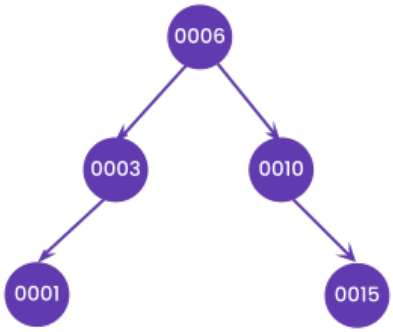
BST

Assignment Solutions

PW SKILLS

Q1. Given an integer array of n elements. You need to check whether it represents the inorder traversal of a Binary Search Tree (BST) or not. [Easy]

Input-1:
n = 5
values = [1, 3, 6, 10, 15]
Output-1:
Yes



Input-2:
n = 5
values = [1, 6, 3, 10, 15]
Output-2:
No
Explanation: Here, in both the cases the trees are the same, but input-2 isn't the correct inorder traversal of BST but input-1 is.

A1. Approach:
- We are going to take use of the fact that inorder of BST is always in sorted order.
- So, we will check if the array is sorted or not.
  a. If sorted, it is inorder of BST.
  b. If not sorted, it is not inorder of BST.

Solution Code: https://pastebin.com/MH0SBdGM

Output:

```
Enter the size of array: 5
Enter the elements of the array: 1 3 6 10 15

Is Inorder? Yes
```

```
Enter the size of array: 5
Enter the elements of the array: 1 6 3 10 15

Is Inorder? No
```

Time complexity: O(N) where N is the number of elements in the array. Since we have to traverse the array at least once in order to check if it is sorted or not hence linear time complexity is there.

Space complexity: O(1), no extra memory is used for storing elements.

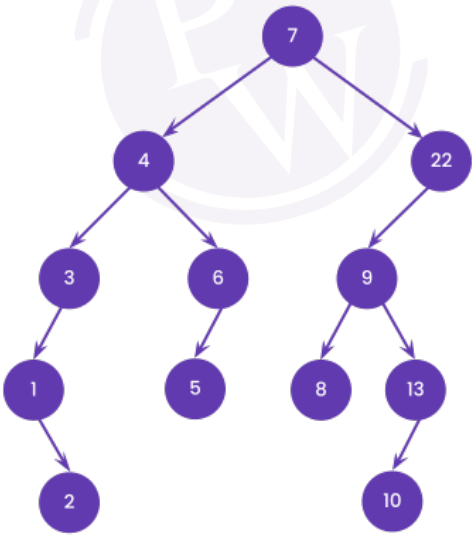
Q2. Differentiate between Binary Tree and Binary Search Tree.

A2. Here's a table summarizing the main differences between Binary Tree and Binary Search Tree:

	Binary Tree	Binary Search Tree
Property	Each node can have at most two children	While each node can have atmost two children, each node also has a value greater than its left child and less than its right child
Ordering	No specific ordering of nodes	Nodes are ordered based on their values
Searching	Less efficient compared to BST	Efficient searching operation possible
Sorting	Inefficient for sorting operations	Efficient for sorting operations
Insertion	Insertion of a new node does not follow any specific rule	Insertion of a new node follows the specific ordering rule based on the node's value
Deletion	Deletion of a node may result in an unbalanced tree	Deletion of a node may require restructuring of the tree to maintain the BST property
Height	The height of a Binary Tree can vary widely	The height of a Binary Search Tree is typically balanced, resulting in faster search operations
Example	A family tree is an example of a Binary Tree	A database index is an example of a Binary Search Tree

Q3. Consider the following values: 7, 22, 4, 6, 9, 3, 1, 13, 2, 5, 10, 8
They are inserted into BST in the given order only. What will be the height of the BST?

A3. The BST created after inserting the values in the given order will be:



Height of this BST = 5

Q4. Which of the following statements about Binary Search Trees is true?
a) All Binary Search Trees are balanced trees.
b) In a Binary Search Tree, the left subtree of a node contains only smaller values than the node, and the right subtree contains only larger values than the node.

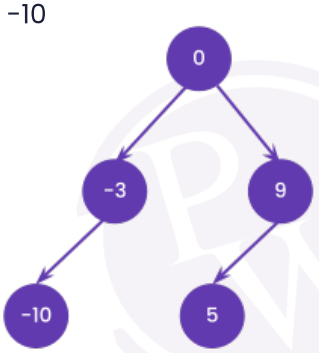
c) Binary Search Trees have O(1) search complexity.
d) The height of a Binary Search Tree is always equal to the number of nodes in the tree.

A4. Answer: b) In a Binary Search Tree, the left subtree of a node contains only smaller values than the node, and the right subtree contains only larger values than the node.

