CS 112: Fall 2016

BST/AVL

Review

1. The worst case search time in a BST with n nodes is:

A. [✓ O(n)]

B. O(1)

C. O(n^2)

D. O(log n)

2. The best case search time in a BST with n nodes is:

A. O(n)

B. [✓ O(1)]

C. O(n^2)

D. O(log n)

3. A BST holds n integers. The running time to print all the items from the BST in reverse sorted order is:

A. O(1)

[✓ B. O(n)]

Flip the inorder traversal: right subtree, then root, then left subtree. Process same as inorder, sequence is different, so same running time as inorder traversal

C. O(n^2)

D. O(log n)

4. The following integers are added to a BST one at a time:

5, 10, 15, 20, 25, 30, 35

Which of the following SEQUENCE of inserts into the BST will result in a tree of height greater than 3? (At least 4 branches from root to farthest leaf):

A. 20, 10, 5, 15, 30, 25, 35

B. 25, 35, 30, 15, 10, 5, 20

[✓ C. 15, 5, 10, 35, 20, 25, 30]

D. None of the above

5. The following SEQUENCE of integers are added to an AVL tree one at a time:

5, 10, 15, 20, 25, 30, 35

What is the height of the resulting AVL tree (height is number of branches from root to farthest leaf):

[✓ A. 2]

B. 3

C. 4

D. 5

6. A BST starts out with n items. The items are then deleted from the BST, one at a time, until the BST is empty. What is the worst case running time of this process?

A. O(n)

B. O(nlogn)

[✓ C. O(n^2)]

D. O(log n)

7. An item is inserted into an AVL tree. Going back toward the root, a node X is encountered with balance factor '-' (equal high). Is there a situation in which this node could be unbalanced, and would need to be fixed with one or two rotations?

A. Yes

B. No ⬅

A node with initial balance factor equal high can, after ani insertion in its left or right subtree either stay that way, or become left high or right high, but never out of balance.

C. Depends

D. None of the above

8. A BST is to be built out of a sorted array of n items. What would be the running time of the fastest algorithm that can do this?

A. O(1)

B. O(log n)

[✓ C. O(n)]

Start with the midpoint, make it the root. This will take O(1) time since finding the midpoint is one arithmetic operation, And making a node is one unit time. Recursively build left and right subtrees using left and right halves of the array, taking up O(1) for each node.

D. O(n log n)