Week 4 feedback and quiz answers

(read only after passing the programming assignment and the quiz)

You've Got the Hang of It!

Congratulations on completing Week 4 of Data Structures: Measuring and Optimizing Performance! To complete this week's quiz requires understanding the differences between a tree and a binary search tree along with knowing its run time. If you struggled with this week's quiz, that's okay! Trees are complex, so here is a helpful hint:

1. The best way to understand trees are to do practice problems. There are many websites are there that gives sample technical interview questions regarding trees. It could be beneficial to look into these websites and get to practicing!

Also, if you found yourself struggling with this week's assignment, that's okay! Before going on to Week 5, think about the big picture and what you are trying to accomplish when using the Tree data structure. Additionally, look back at your code and comment sections that you had trouble with and write why you had trouble. Thus, if you ever get stuck again, you can look back at these notes and see why you struggled and how you dealt with that obstacle.

If working with Trees really interested you, explore other data structures that are similar to trees and continue to explore the complexity of trees! If this week did not interest you, don't worry because next week you will be learning about a new topic: Hash Maps and Edit Distance

Before you begin here is a helpful hint:

Give this last Module your all and use all of your resources including the videos, lessons, quizzes, and Discussion Forum in order to complete the final Module.

The rest of this reading contains the answers, as well as some explanations, for this week's quiz questions. Correct answers to the quiz questions are shown in **bold**. Incorrect answers are shown in *italics*.

Good luck and have fun learning!

Sincerely,

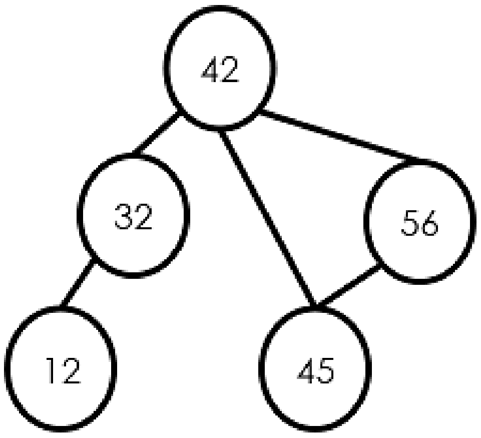
Our MOOC Team

Quiz Answers and Explanations

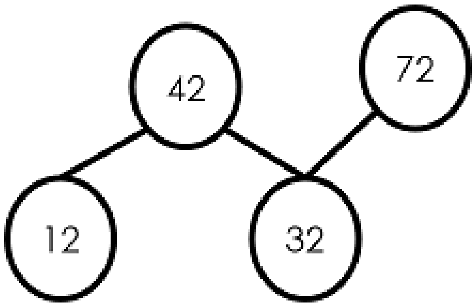
Correct answers are in **bold**. Incorrect answers are shown in *italics*.

1. Which of the following structures are trees? Select all that apply.

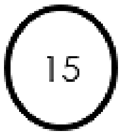
- *(Incorrect)* The answer below is *incorrect.* Notice the loop.



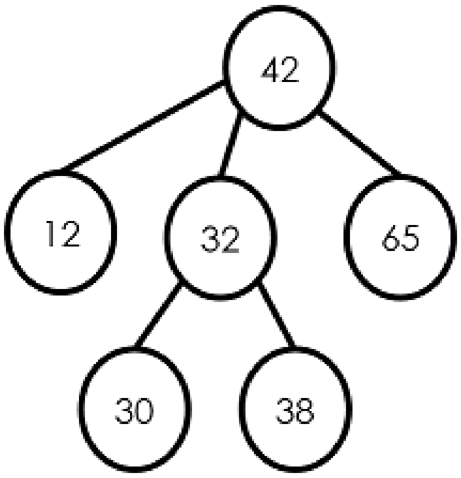
- *(Incorrect)* The answer below is *incorrect*. There are no loops, but notice that node 32 has two parents.



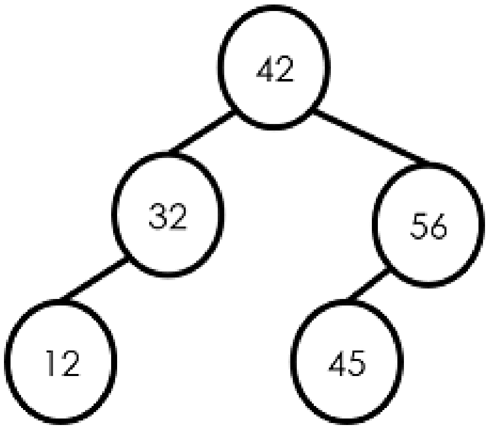
- **(Correct)** The answer below is **correct**. This single node has a root, it does not have more than one parent and there are no loops.



- **(Correct)** The answer below is **correct.** There is a single root, no loops and no node has more than 1 parent.

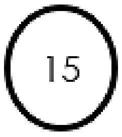


- **(Correct)** The answer below is **correct**. There is a single root, no loop and each node has no more than 1 parent.