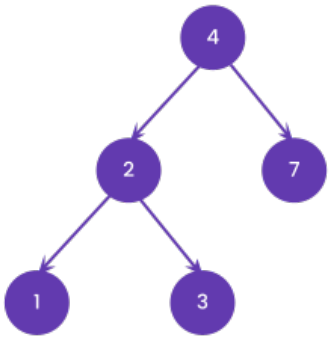
Explanation: A Binary Search Tree is a tree data structure where each node has at most two children, and the left subtree of a node contains only values smaller than the node, while the right subtree contains only values larger than the node. This property makes searching for a value in a Binary Search Tree very efficient, with a worst-case time complexity of O(h), where h is the height of the tree. Option b) is therefore the correct answer. Option a) is incorrect because not all Binary Search Trees are balanced trees, although a balanced Binary Search Tree is desirable for efficient searching. Option c) is incorrect because Binary Search Trees have O(h) search complexity, which depends on the height of the tree. Option d) is incorrect because the height of a Binary Search Tree can vary depending on the arrangement of the nodes, and is not always equal to the number of nodes in the tree.

Q5. Given an integer n representing the number of nodes in a BST. Next are the values of nodes in BST. Find the node with the minimum value.

Input-1:
n = 5
values = [0,-3,9,-10,5]
Output-1:
-10



Input-2:
n = 5
values = [4,2,7,1,3]
Output-2:
1



Approach: In a BST, the left node always contains a value lesser than the current node. So, if we move to the most left node in BST, then it will be having the least value in the BST. So here we will either recursively or

iteratively reach the most left node and will return its value.

Solution Code: https://pastebin.com/B3mV8zs2

Output:

```
Enter the number of nodes: 5
Enter the preorder of BST: 4 9 0 5 1

Second largest value of BST: 5
```

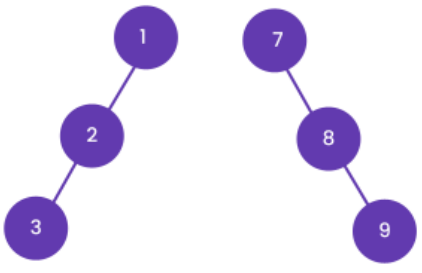
```
Enter the number of nodes: 5
Enter the preorder of BST: 4 2 7 1 3

Second largest value of BST: 4
```

Time complexity: O(N) where N is the number of nodes in the BST. Even if we are only going right here, in the worst case, when it is a right-skewed BST, then we will be traversing all the nodes of BST.
Space complexity: O(N), no extra memory is used for storing elements, however, depth first search may use call stack for function calls, hence in worst case i.e. skewed BST, linear space complexity can be there.

Q7. Choose what statements are True or False. Also, specify the reason for the same:
A) Every Binary Search Tree is a height-balanced Binary Tree.
B) Inorder of a BST is in non-decreasing order.
C) The worst case time complexity of operations on BST is when it is left or right skewed.
D) The left and right subtree of each node in BST are also BST.
E) In a height balanced BST, the height of left and right subtree of only the root node differ by not more than 1.
F) The worst case time complexity of operations on a balanced BST is O(N), where N is number of nodes in BST.
G) The height of a balanced BST is always, O(logN), where N is number of nodes.
H) In a right skewed BST, the values of nodes will be in ascending order, starting from the root till the rightmost node i.e. the leaf node.

A7. Answer and Explanation:
A) **False:** Left and right skewed BST are also there and they are not height-balanced binary trees. For eg:



Left-skewed BST **Right-skewed BST**

B) **True:** In a BST the values of the nodes in the left subtree are lesser than the root, and values of the nodes in the right subtree are higher than the root. In the inorder traversal, the order of traversing is left node -> root node -> right node. Thus, the order becomes ascending or non-decreasing (in case there are repetitive values).

C) **True:** The average case time complexity of operations on BST is O(logN) where N is the number of nodes and this is because we need not traverse half the subtree when the comparison is made.
But in case of skewed BST, since there is no subtree in left or right of the nodes, depending upon how it is

skewed, we have to traverse all the nodes of the tree thus giving worst case time complexity of O(N).

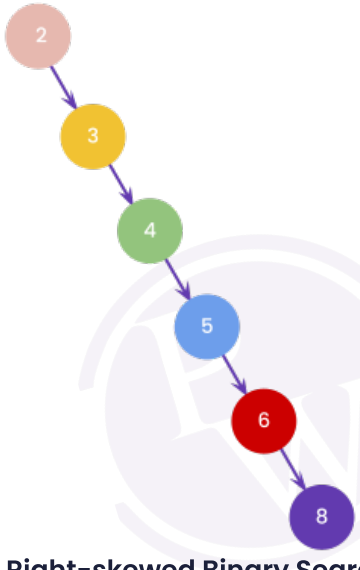
D) **True:** It is one of the property of the BST, that the left and right subtrees of each node in a BST should also be BSTs.

E) **False:** In a height balanced BST, the height of left and right subtree of not only the root node but also every node should differ by not more than 1.

F) **True:** In the worst case, we need to traverse each node of BST, thus making the time complexity of operations as O(N), where N is the number of nodes in BST.

