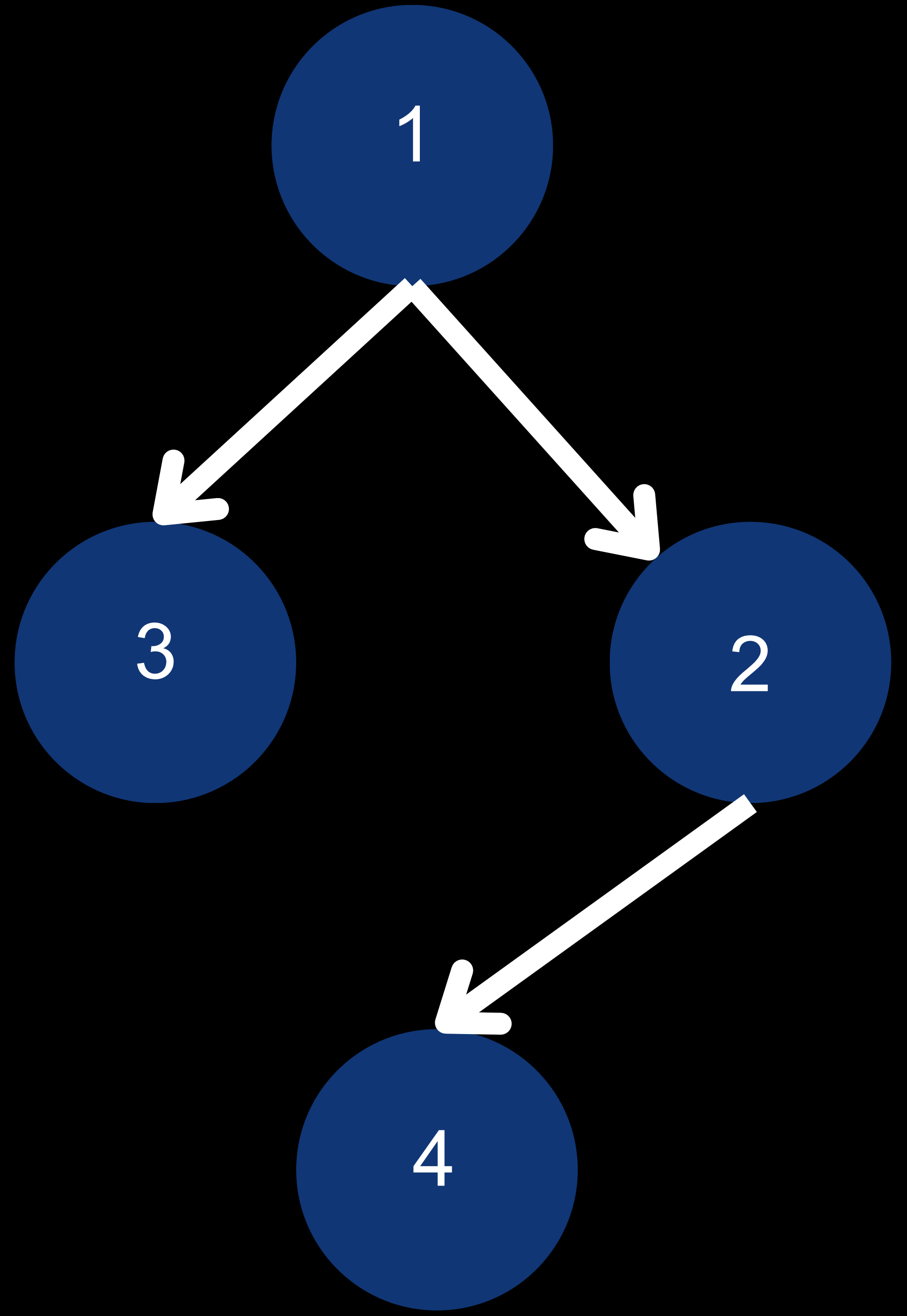
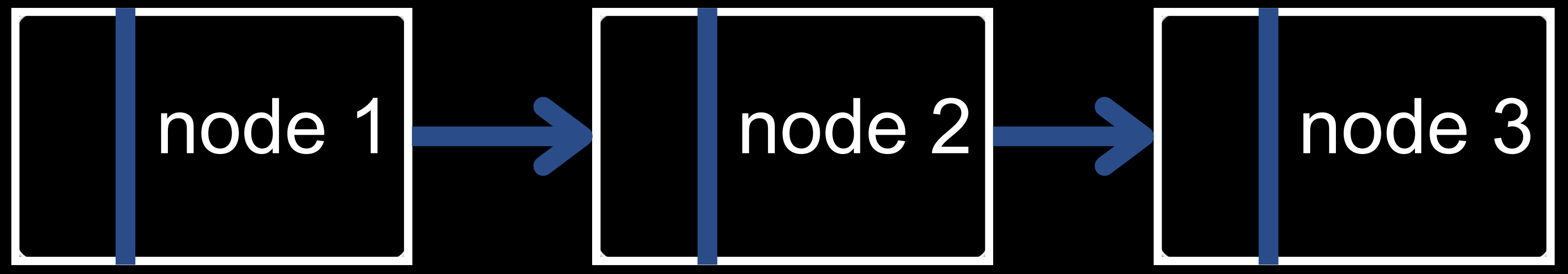
DATA STRUCTURE





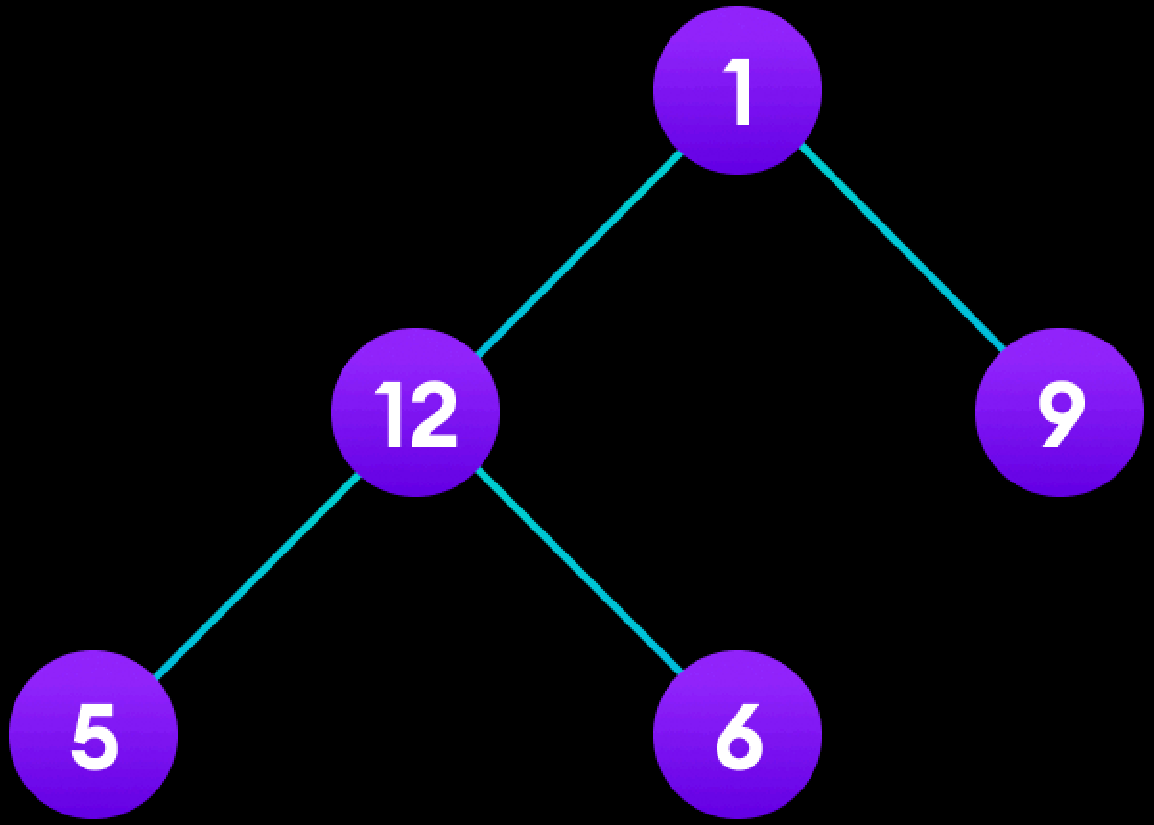
essammohamedst1@gmail.com

Tree Traversal

Traversing a tree means visiting every node in the tree

Linear data structures like arrays, stacks, queues, and linked list have only one way to read the data. But a hierarchical data structure like a tree can be traversed in different ways.

