CS6033 Design and Analysis of Algorithms I
Homework Assignment 7 Part 1
December 05, 2022

Group Members:
rk4305 (Sai Rajeev Koppuravuri)
sg7372 (Sriharsha Gaddipati)
vt2182 (Venu Vardhan Reddy Tekula)
vt2184 (Veeravenkata Raghavendra Naveenkumar Tata)

---

1. (5 points) Show that if a node in a binary search tree has two children, then its successor has no left child and its predecessor has no right child.

   Ans:

   Yes, we can prove this using proof of contradiction.

   Suppose that a node in a binary search tree has two children and its successor has a left child. Let the value of this node be x, the value of its left child be y, the value of its right child be z, and the value of its successor's left child be w. Since the successor of x has a left child, we know that the value of w is greater than y and less than z.

   Since the value of w is greater than y, it follows that w is not in the left subtree of x. This means that w must be in the right subtree of x, which means that w is also in the right subtree of z. However, since the value of w is less than z, this contradicts the fact that z is the smallest value in the right subtree of x. Therefore, our assumption that the successor of x has a left child leads to a contradiction, and we can conclude that the successor of x cannot have a left child.

   We can prove the same thing for the predecessor of x using a similar argument. Suppose that the predecessor of x has a right child. Let the value of this right child be v. Since the value of v is less than z, it follows that v is not in the right subtree of x. This means that v must be in the left subtree of x, which means that v is also in the left subtree of y. However, since the value of v is greater than y, this contradicts the fact that y is the largest value in the left subtree of x. Therefore, our assumption that the predecessor of x has a right child leads to a contradiction, and we can conclude that the predecessor of x cannot have a right child.

   In summary, we have shown that if a node in a binary search tree has two children, then its successor cannot have a left child and its predecessor cannot have a right child. This completes the proof.

2. (5 points) You are searching for the number 363 in a binary search tree containing numbers between 1 and 1000. Which of the following sequences cannot be the sequence of nodes examined? Also, justify your answer.

   A. 2, 252, 401, 398, 330, 344, 397, 363
   B. 924, 220, 911, 244, 898, 258, 362, 363
   C. 925, 202, 911, 240, 912, 245, 363

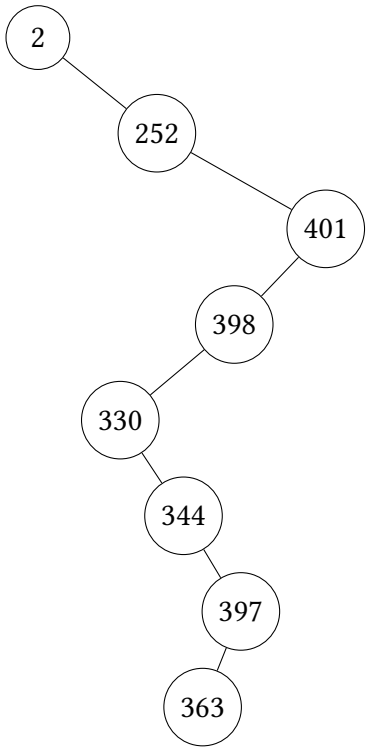
D. 2, 399, 387, 219, 266, 382, 381, 278, 363
E. 935, 278, 347, 621, 299, 392, 358, 363

Ans:

In a binary search tree, the left subtree of a node contains all nodes with values that are less than the value of the node, and the right subtree of a node contains all nodes with values that are greater than the value of the node. This means that when we search for a particular value in a binary search tree, we first compare the value we are searching for with the value of the root node. If the value we are searching for is less than the value of the root node, we search for it in the left subtree of the root node. If the value we are searching for is greater than the value of the root node, we search for it in the right subtree of the root node. We continue this process until we find the value we are searching for or until we reach a leaf node (a node with no children) without finding the value.

