READ THIS NOW!

- Print your name in the space provided below.
- There are 4 short-answer questions, priced as marked. The maximum score is 50.
- Most questions require analyzing a scenario, drawing a conclusion, and justifying that conclusion. Credit will be awarded for making statements that are both true and relevant to the question. Do not feel obligated to fill all the available space when answering the questions.
- This examination is closed book and closed notes.
- No calculators, cell phones, or other computing devices may be used. The use of any such device will be interpreted as an indication that you are finished with the test and your test form will be collected immediately.
- Until solutions are posted, or gone over in class, you may not discuss this examination with any student who has not taken it.
- Failure to adhere to any of these restrictions is an Honor Code violation.
- When you have finished, sign the pledge at the bottom of this page and turn in the test.

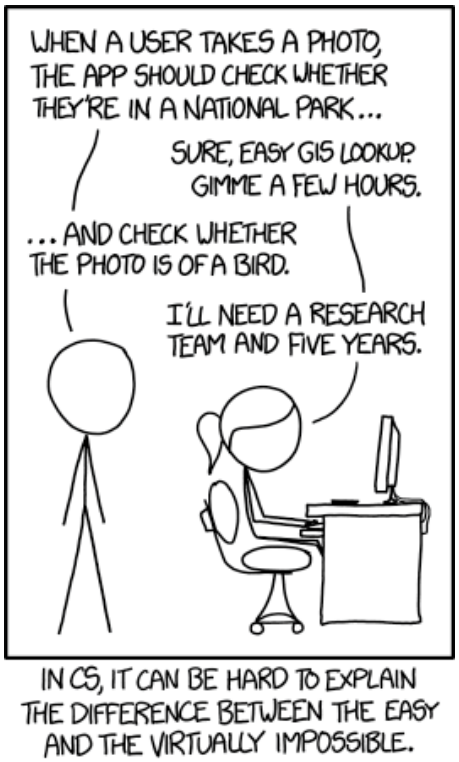
Name (Last, First) Solution

printed

Pledge: On my honor, I have neither given nor received unauthorized aid on this examination.

_____

signed



In the 60s, Marvin Minsky assigned a couple of undergrads to spend the summer programming a computer to use a camera to identify objects in a scene.

He figured they'd have the problem solved by the end of the summer.

Half a century later, we're still working on it

xkcd.com

1. An AVL tree uses rotations to restructure part of the tree following some insertion operations. (This is also true for some deletion operations, but that's not relevant to the questions being asked here.)

Suppose that a new value is inserted to an AVL tree, and as a result a single rotation operation is performed. Suppose the insertion added a new node to the left subtree of a node ***N***, and the subsequent rotation was performed at that node, ***N***.

a) **[5 points]** Is the rotation performed before the new node is added to the tree, or after the new node is added to the tree?

After… until we've determined how the insertion of the new node affects the balance factors in its ancestor nodes (path back to the root), we don't have enough information to determine whether any rebalancing will be needed. (Technically, we could determine the new balance factors for the ancestor nodes before physically inserting the new leaf, but that would require keeping track of where the new leaf should wind up after any necessary rotations were performed; there's no gain in doing that.)

b) **[5 points]** What relevant fact must have been true about ***N***'s two subtrees immediately before the new node was inserted?

The left subtree of N must have had exactly one more level than the right subtree of N.

We know the insertion caused a rotation due to a bad balance factor at N, and the insertion was performed in the left subtree of N… therefore, the left subtree of N must have had two more levels (than the right subtree) after the insertion was performed. And, inserting a leaf cannot increase the number of levels in the subtree by more than 1.

c) **[5 points]** Continuing from the situation described in part **b)**, what relevant fact must have been true about ***N***'s two subtrees immediately after the new node was inserted, but before any rotations were performed?

The left subtree must have had exactly two more levels than the right subtree.

See comments for part b).

d) **[5 points]** Let's suppose that, before the insertion was performed, ***N*** was the right child of its parent node, ***P***. After the single rotation has been performed, the node ***N*** will have been shifted within the tree, and a different node will now be the right child of ***P***.

From ***P***'s perspective, how will the right subtree ***P*** had before the insertion compare to the right subtree it has after the insertion and rotation?

The key point is the entire reason we perform the rotations: after the insertion and rotation, P's right subtree will have exactly the same number of levels as it had before the insertion, so the rotation prevented the subtree from getting taller.

Lesser points:

- **P's right child after the rotation was the left child of N before the insertion**
- **P's right subtree contains one more node than before the insertion**

2. **[10 points]** In an AVL tree, rebalancing after an insertion never requires more than two single rotations (if we think of a double rotation as a sequence of two single rotations). Since the cost of a single rotation is Θ(1), the cost of rebalancing after an insertion only adds a small constant cost to the insertion (compared to a plain BST).

On the other hand, rebalancing after a deletion may require a single rotation at (essentially) every level, all the way to the root node of the tree. Even so, the cost of a deletion, including all the rebalancing, is still Θ(log N). Explain why.

The cost of all those single rotations cannot be more than a constant times the number of levels in the AVL tree.

An AVL tree with N nodes is guaranteed to have Θ(log N) levels.

Therefore, the total cost of the deletion is no worse than the cost of going to the bottom level, plus the cost of the physical deletion of a node plus the cost of backing out and restructuring, which is Θ(log N + 1 + log N) = Θ(log N).

A complete answer will explain the complexity result entirely.

---

3. **[10 points]** How does the worst-case cost of a single insertion to a splay tree compare to the worst-case cost of a single insertion to an AVL tree, if we (fairly) take into account the cost of the rotation operation(s) that would be required in each case? Be precise.

In an AVL tree, after inserting a new node, we never need more than two single rotations to rebalance. Since the number of levels for an AVL tree with N nodes is Θ(log N), the cost of the insertion is no worse than Θ(log N).

For a splay tree with N nodes, the number of levels is only guaranteed to be O(N). However, the splay tree will perform NO rotations after an insertion. So, if the found node is in the lowest level of the tree, the cost of the insertion, which involves NO rebalancing, could be as large as Θ(N).

A common error was to say the splay tree will perform rotations after an insertion (or a deletion).

4. [10 points] Suppose you are building an application that will involve performing a large number of searches on a huge collection of data records, stored in a file on disk. One type of search will involve a key that is a non-negative integer, an ID number. Each record is guaranteed to have a different value for that key. You have decided you will handle these searches by building an index in memory, and the index will use some sort of binary search tree for its physical structure. You have chosen a binary search tree, instead of a hash table, because the application will also involve a substantial number of insertions and deletions.

Research has convinced you that when your application is in use, searches will exhibit no temporal locality. What kind of binary search tree should you adopt for your application? Why?

If we expect no temporal locality when searching, then we expect that each record is just as likely as any other record to be searched for. In other words, if a huge number of searches were performed, we would expect each record to be searched for about the same number of times.

In that case, we should prefer an AVL tree to a splay tree, since the AVL tree will guarantee a Θ(log N) upper bound on the search cost for a randomly-chosen record, while the splay tree can yield search costs as high as Θ(N).

And, of course, we would prefer that to a plain BST, which doesn't even give a good bound on the number of levels.

A complete answer would not only explain why the AVL tree would be good, but why the obvious alternative would not be as good.