Let's think about how we can read the elements of the tree in the image



Starting from top, Left to right
1 -> 12 -> 5 -> 6 -> 9

Starting from bottom, Left to right
5 -> 6 -> 12 -> 9 -> 1

Although this process is somewhat easy, it doesn't respect the hierarchy of the tree, only the depth

of the nodes.

Instead, we use traversal methods that take into account the basic structure of a tree i.e.

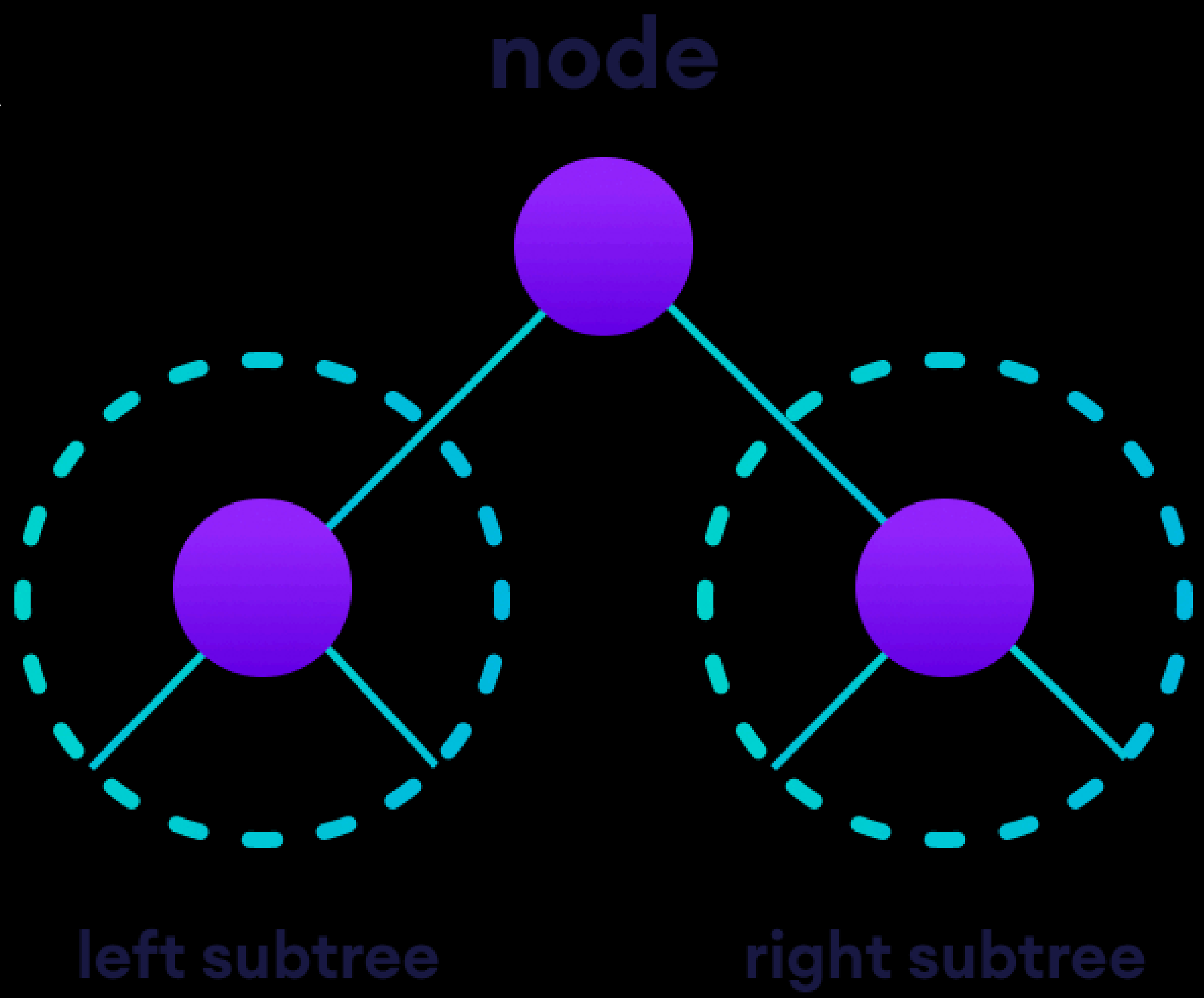
```
struct node {
    int data;
    struct node* left;
    struct node* right;
}
```

The struct node pointed to by left and right might have other left and right children so we should think of them as sub-trees instead of sub-nodes.

According to this structure, every tree is a combination of

A node carrying data

Two subtrees



Remember that our goal is to visit each node, so we need to visit all the nodes in the subtree, visit the root node and visit all the nodes in the right subtree as well.
Depending on the order in which we do this, there can be three types of traversal.

Inorder traversal

First, visit all the nodes in the left subtree
Then the root node
Visit all the nodes in the right subtree

```
inorder(root->left)
display(root->data)
inorder(root->right)
```

Preorder traversal

Visit root node
Visit all the nodes in the left subtree
Visit all the nodes in the right subtree

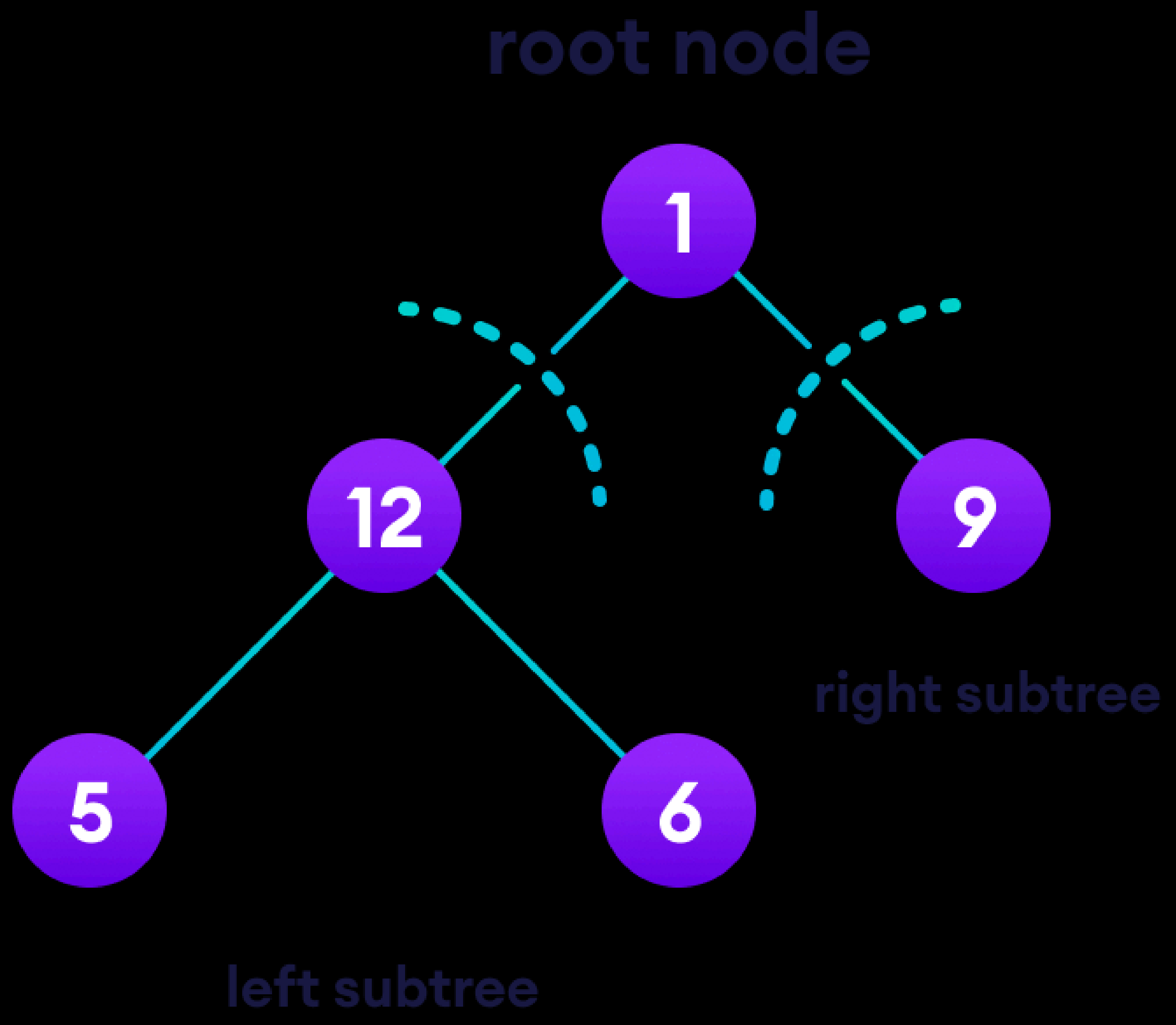
```
display(root->data)
preorder(root->left)
preorder(root->right)
```

