CS61B, Fall 2009 **Test #2 Solutions** **P. N. Hilfinger**

1. [8 points] Please give short answers to the following, being sure to answer all parts of each question. Time estimates refer to asymptotic bounds ($O(\cdots)$, $\Omega(\cdots)$, $\Theta(\cdots)$). Always give the simplest bounds possible ($O(f(x))$ is "simpler" than $O(f(x) + g(x))$ and $O(Kf(x))$).

a. Consider a QuadTree, as represented in Project 2, in which each leaf node happens to contain at most one point. What is the worst-case time for determining whether a point with coordinates (x, y) is in the tree? Your answer may depend on the number of points in the QuadTree or the parameters to the QuadTree's constructor (or both).

> *Since all leaf nodes contain no more than one point, the determining factor is the depth of the tree. For a tree with width W and height H (determined by subtracting the upper-right and lower-left x and y coordinates, respectively) and a delta of δ, there are at most W/δ minimum-sized boxes along the x axis and H/δ along the y, so the total number of times the space can be divided (hence the maximum height of the tree is $\Theta(\lg \min(W, H)/\delta)$.*

b. A certain application receives and stores many timestamped records from a large number of satellites. Because the satellites are constantly moving and sometimes have to relay their data in various ways, the transit times of these messages can vary by several seconds and therefore the data records may get stored out of sequence. Occasionally, therefore, the data are sorted. A new programmer is horrified to discover that whoever wrote this application used insertion sort, an $O(N^2)$ algorithm. He decides to improve matters by switching to quicksort. However, this turns out to slow down the sorting process. Why? (Be specific; use the information given in the problem.)

> *From the problem statement, we expect that records are all close to their correct positions, so that insertion sort approaches being $\Theta(N)$. Quicksort, on the other hand, is $\Omega(N \lg N)$ (that is, $N \lg N$ even in the best case), and has somewhat worse constant factors, so we should expect it to be slower.*

c. We wish to find all the values in a binary search tree with N nodes that are between two given values L and U, *assuming that the value at the root of the tree is itself between L and U.* If there are K such values in the tree, how long does it take to find them all, and would this answer change if the root's value need not be between L and U?

   The answer turns out to be $O(K + H)$ in either case, where H is the height, because it is possible for the first of last node to be at depth H in the tree, and ones in between get delivered symmetric order in time $O(K)$. At worst, H can be N.

d. A binary search tree with N nodes has parent links—that is, from any node, one can reach either child of a node or the node's parent in one step. Show a worst-case situation for finding the node with next-larger value starting at an arbitrary node in the tree. Also, what is the worst-case time?

   Worst-case time required is $\Theta(N)$, because of this case:

2. [1 point] From Earth, what star has the largest known proper motion?

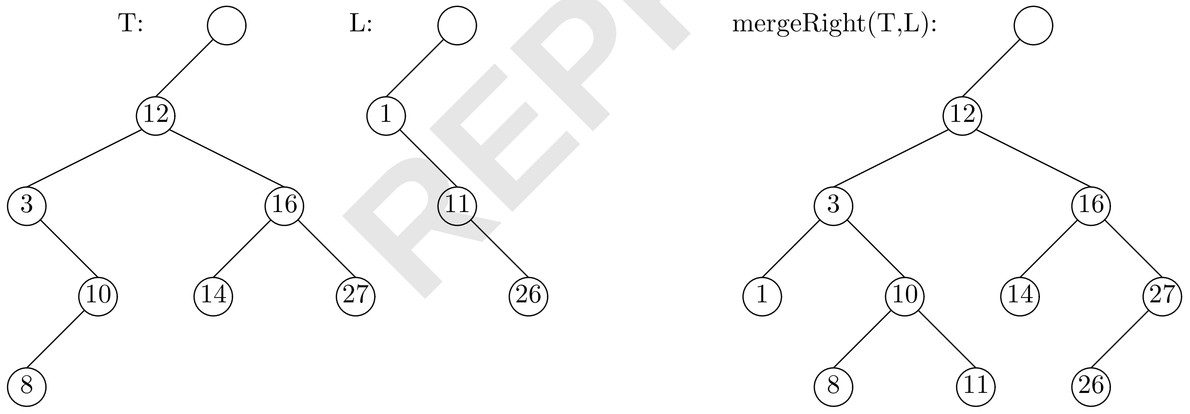
Well, one could say "the Sun," but assuming this is not a trick question, one might rather say "Barnard's star."

