Date : 12th - Oct- 2020
Morning Session : 9am – 11.00 PM
By ~ Rohan Kumar

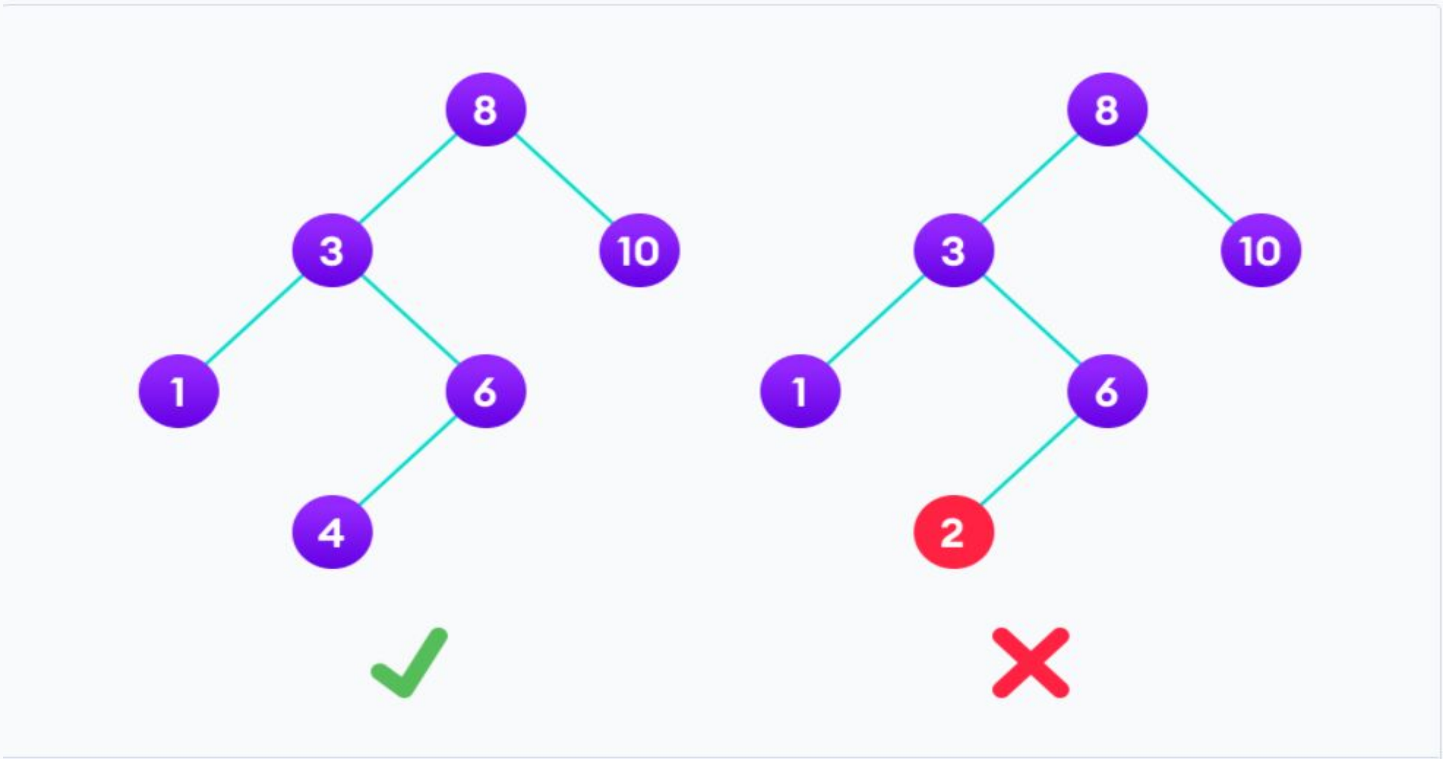
Topics: Binary Search Tree

Binary search tree is a data structure that quickly allows us to maintain a sorted list of numbers.

- It is called a binary tree because each tree node has a maximum of two children.
- It is called a search tree because it can be used to search for the presence of a number in O(log(n)) time.

The properties that separate a binary search tree from a regular binary tree is

1. All nodes of left subtree are less than the root node
2. All nodes of right subtree are more than the root node
3. Both subtrees of each node are also BSTs i.e. they have the above two properties



A tree having a right subtree with one value smaller than the root is shown to demonstrate that it is not a valid binary search tree

The binary tree on the right isn't a binary search tree because the right subtree of the node "3" contains a value smaller than it.