Postorder traversal

Visit all the nodes in the left subtree
Visit all the nodes in the right subtree
Visit the root node

```
  postorder(root->left)
 postorder(root->right)
  display(root->data)
```

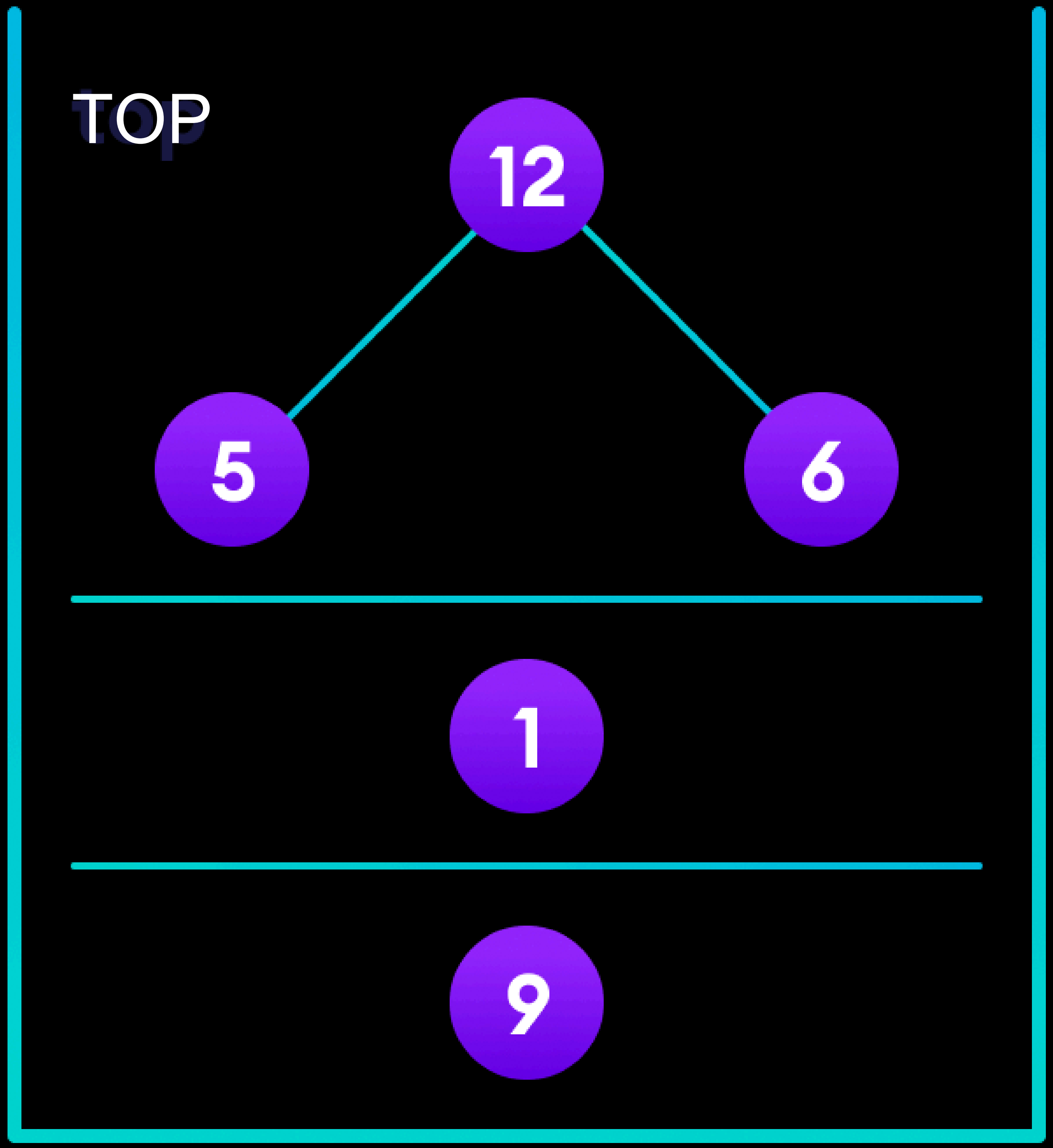
in-order traversal. We start from the root node.



We traverse the left subtree first. We also need to remember to visit the root node and the right subtree when this tree is done.

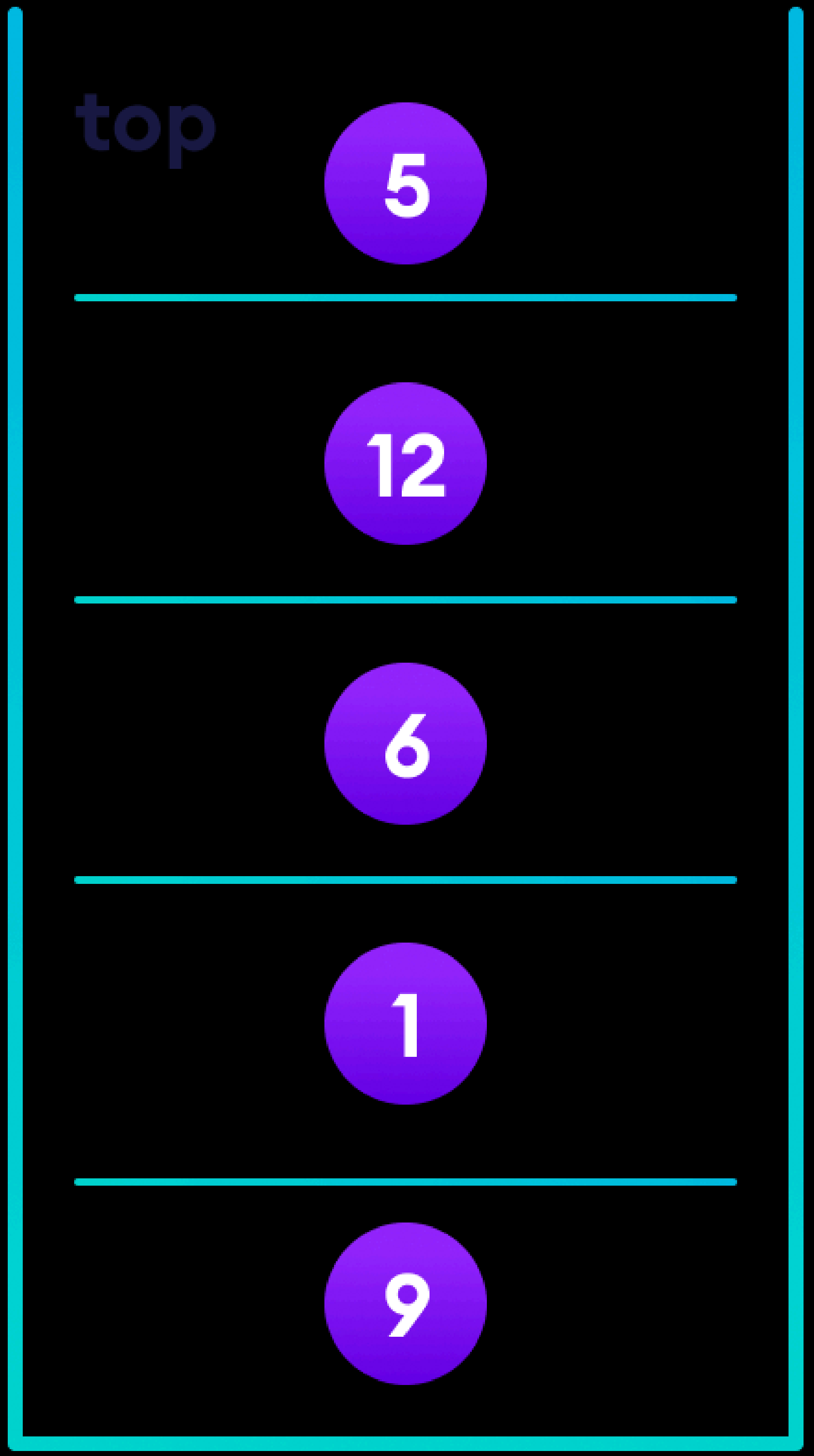
Now we traverse to the subtree pointed on the TOP of the stack.

Left subtree -> root -> right subtree



After going through all the elements,
we get the inorder traversal as
5 -> 12 -> 6 -> 1 -> 9

We don't have to create the stack ourselves because recursion maintains the correct order for us.