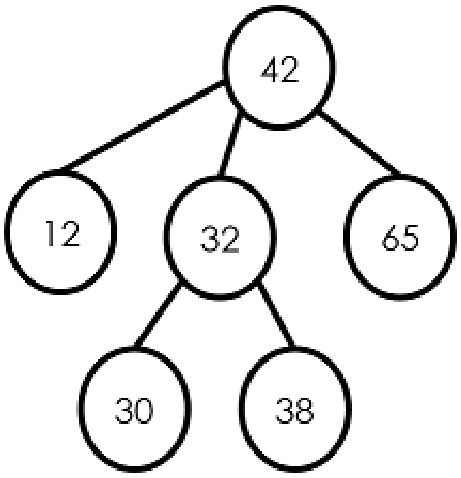


2. Which of the following are Binary Search Trees? Select all that apply.

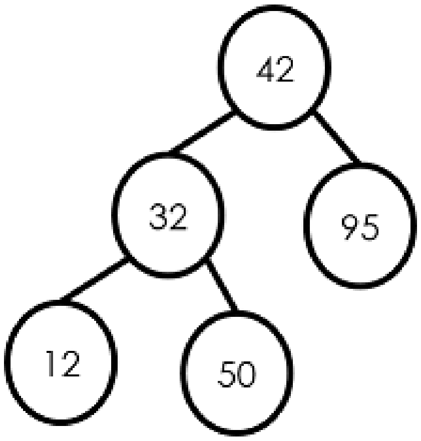
- **(Correct)** The answer below is **correct.** Even though there is just one node, this is a legal tree and the nodes on in the left subtree are less than the the root and the nodes in the right subtree are greater than the root.



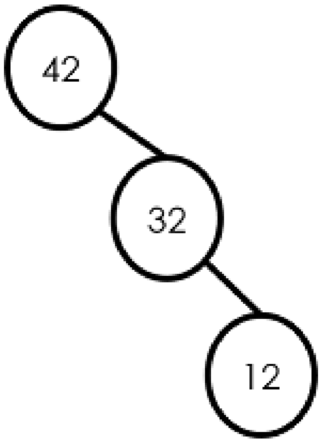
- *(Incorrect)* The answer below is *incorrect*. Is this a binary tree?



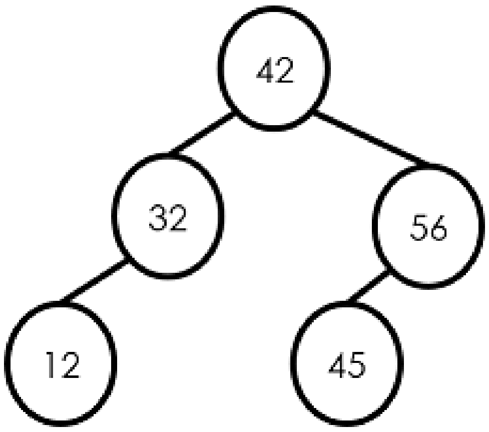
- *(Incorrect)* The answer below is *incorrect*. Remember that all nodes in the left subtree must be less than the node whose subtree they are, and all nodes in the right subtree must be greater. Notice that 50 is greater than 42, but found in its left subtree.



- *(Incorrect)* The answer below is *incorrect*. Nodes to the right should be greater than their parents.



3. Which of the following statements are true about the following binary search tree? Select all that apply.



Assume that no nodes have been deleted from this tree.

- *32 must have been inserted before 45 (Incorrect)*

Try the following insertion order: 42, 56, 45, 32, 12

- **A different insertion order with the same values could give a different tree structure. (Correct)**

Try a bunch of insertion orders and see what happens to the structure of the tree. E.g. try 12, 32, 42, 45, 56

- *There is only one insertion order that could have produced this tree. (Incorrect)*

Try a bunch of different insertion orders and see what happens.

- **42 must have been inserted first. (Correct)**

See what happens to the structure if you insert another value first.

- **32 must have been inserted before 12 (Correct)**

See what happens if you try inserting 12 before 32.

4. Which of the following statements are true about the running time of binary search trees? Select all that apply.

- **The tightest possible worst case time to find an element in an arbitrary BST is O(n) (Correct)**

It could be the case that the tree is totally unbalanced and looks just like a linked list.

- *The tightest possible worst case time to find an element in an arbitrary BST is O(log(n)) (Incorrect)*

Again, because the tree could look like a linked list, the worst case time is O(n), and no faster.

- **The tightest possible worst case time to find an element in a balanced BST is O(log(n)) (Correct)**

If you know the tree is balanced, then its height is at most O(log(n)).

- **Inserting sorted data into an unbalanced BST leads to the worst case BST structure (i.e. a structure where finding an element will take the longest). (Correct)**

Try it! You'll find this always leads to a BST that looks like a linked list.

- *The tightest possible worst case time to find an element in a balanced BST is O(n) (Incorrect)*

If you know the tree is balanced, then you can do better than O(n) because the height of the tree is at most O(log(n)).

5. From your benchmarking of the DictionaryLL and DictionaryBST structures you implemented, which structure is the better choice?

- *DictionaryLL (Incorrect)*

Plot the graphs to see which is faster. A linked list has O(n) find time, while a BST has O(log(n)) find time.

- **DictionaryBST (Correct)**

Plot the graphs to see which is faster. A linked list has O(n) find time, while a BST has O(log(n)) find time.

- *They are about the same (Incorrect)*

Plot the graphs to see which is faster. A linked list has O(n) find time, while a BST has O(log(n)) find time.

6. In your benchmarking of the DictionaryBST structure, you probably found that the time to find words did not significantly increase as the Dictionary got larger. Which of the following is the most likely reason for this behavior?

- **log(n) is sufficiently small and grows sufficiently slowly that other factors (e.g. memory use) had a bigger effect on the running time than the size to find the word in the dictionary. (Correct)**

You may recall some of this noise from your benchmarking.

- *We were measuring the best case performance of the dictionary, which does not change as the dictionary size grows. (Incorrect)*

It is not in fact the best case, because the word we were looking for wasn't even found in the dictionary.

- *The running time to find an element in a balanced BST is O(1) so we would not expect the running time to get larger as the dictionary got bigger. (Incorrect)*

Review Mia's lectures about BST running time. Balanced BSTs have O(log(n)) running time for find.