)$ time. Sketch your algorithm in Scala-like pseudocode and explain why it has the desired time. As usual, you may presume that reading and comparing fields of nodes are $O(1)$-time operations unto themselves. (A virtual 27 pts., making 72 on the virtual set.)