Binary Search Tree(BST)

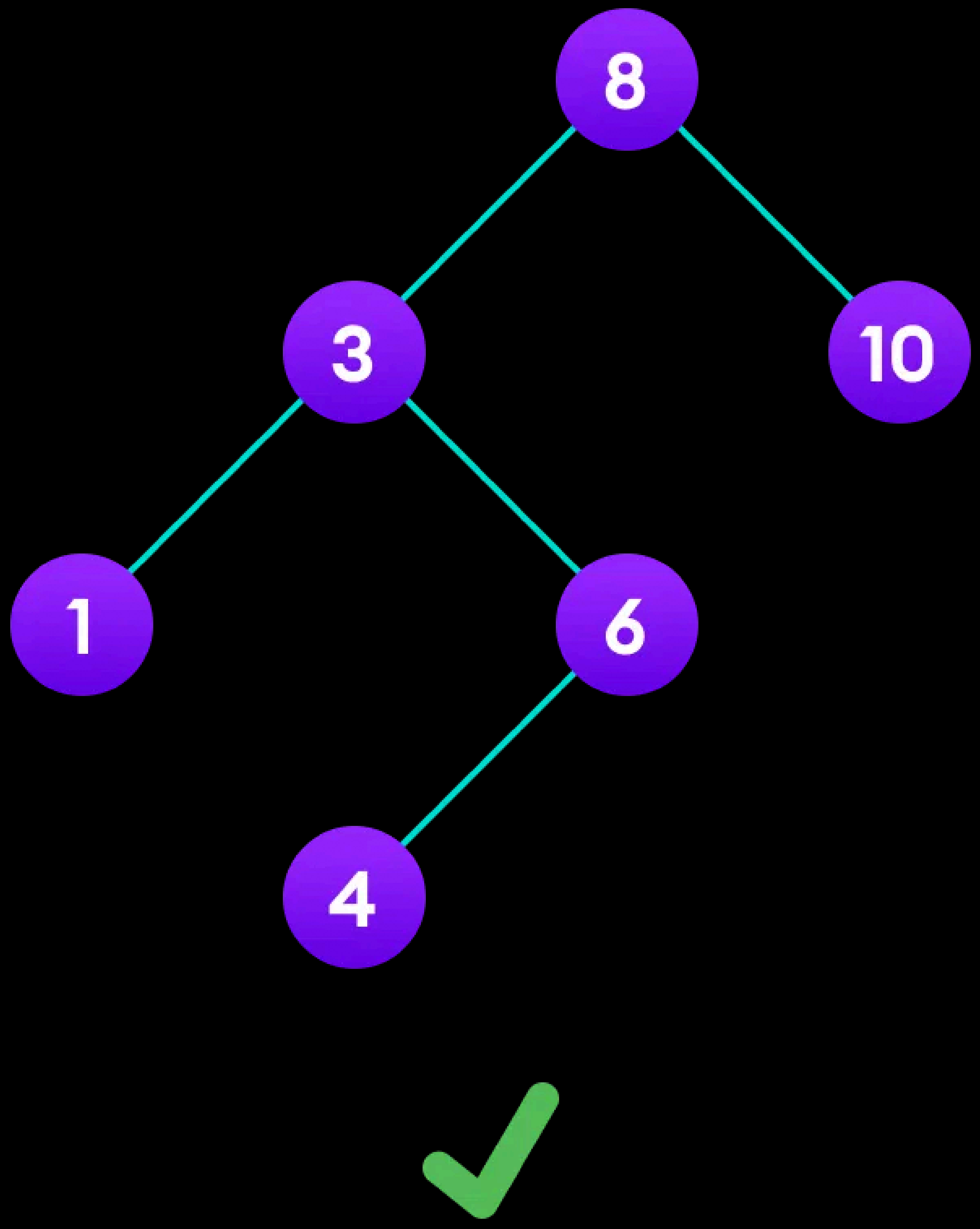
Binary search tree is a data structure that quickly allows us to maintain a sorted list of numbers.

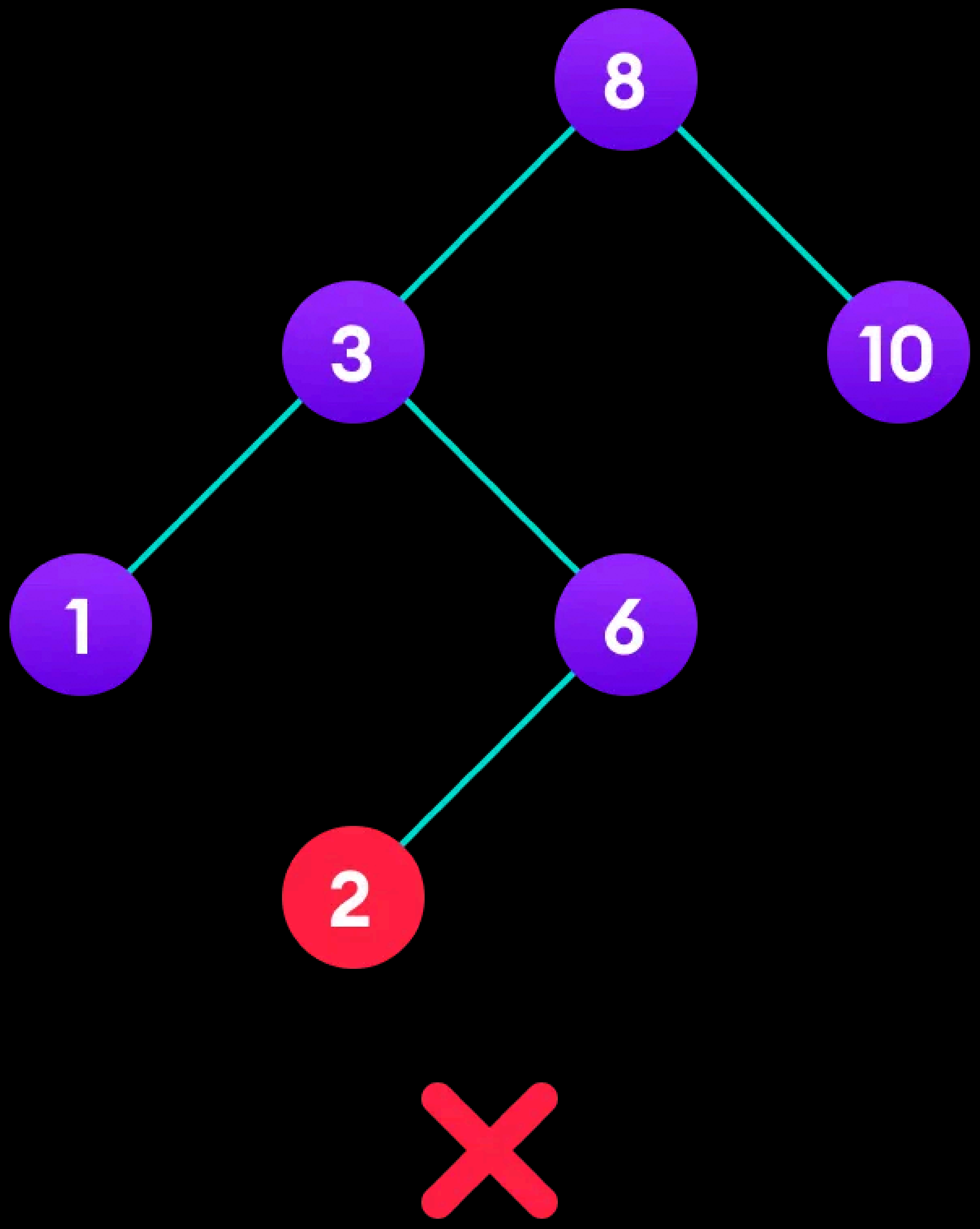
is called a binary tree because each tree node has a maximum of two children.

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

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





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.