G) **True:** Height of not only balanced BST, but also balance binary tree is also in order O(logN), where N stands for number of nodes in the tree.

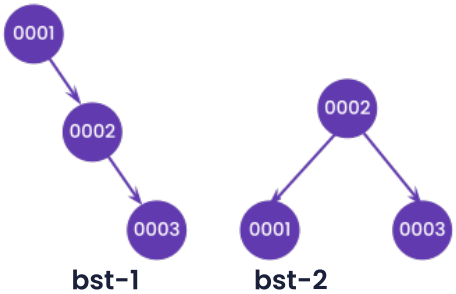
H) **True:** Since, the right subtree of the current node always contains a value greater than the current node's value, so with respect to this, in case of right-skewed BST, starting from root till leaf node, values will be in increasing order. **For eg:**



Right-skewed Binary Search Tree (BST)

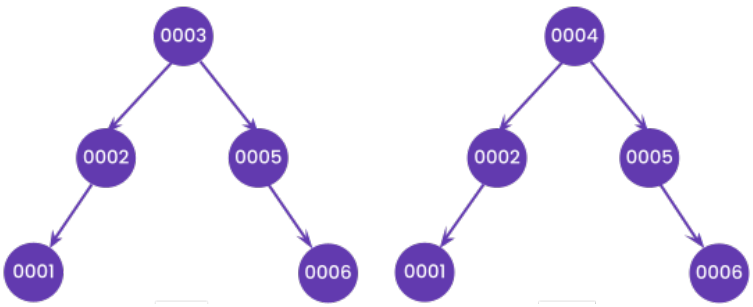
Q8. Given two BSTs. Check if they contain the same set of elements or not.

Input-1:
n = 3
bst1 = [1, 2, 3]
bst2 = [2, 1, 3]
Output-1: Yes



bst-1 **bst-2**

Input-2:
n = 5
bst1 = [3, 5, 6, 2, 1]
bst2 = [4, 2, 1, 5, 6]
Output-2: No



bst-1 bst-2

A8. Approach:

- The inorder traversal of a BST is all its nodes in sorted order.
- So, we will do inorder traversal of both the BSTs. Thus, generating two arrays.
- Then we will compare these two arrays.
- If both arrays are same then the two BSTs have same set of elements otherwise not.

Solution Code: https://pastebin.com/4SCjZjMw

Output:

```
BST-1: 1 2 3
BST-2: 2 1 3

Do they contain same set of elements? Yes
```

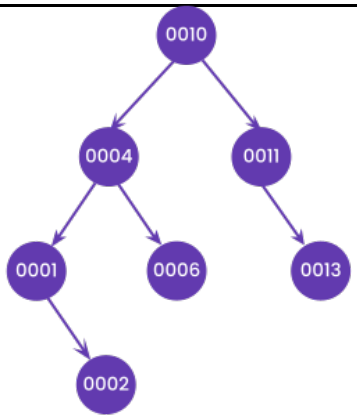
```
BST-1: 3 2 1 5 6
BST-2: 4 2 1 5 6

Do they contain same set of elements? No
```

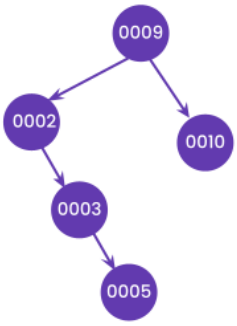
Time complexity: O(N) where N is the number of nodes in the BST. Here, we have to traverse each and every node of BST.
Space complexity: O(N), because we are using two vectors to store the values of BST nodes.

Q9. Given a binary search tree and two integers low and high. Print the sum of values of all nodes with values in the range [low, high].

Input-1:
n = 7
Elements = [10, 4, 5, 11, 13, 1, 2]
Low = 4
High = 12
Output-1:
Sum = 30



Input-2:
n = 5
Elements = [9, 10, 2, 3, 5]
Low = 1
High = 5
Output-2:
Sum = 10



A9. Approach:

- Here, we recursively traverse through the BST in preorder fashion.
- And check the current node's value.
- If the node's value lies between low and high, then we add it to sum.
- Otherwise,
  - If the node's value < low, then we go to right of the node.
  - If the node's value > high, then we go to left of the node.
- And at the end we return the sum.

Solution Code: https://pastebin.com/dL8hxnPd

Output:

```
Enter the number of nodes = 7
Enter the values of nodes = 10 4 5 11 13 1 2
Low = 4
High = 12

Range Sum = 30
```

```
Enter the number of nodes = 5
Enter the values of nodes = 9 10 2 3 5
Low = 1
High = 5

Range Sum = 10
```

Time complexity: O(N) as in the worst case we need to traverse each node of BST.
Space complexity: O(N) because we are using recursion and it requires additional space to maintain the function call stack. In case of skewed BST, the size of stack would be equal number of nodes in BST, thus making worst case space complexity as linear with respect to number of nodes present in the BST.