There are two basic operations that you can perform on a binary search tree:

Search Operation

The algorithm depends on the property of BST that if each left subtree has values below root and each right subtree has values above the root.

If the value is below the root, we can say for sure that the value is not in the right subtree; we need to only search in the left subtree and if the value is above the root, we can say for sure that the value is not in the left subtree; we need to only search in the right subtree.

Insert Operation

Inserting a value in the correct position is similar to searching because we try to maintain the rule that the left subtree is lesser than root and the right subtree is larger than root.

We keep going to either right subtree or left subtree depending on the value and when we reach a point left or right subtree is null, we put the new node there.

Deletion Operation

There are three cases for deleting a node from a binary search tree.

Case I

In the first case, the node to be deleted is the leaf node. In such a case, simply delete the node from the tree.

Case II

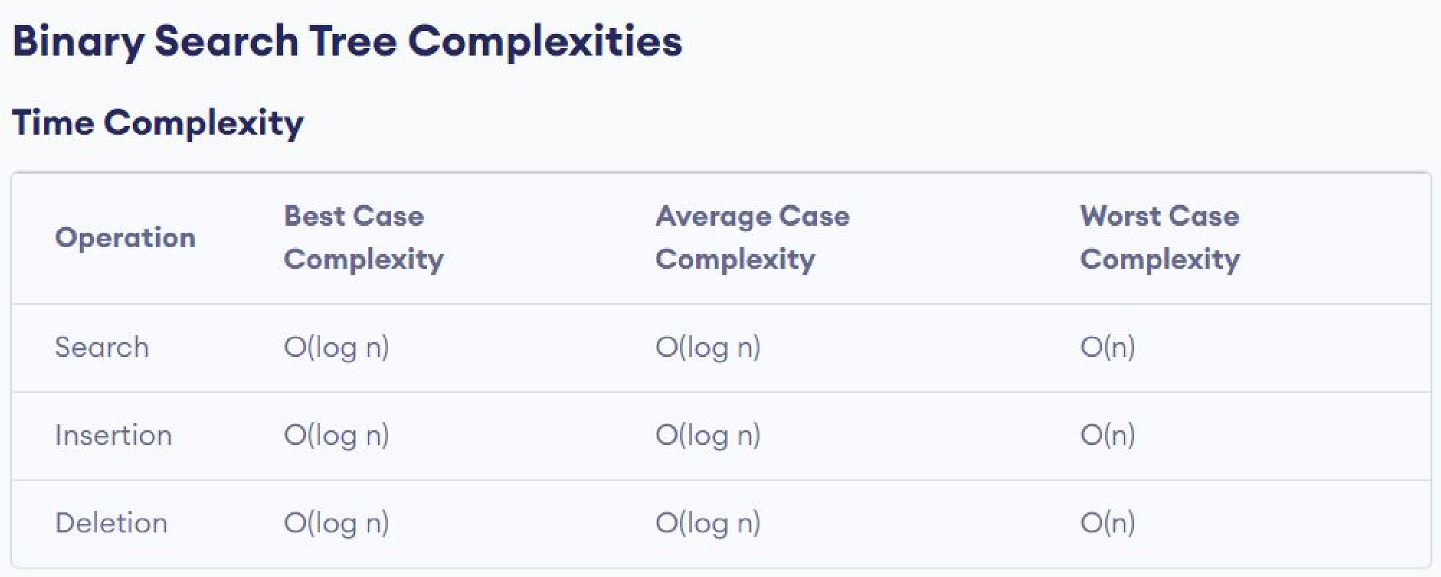
In the second case, the node to be deleted lies has a single child node. In such a case follow the steps below:

1. Replace that node with its child node.
2. Remove the child node from its original position.

Case III

In the third case, the node to be deleted has two children. In such a case follow the steps below:

1. Get the inorder successor of that node.
2. Replace the node with the inorder successor.
3. Remove the inorder successor from its original position.