Search Operation in BST

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

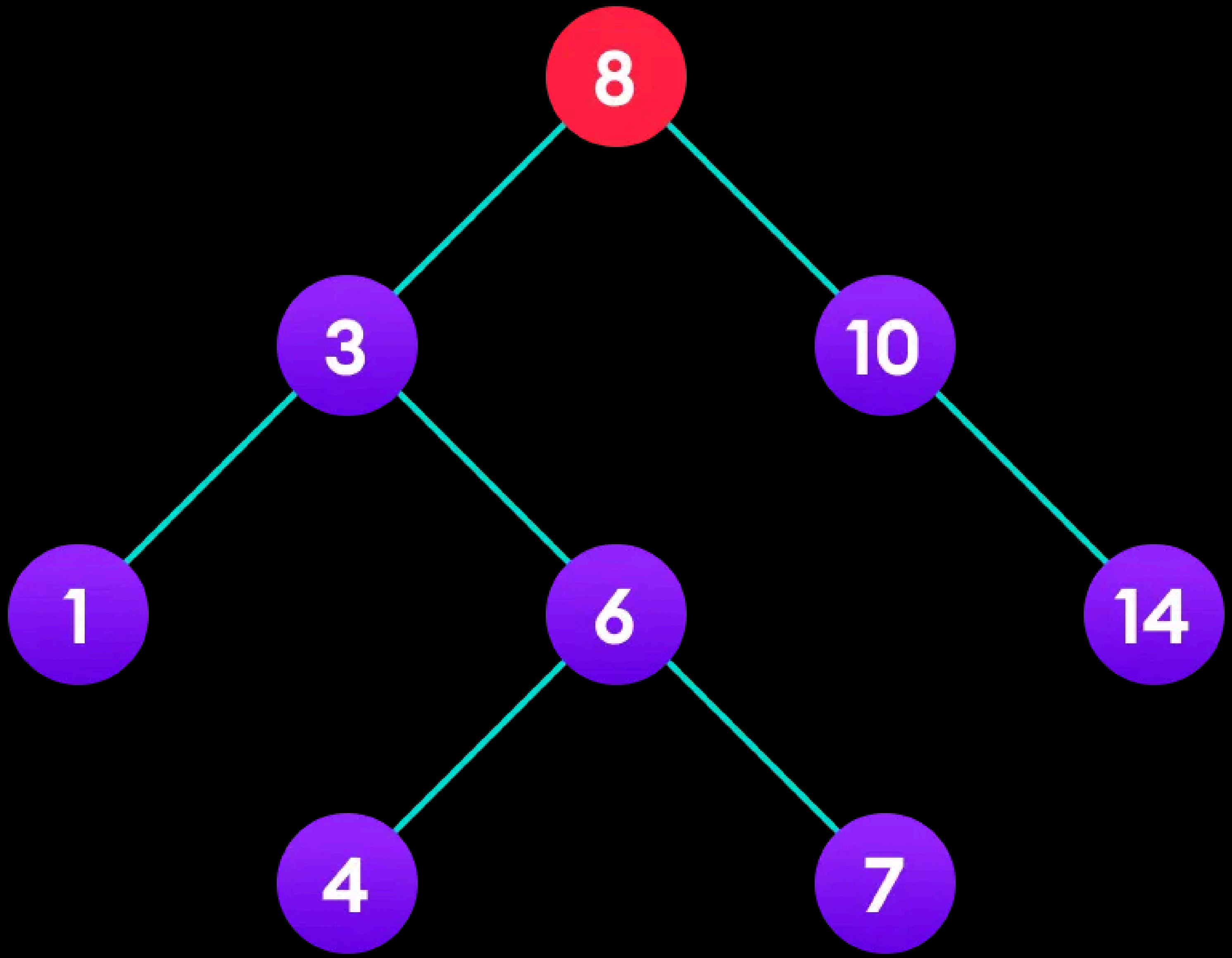
each right subtree has values above the root.

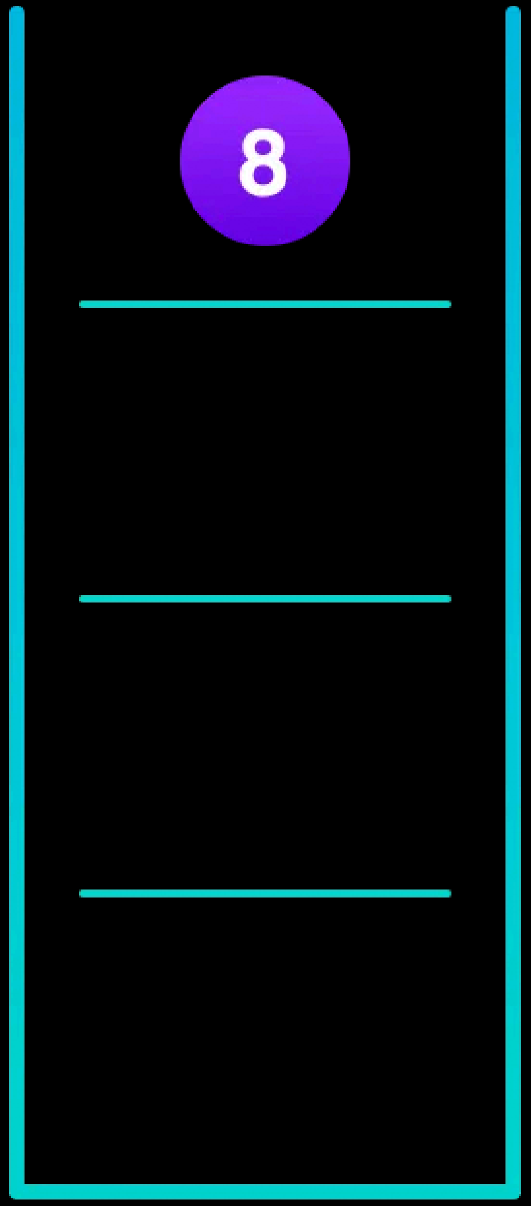
If the value is below the root, we can say for sure that the value is not in the right subtree; we

need to only search in the left subtree and if the value is above the root, we can say for sure that

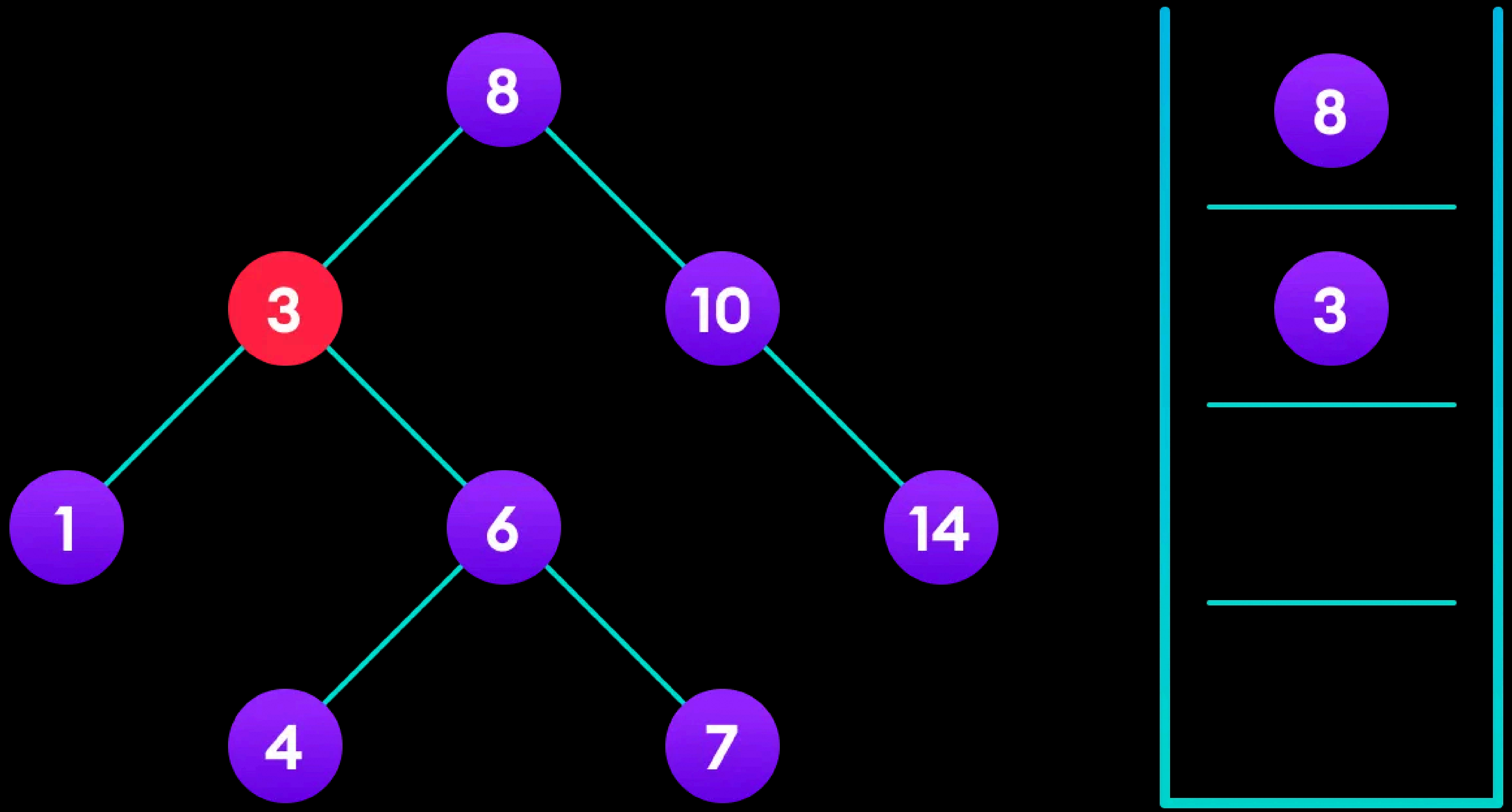
the value is not in the left subtree; we need to only search in the right subtree.

```
If root == NULL  return NULL;
If number == root->data
    return root->data;
If number < root->data
    return search(root->left)
If number > root->data
    return search(root->right)
```