A. 2, 252, 401, 398, 330, 344, 397, 363

We can construct the below binary search tree from the above sequence

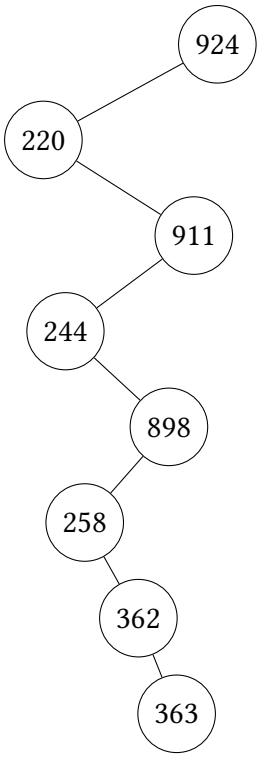


Below is the sequence needs to be followed to find 363

2, 252, 401, 398, 330, 344, 397, 363. Obtained sequence is same as given sequence.

B. 924, 220, 911, 244, 898, 258, 362, 363

We can construct the below binary search tree from the above sequence

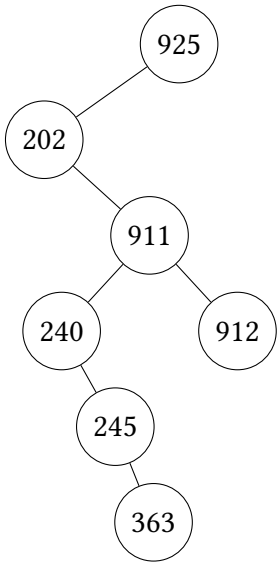


Below is the sequence needs to be followed to find 363

924, 220, 911, 244, 898, 258, 362, 363. Obtained sequence is same as given sequence.

C. 925, 202, 911, 240, 912, 245, 363

We can construct the below binary search tree from the above sequence



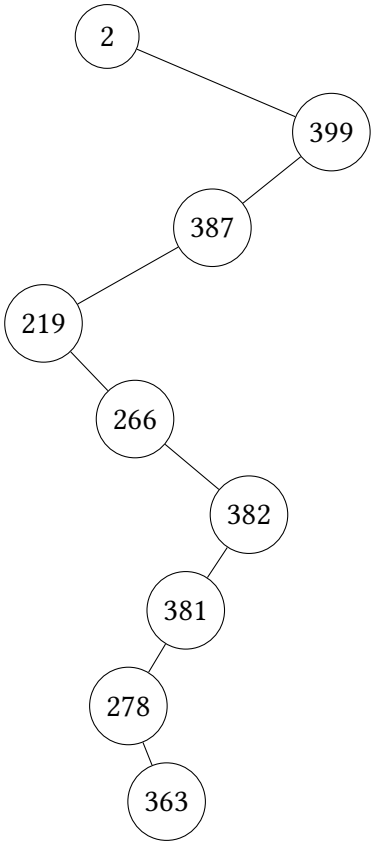
Below is the sequence needs to be followed to find 363

925, 202, 911, 240, 245, 363. Obtained sequence got differed from the given sequence because node 912 doesn't encounter while searching for node 363.

It violates the given sequence.

D. 2, 399, 387, 219, 266, 382, 381, 278, 363

We can construct the below binary search tree from the above sequence

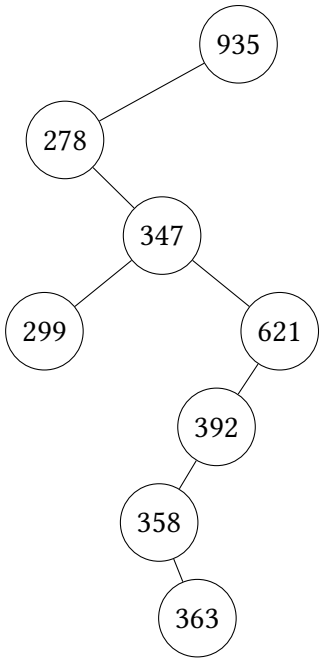


Below is the sequence needs to be followed to find 363

2, 399, 387, 219, 266, 382, 381, 278, 363. Obtained sequence is same as given sequence.