Binary Search Tree Complexities

Time Complexity

Operation	Best Case Complexity	Average Case Complexity	Worst Case Complexity
Search	O(log n)	O(log n)	O(n)
Insertion	O(log n)	O(log n)	O(n)
Deletion	O(log n)	O(log n)	O(n)

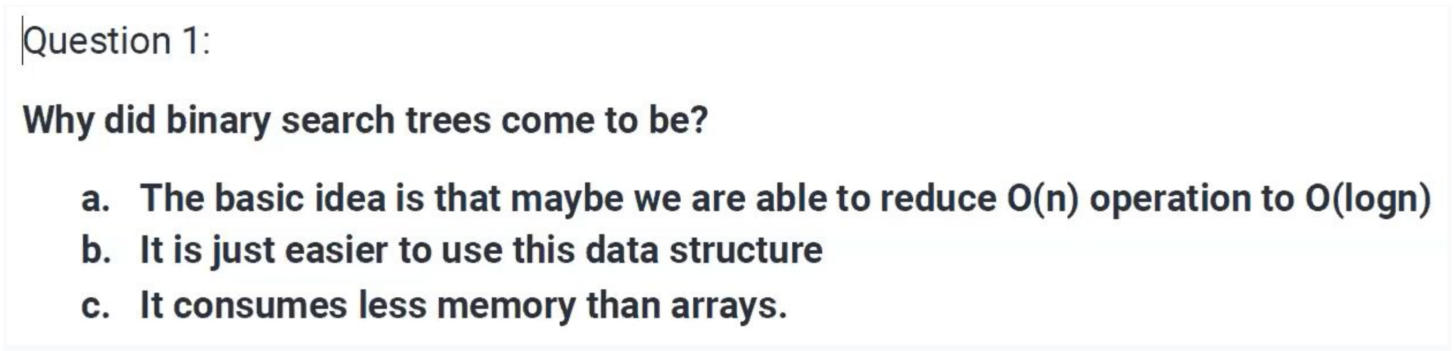
Here, `N` is the number of nodes in the tree.

Space Complexity

The space complexity for all the operations is `O(n)`.

Binary Search Tree Applications

1. In multilevel indexing in the database
2. For dynamic sorting
3. For managing virtual memory areas in Unix kernel

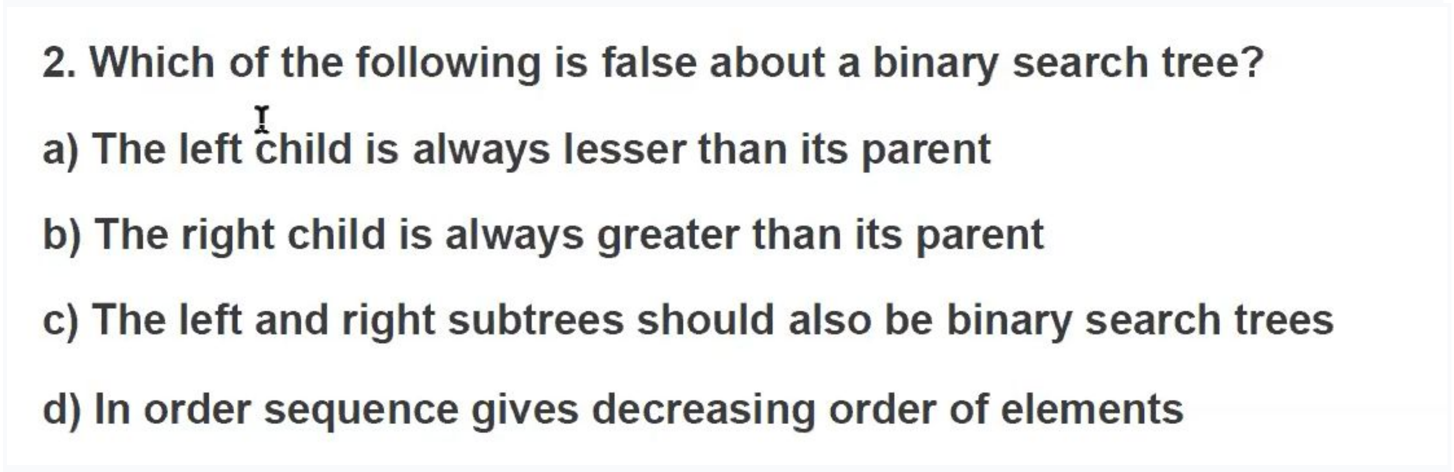


Question 1:

Why did binary search trees come to be?

a. **The basic idea is that maybe we are able to reduce O(n) operation to O(logn)**
b. **It is just easier to use this data structure**
c. **It consumes less memory than arrays.**

Answer: A



2. Which of the following is false about a binary search tree?

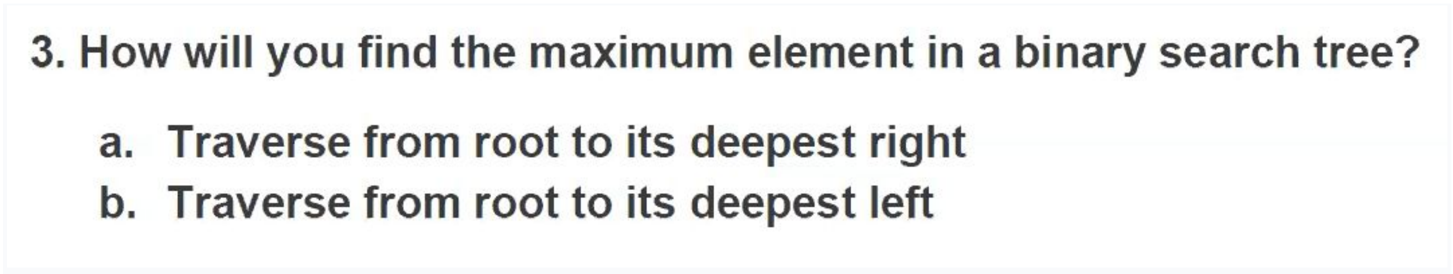
a) The left child is always lesser than its parent

b) The right child is always greater than its parent

c) The left and right subtrees should also be binary search trees

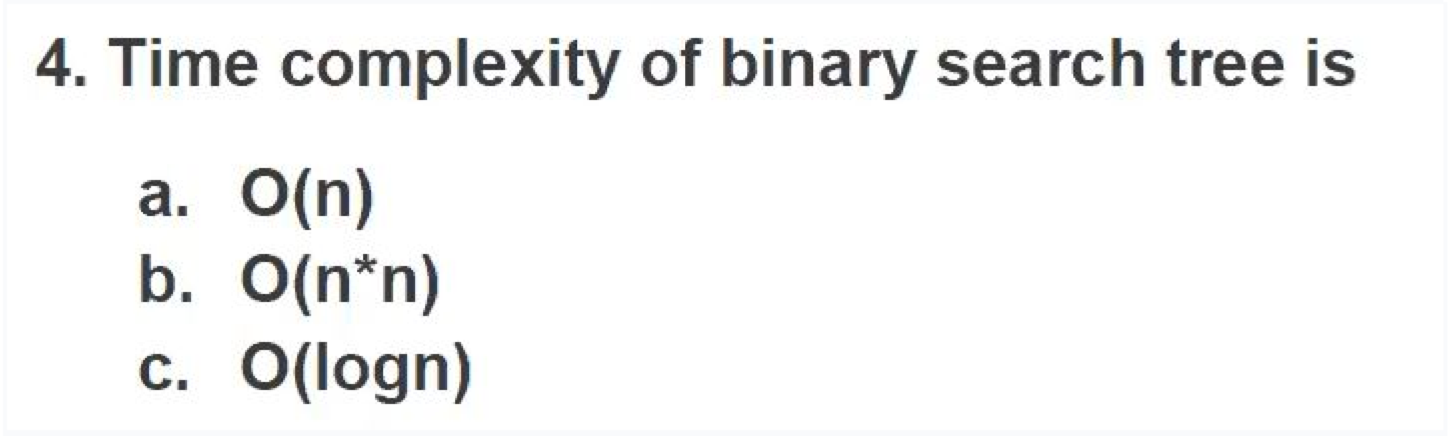
d) In order sequence gives decreasing order of elements

Answer: D


3. How will you find the maximum element in a binary search tree?

a. Traverse from root to its deepest right
b. Traverse from root to its deepest left

Answer: A


4. Time complexity of binary search tree is

a. O(n)
b. O(n*n)
c. O(logn)

Answer: C

For Binary Search Tree Code implementation please go through record lecture

[Record Lecture]()

Resource Link;

https://www.programiz.com/dsa/binary-search-tree