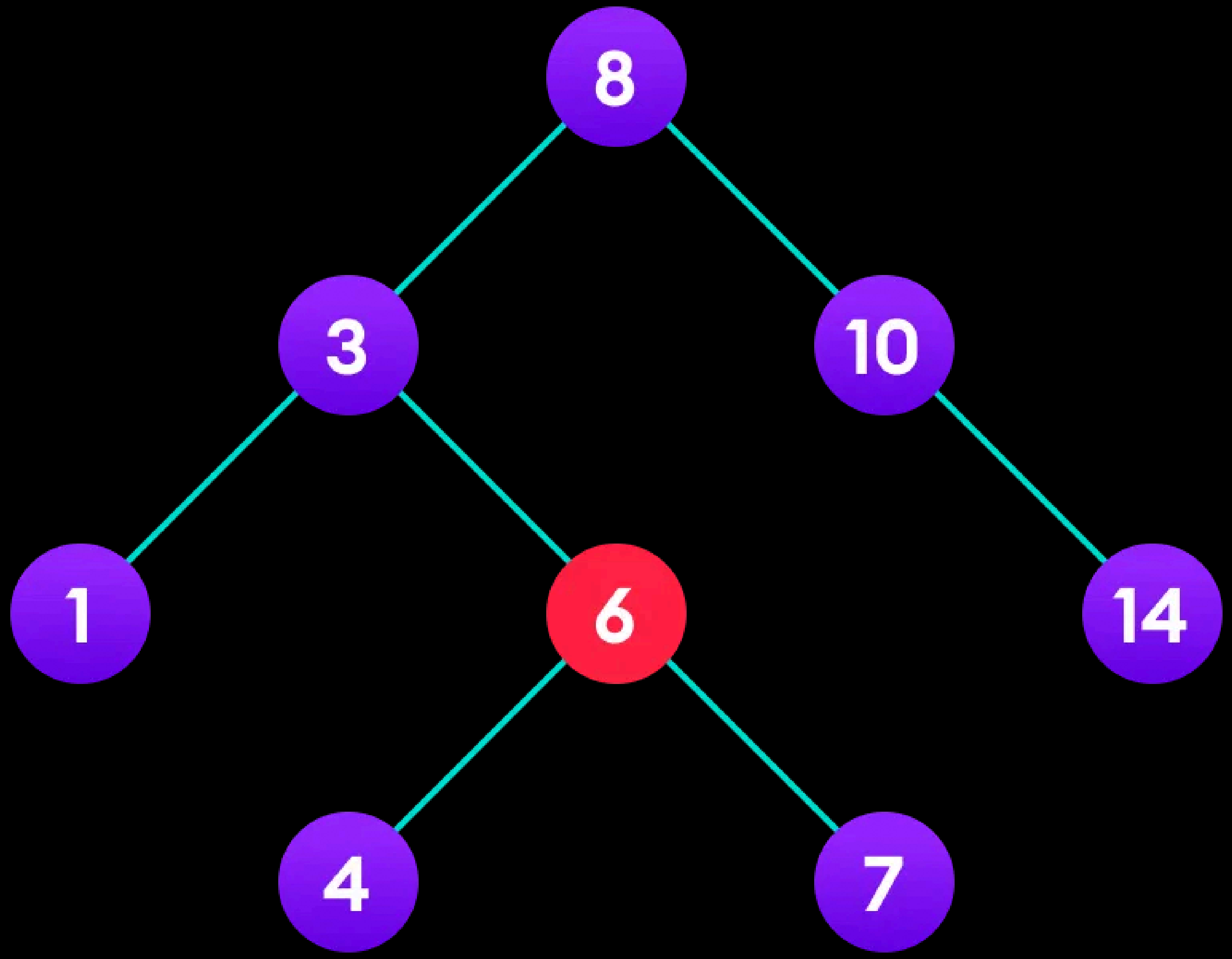


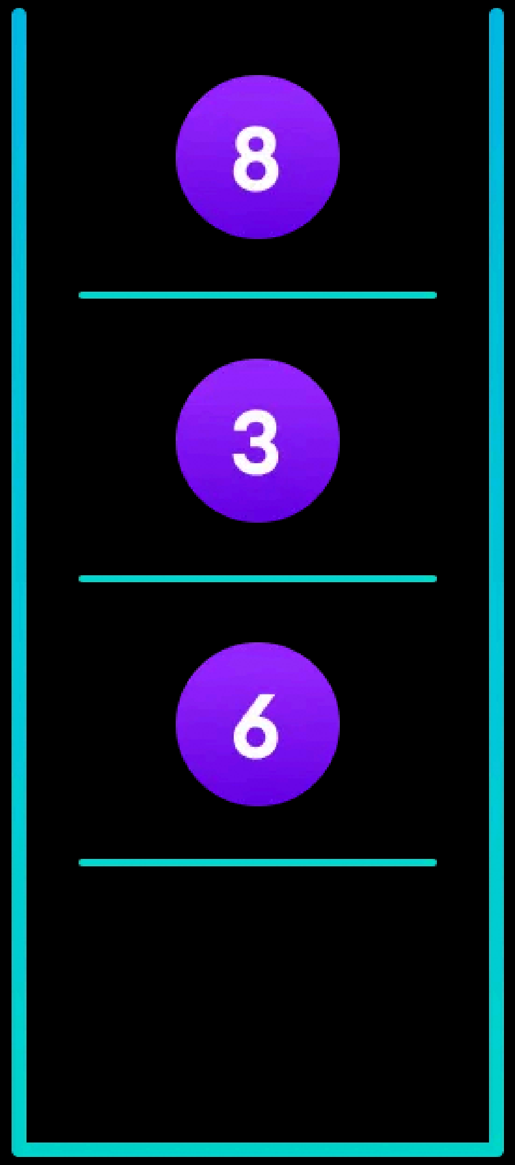


4 is not found so, traverse through the left subtree of 8

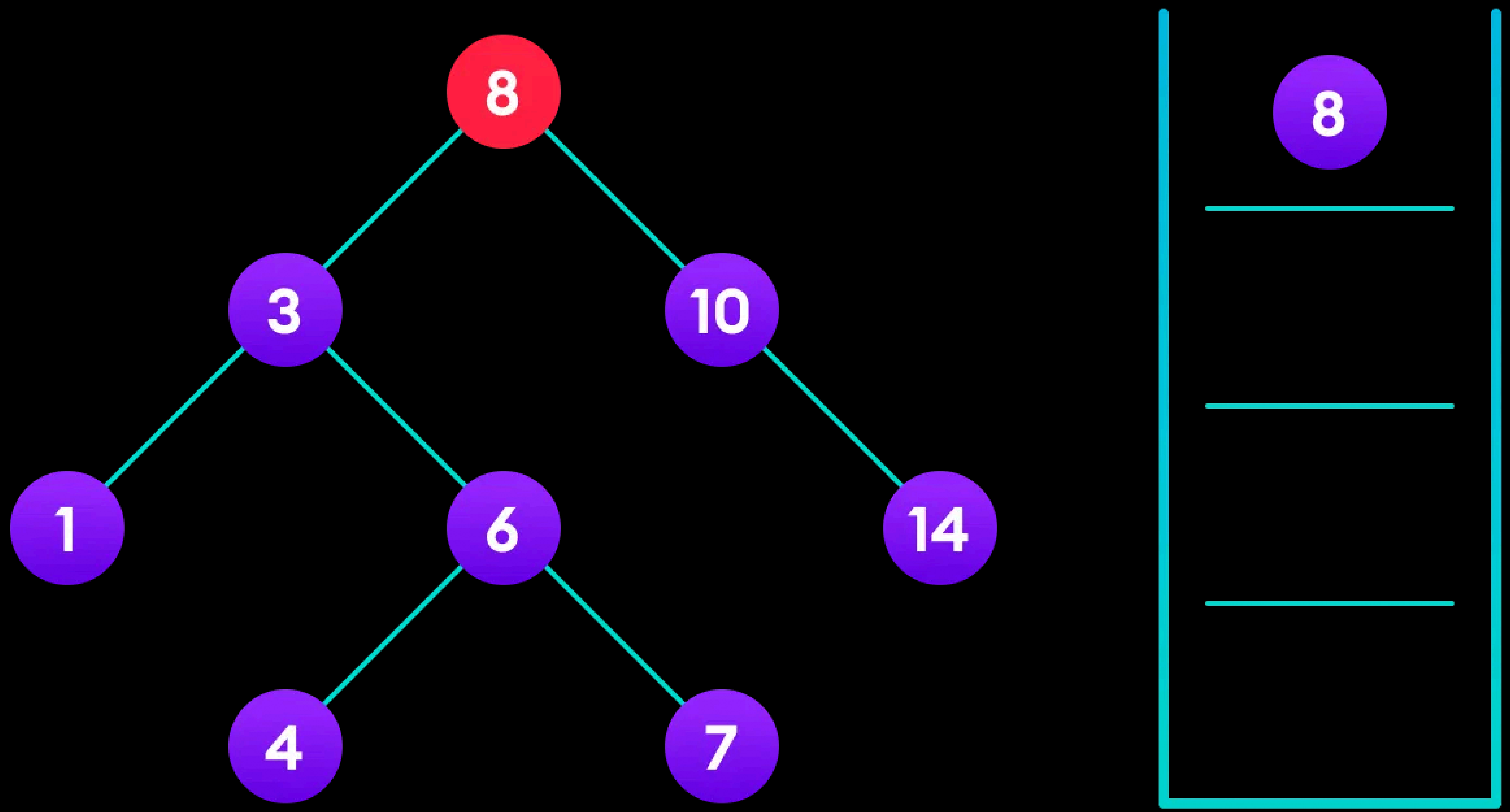


4 is not found so, traverse through the right subtree of 3





4 is not found so, traverse through the left subtree of 6



4 is found

