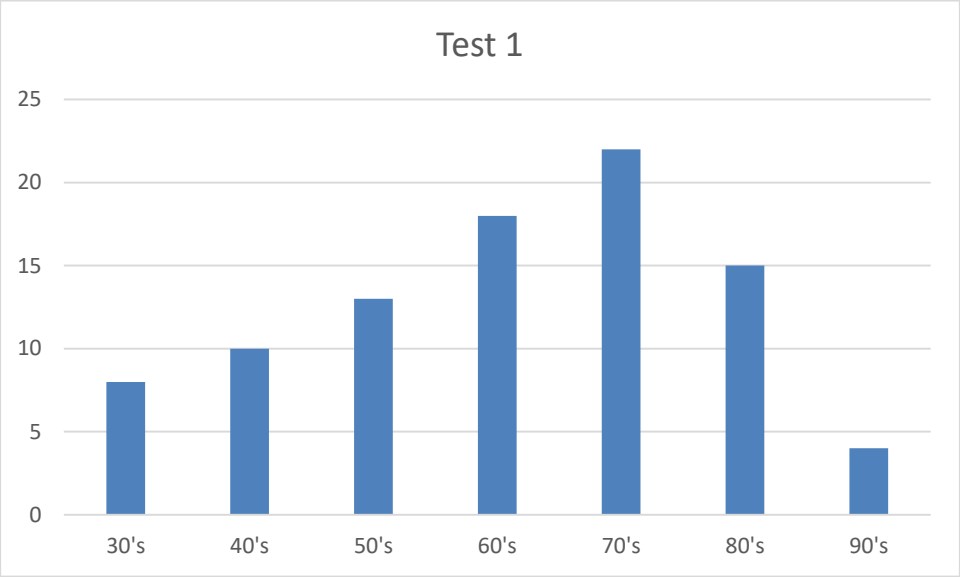


Average: 64.6%
Median: 68%

1. **[10 points]** Suppose we have a value, X that has been inserted into a BST that also contains numerous other values. Suppose we delete X from the structure, and then immediately reinsert X into the structure.

Could the cost of finding X before it was deleted be different than the cost of finding X after it has been reinserted? If yes, would there always be a difference, or only in certain cases, and could the cost increase or decrease or both? Explain.

Answer:

Key fact: Before it's been deleted, X could reside anywhere between the root node and the lowest leaf. After the deletion, when X is reinserted it will reside in a leaf node.

Simplistic answer: **the cost of searching for X will increase if X was originally not in a leaf node, and will be same if X was originally in a leaf node**

Better answer: **A key observation is that when X is reinserted, the insertion traversal will initially follow a path through the BST to the position formerly occupied by the node containing X.**

If X was reinserted at the same position it formerly occupied, then the cost of searching for X would be the same. But this can only occur if X was in a leaf node when it was removed from the tree.

Otherwise, X will proceed further down a branch when being reinserted, and so the cost of searching for X will now be greater than before X was removed.

There's no case in which the cost could be less.

Common issues with student responses:

- **Failing to address all three possibilities (cost is less, cost is same, cost is more). I ignored the omission of the middle case.**
- **Not clearly saying that, on reinsertion, X must be a leaf. That's a key point in the analysis.**

---

2. **[10 points]** Suppose we have a BST containing numerous values, including two values, X and Y, so that Y is the left child of X, and Y is in a leaf node.

`Could there be a value, Z, which was inserted into the BST after both X and Y were inserted, such that Y < Z < X?

If yes, explain where the node containing Z might be, in relation to the node containing X. If not, explain why not.

Answer:

Clever argument:

Suppose there is a value Z in the tree such that Y < Z < X, and Z was inserted after both X and Y, and Y is a leaf of X.

Now, consider performing an inorder traversal in the tree; such a traversal must reach the values in the order Y then Z then X (there may be other values between these, of course).

OTOH, an inorder traversal visits the left subtree of X, containing only Y, then X. So, the traversal cannot visit Z between Y and X unless Z is in the right subtree of X. But that contradicts the fact that Y is a leaf.

Therefore, there cannot be such a value in the tree.

You might wonder, where would Z have to be? The answer (clarified by the argument below) is that Z would have to be the right child of Y.

You might also wonder, what if Z was inserted after X and before Y, or before both X and Y? That's a different, and also interesting, question.

Convoluted argument:

Suppose that a value Z, lying between Y and X, was inserted into the BST after both X and Y were.