E. 935, 278, 347, 621, 299, 392, 358, 363

We can construct the below binary search tree from the above sequence



Below is the sequence needs to be followed to find 363

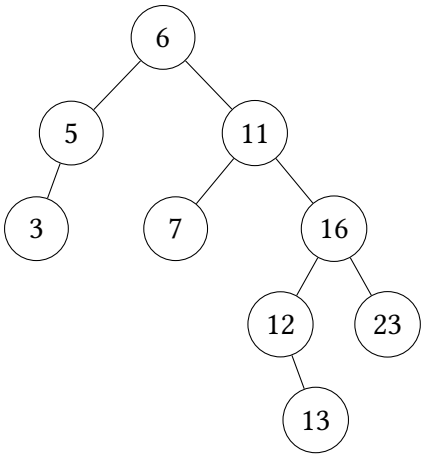
935, 278, 347, 621, 392, 358, 363. Obtained sequence got differed from the given sequence because node 299 doesn't encounter while searching for node 363.

It violates the given sequence.

3. (5 points) Suppose that you construct a binary search tree by repeatedly inserting distinct values into the tree. Argue that the number of nodes examined in searching for a value in the tree is 1 plus the number of nodes examined when the value was first inserted into the tree.

   Ans:

   The claim is true because when we search for our node, we check it's value when we reach it. When it was inserted, we only checked if the appropriate child was NIL and inserted our node there. On the other hand, upon reaching the node we are searching for, we still have to check if it is NIL and then check if it is the desired value. That is one comparison more than the case when we were inserting the node.

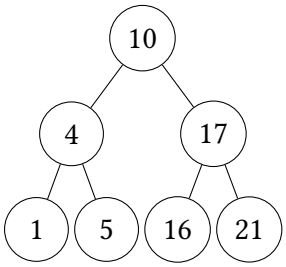


For example, let's take a look at our example here. When inserting the green node, we compared it with 5,10,15 and 11 and inserted it as the right child of the node with the value of 11. That means we had 4 value comparisons. On the other hand, when searching for our green node, we compare it with 5,10,15,11 and then itself, which means we have 5 comparisons in that operation. This confirms the claim from the book.

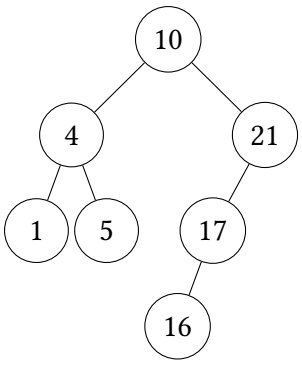
4. (15 points) For the set $\{1, 4, 5, 10, 16, 17, 21\}$ of keys, draw binary search trees of heights 2, 3, 4, 5 and 6.

   Ans:

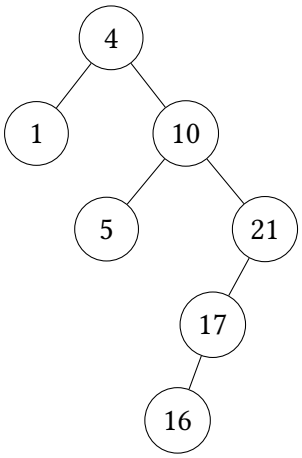
   Binary search tree of height 2:



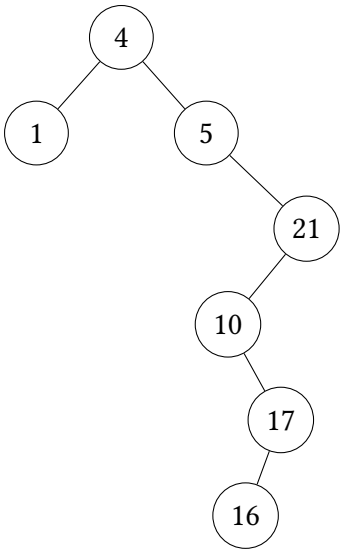
   Binary search tree of height 3:



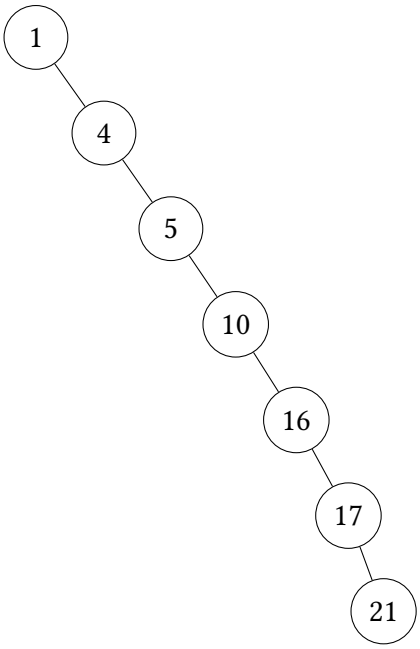
Binary search tree of height 4:



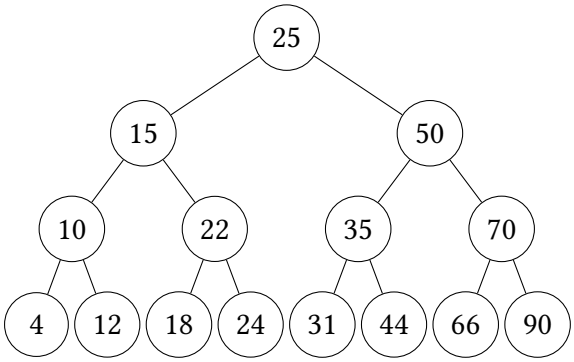
Binary search tree of height 5:



Binary search tree of height 6:



5. (10 points) Given the below binary search tree. Write inorder, preorder and postorder Traversal for the same. Justify the behavior of the inorder traversal output.



Ans:

The inorder traversal (left, root, right) of the given binary search tree is
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90.

The preorder traversal (root, left, right of the given binary search tree is
25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90.

The postorder traversal (left, right, root) of the given binary search tree is
4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25.

In an inorder traversal of a binary search tree, the nodes are visited in the order of their values. In the given binary search tree, the left subtree of a node contains all nodes with values that are less than the value of the node, and the right subtree of a node contains all nodes with values that are greater than the value of the node. This means that in an inorder traversal of the given binary search tree, the nodes in the left subtree of a node are visited before the node, and the nodes in the right subtree of a node are visited after the node. This explains the behavior of the inorder traversal output, which is sorted in ascending order.

6. (10 points) Determine whether given binary tree is binary search tree(BST) or not. Write a Puesdocode for the same and running time. (Note : You can assume that there are no duplicates elements in the tree)

   Ans:

   We can find if the given tree is a binary tree by using the following algorithm

   (a) Initializing $(mini, maxi)$ as $-\infty$, ∞

   (b) Define a function $is\_bst(root, mini, maxi)$ that takes a node $root$ and minimum and maximum values $mini$ and $maxi$ as arguments and returns a boolean indicating whether the binary tree rooted at $root$ is a BST.

   (c) If $root$ is $null$, return $true$ (because an empty tree is considered to be a BST).

   (d) If $root.value$ is less than $mini$ or greater than $maxi$, return $false$ (because a BST must satisfy the property that all nodes in the left subtree of a node have values that are less than the value of the node, and all nodes in the right subtree of a node have values that are greater than the value of the node).

   (e) Recursively call $is\_bst(root.left, mini, root.value)$ and $is\_bst(root.right, root.value, maxi)$ to check whether the left and right subtrees of root are BSTs. Return $true$ if both subtrees are BSTs, and $false$ otherwise.

   Here is a sample implementation of the is_bst function in Python:

   mini = $-\infty$
   maxi = ∞

```
def is_bst(root, mini, maxi):
    if root is None:
        return True
    if root.value < mini or root.value > maxi:
        return False
    return is_bst(root.left, mini, root.value)
        and is_bst(root.right, root.value, maxi)
```

   The time complexity of this algorithm is O(n), where n is the number of nodes in the BST. This is because the algorithm visits every node in the tree exactly once.