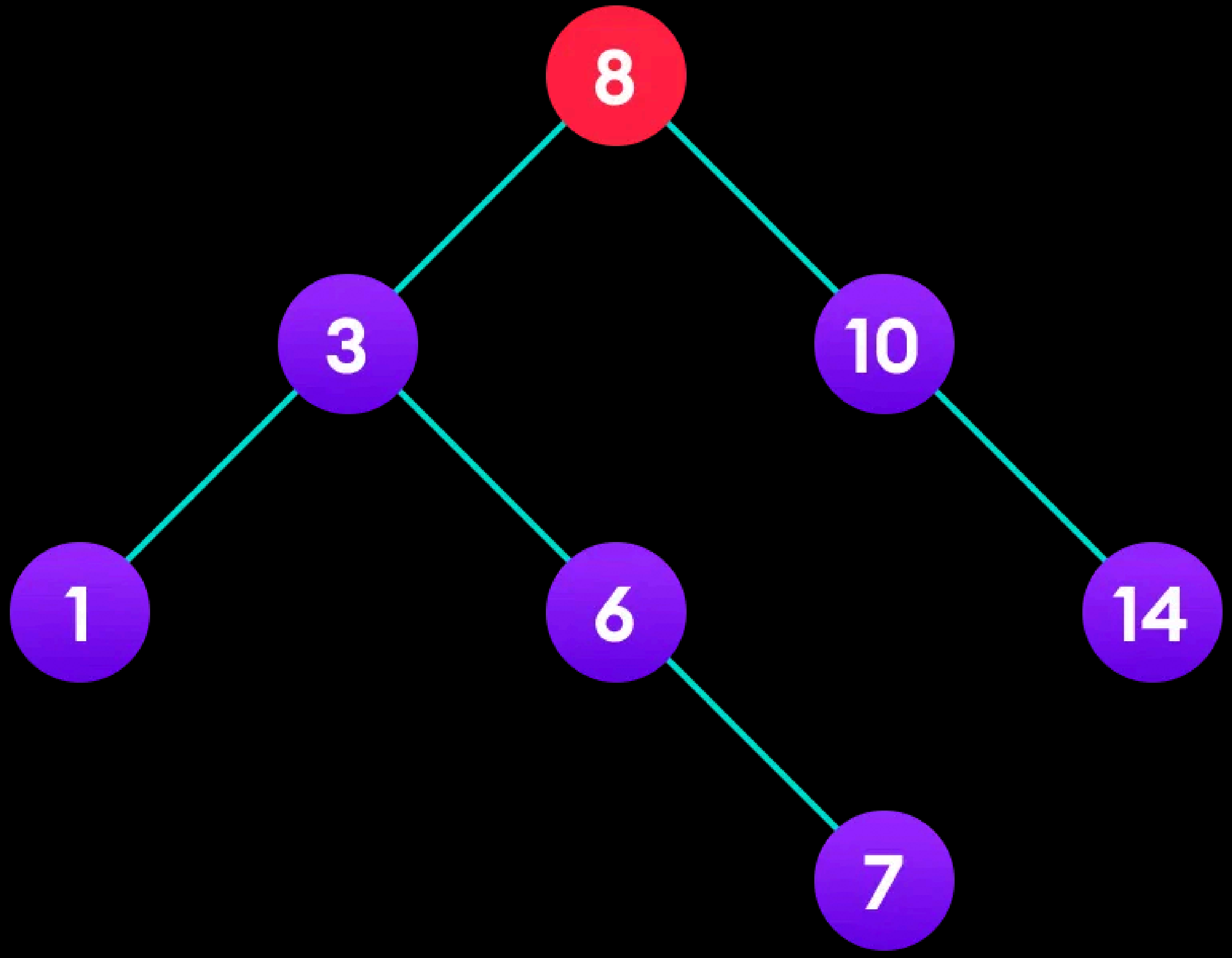
Insert Operation

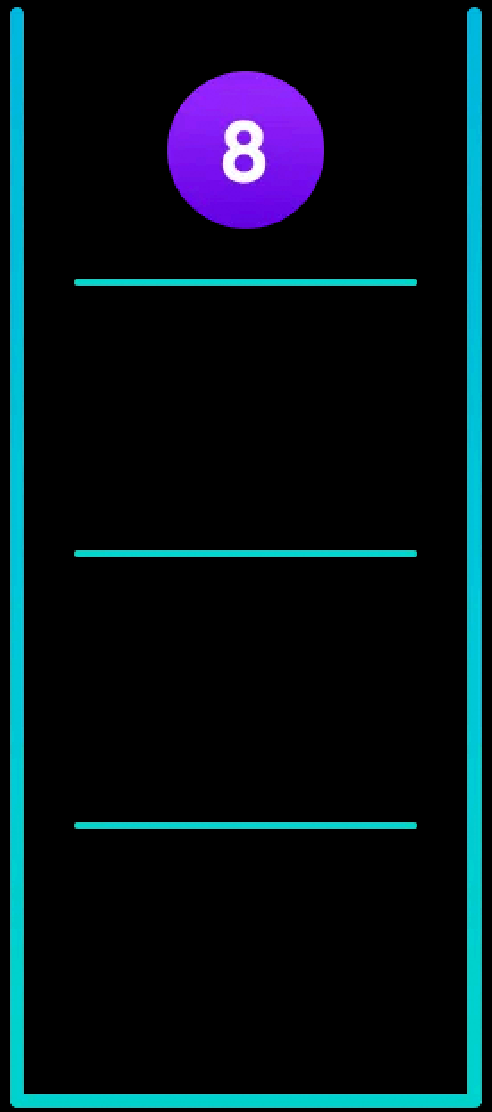
Inserting a value in the correct position is similar to searching because we try to maintain the rule that the left subtree is lesser than root and the right subtree is larger than root.

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.