Where could Z wind up? Think about the path from the root to X. Z must either follow that same path, all the way to meet X, or Z must move from that path somewhere to the LEFT of that path. (Because if Z went right where X did not, say at some value V, then it must be that X < V but V < Z, a contradiction.)

Could Z actually head left before reaching X? Only if there's a value, V, along the path from the root to X such that V < X and V < Y (so X and Y both go right at V), but Z < V (so Z goes left at V). But that contradicts the hypothesis that Z is between X and Y.

If Z meets X, then Z must go LEFT at X and meet Y. Z would then go RIGHT from Y. But, that contradicts the fact that Y is a leaf. Therefore, cannot meet X.

Therefore, given that Y is a leaf, no such Z can exist.

Common issues with student responses:

- Ignoring the given fact that Y is a leaf. Read carefully. The question does not merely say Y was a leaf before Z was inserted.
- Assuming, without explanation, that when Z is inserted it must follow the path from the root down to X. That is true, but it needs to be argued.

3. [10 points] Haskell Hoo IV is attempting to implement a BST generic. Being concerned that his logic may be untrustworthy, he has written the private member function shown below. He plans to call it at the end of each insertion and deletion operation while he is testing his BST.

```
// Pre:      sroot is null or points to the root node of a BST
// Post:     the BST is unchanged
// Returns:  true iff the BST ordering property is NOT violated somewhere
//              in the tree
//
private boolean hasBSTProperty( BinaryNode sroot ) {

   if ( sroot == null ) return true;

   if ( sroot.left != null &&
        sroot.element.compareTo( sroot.left.element ) <= 0 )
       return false;

   if ( sroot.right != null &&
        sroot.element.compareTo( sroot.right.element ) >= 0 )
       return false;

   return hasBSTProperty( sroot.left ) && hasBSTProperty( sroot.right );
}
```

Could Hoo's function ever return true even though his tree did not have the BST ordering property?

Justify your answer; if yes, use a specific example of a BST to illustrate; if no, explain exactly why.

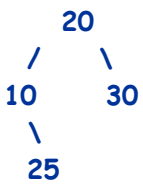
Answer:

One problem with Hoo's function is that it only does LOCAL tests. That is, it compares the value in a node only to the values in its child nodes. But, that's not sufficient.

The handling of duplicate values is also problematic, but far less significant.

So, when could Hoo's function return true? One case would be if the tree was either empty, or consisted only of a root node. Both of those cases are valid BSTs.

Another would be if the children of each node were, in fact, correctly related to the parent, but some child nodes were not correctly related to "grandparent" nodes:



Common issues with student responses:

- **Saying the function is actually correct. This fooled many of you*.**
- **Not giving either a specific example as a diagram, or a clear description of a specific example of a non-BST for which the function would return true.**

* I originally came across a variation of this in the solutions manual for a data structures textbook. In that version, the comparisons were done correctly, but they were purely local (parent to child). In fairness to the author of the textbook, solutions manuals are usually contracted out, and written by some overworked, underpaid graduate student.

Instructions for questions 4 and 5:

Suppose we are building a PR quadtree, with world boundaries 0 <= x <= 64 and 0 <= y <= 64, and bucket size 1.

4. **[10 points]** We begin by inserting the point A(20, 50), and then the point B(28, 54). How many splits will occur when B is inserted?

Answer:
The y-coordinates are close together, so let's start by just considering the horizontal splits:

Split 1: separates < 32 from > 32; A and B are on the left side of that
Split 2: separates < 16 from > 16; A and B are on the right side of that
Split 3: separates < 24 from > 24; and that separates A and B

Common issues with student responses:
- Not understanding PR quadtrees only store data in the leaf nodes.
- Not understanding how splitting works in a PR quadtree.

5. **[10 points]**
Next, we insert the following points:

C(10, 25) , D(50, 25) , E(25, 45) , F(53, 27)

How many leaf nodes will the PR quadtree contain now? Why?

Answer:
Since the bucket size is 1, the number of leaf nodes equals the number of data points that have been inserted.

That would be 6.

Common issues with student responses (aside from the issues in Q4):
- Not including the points A and B.
- Not giving a clear justification; all that needed to be said is said in my answer.
- Counting empty leaf nodes. I did not penalize for this if you got the correct answer (13 empty + 6 filled), but no implementation would ever create a node that served no purpose.