3. [8 points] Given the following class definition:

```
class IntTree {
    public IntTree (int data, IntTree left, IntTree right) {
        this.data = data; this.left = left; this.right = right;
    }

    public final int data;
    public IntTree left, right;
}
```

we want to define a destructive `mergeRight` routine that combines the values in two search trees. The trees here are binary search trees with a sentinel node (whose `data` field is irrelevant) at their root. The sentinel's left child is the tree containing the data in each case. One of the trees is right-leaning—essentially an ordered list. For example, if `T` and `L` contain the trees below the resulting tree is shown on the right. Fill in the method on the next page to fulfill its comment. Feel free to add any auxiliary methods you need. *IMPORTANT WARNING:* To get the running time specified in the comment, you cannot simply call the standard insertion method on `T` for each value in `L`. You need to take advantage of the fact that `L` is a right-leaning binary search tree.



```
/** Assuming that T and L are binary search trees each with a single
 *  sentinel tree node, and that all left children in L aside from
 *  the sentinel are empty (L is "right-leaning"), returns (the
 *  sentinel of) a binary search tree containing the original elements
 *  of T and L.  The operation is destructive, and creates no
 *  new nodes.  In the worst case, it takes time linear in the
 *  total number of items in T and L. */
public static IntTree mergeRight (IntTree T, IntTree L) {
    T.left = mergeRight2(T.left, Integer.MAX_VALUE, L);
}

/** Assuming T is a BST, and L is the sentinel of a
 *  right-leaning BST, return the result of inserting all items of L
 *  that are <= NEXT in T, removing them from L. */
private static IntTree mergeRight2(IntTree T, int next, IntTree L) {
    if (L.left == null)
        /* Done */
        return T;
    if (T == null) {
        if (L.left.data <= next) {
           IntList p = L.left;
           L.left = L.left.right;
           p.right = mergeRight2 (nil, next, L);
           return p;
        }
    } else {
        if (L.left.data <= T.data)
            T.left = mergeRight2(T.left, T.data, L);
        if (L.left != null && L.left.data > T.data)
            T.right = mergeRight2(T.right, next, L);
        return T;
    }
}
```

4. [4 points] Give short answers to the following questions. As before, time estimates refer to asymptotic bounds ($O(\cdots)$, $\Omega(\cdots)$, $\Theta(\cdots)$). Always give the simplest bounds possible ($O(f(x))$ is "simpler" than $O(f(x)+g(x))$ and $O(Kf(x))$).

a. A certain hash table on strings uses a very fast hash function: it simply takes the first and last characters in a string and interprets them as a two-digit number—specifically (assuming that all characters are lower-case letters) a two-digit base-26 number, so that "algebra" hashes to 0, "add" hashes to 3, "bad" to 29, and so forth. Each hash bucket is represented by a linked list. *Assuming that keys are distributed evenly by this hash function,* what is the worst-case time (including the time required to compute the hash function) for inserting one new key, as a function of N, the maximum number of keys in the table?

   $\Theta(N)$, *since the hash function will divide the data evenly into only* $26^2 = 676$ *buckets, so the time to check the selected bucket grows as* $N/676$.

b. How does your answer to (a) change if we change the hash function so that it considers up to the first $\log_{26} N$ characters (or the length of the key, whichever is smaller), keeping other assumptions the same? (Ignore the time to compute $\log_{26} N$.)

   This is enough characters so that there will be at most a constant number of keys per bucket, since they are assumed to be divided evenly. On the other hand, it takes $O(\lg N)$ *time to compute the hash function, so overall* $O(1+\lg N) = O(\lg N)$.

c. A heap data structure containing integers is represented as an array. Assuming that the heap is initially empty, show that for any N, there is a sequence of N values that can be added to the heap in total time $O(N)$.

d. For Java `int` variable x, what is `(~x) - ((-1) ^ x)`?

0, since -1 is all 1-bits, and xoring 1 with any bit complements that bit.