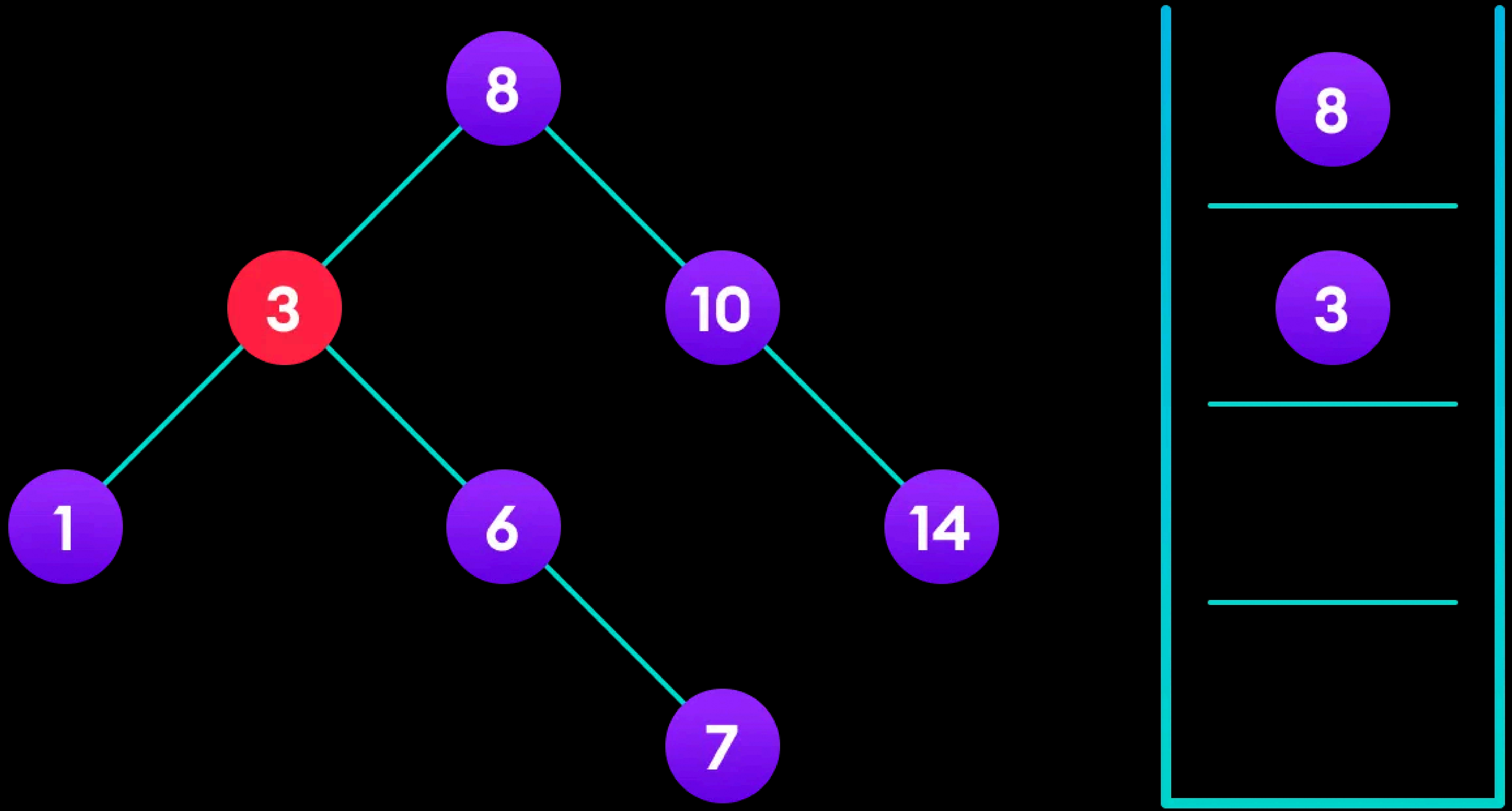
```
If node == NULL
    return createNode(data)
if (data < node->data)
    node->left  = insert(node->left, data);
else if (data > node->data)
    node->right = insert(node->right, data);
return node;
```

The algorithm isn't as simple as it looks. Let's try to visualize how we add a number to an existing BST.

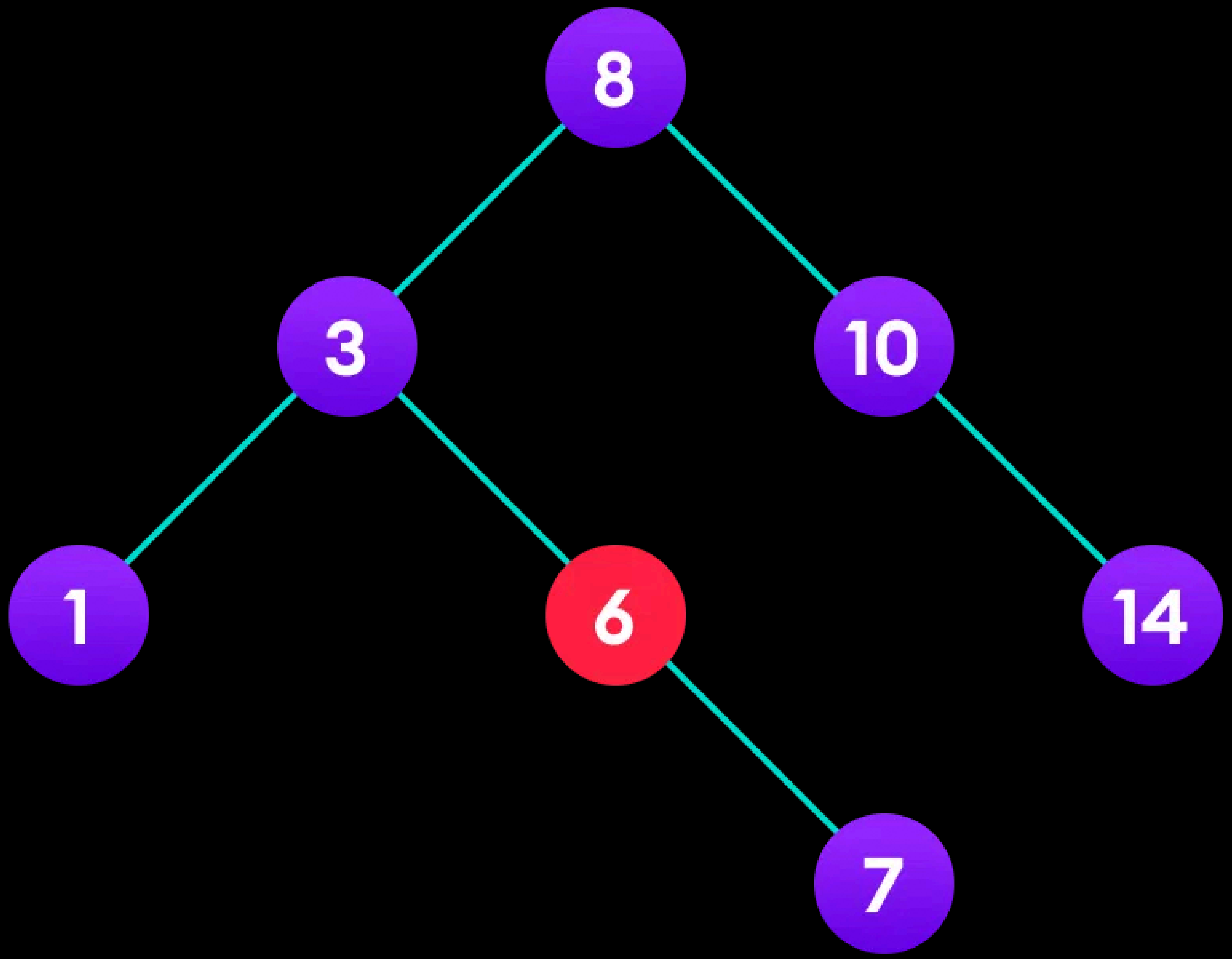


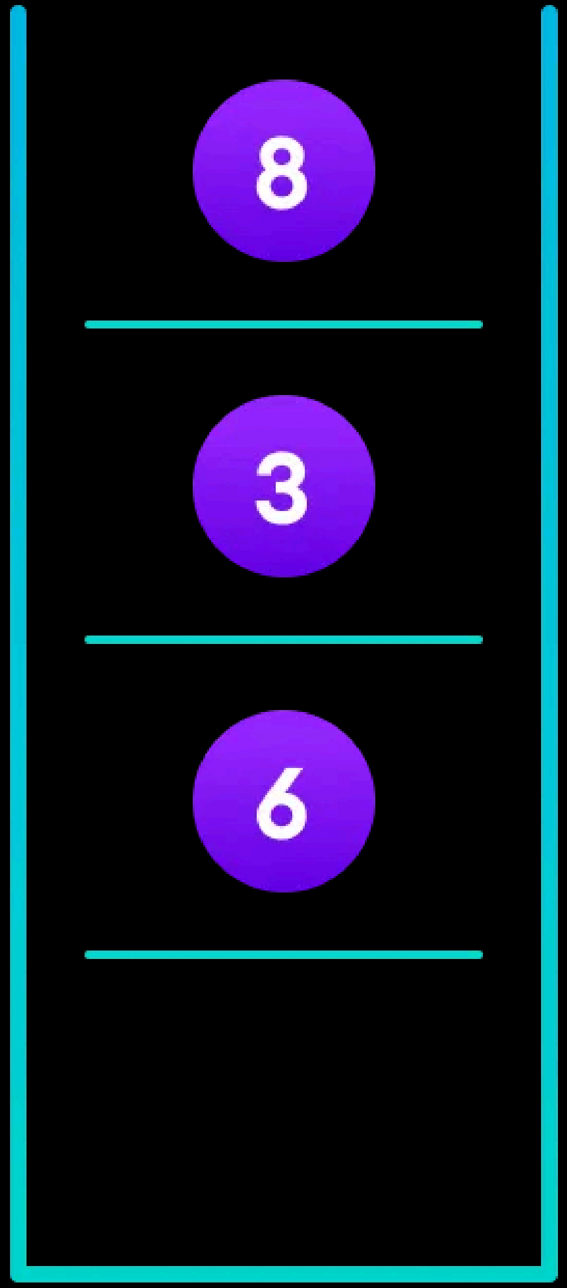


4<8 so, transverse through the left child of 8

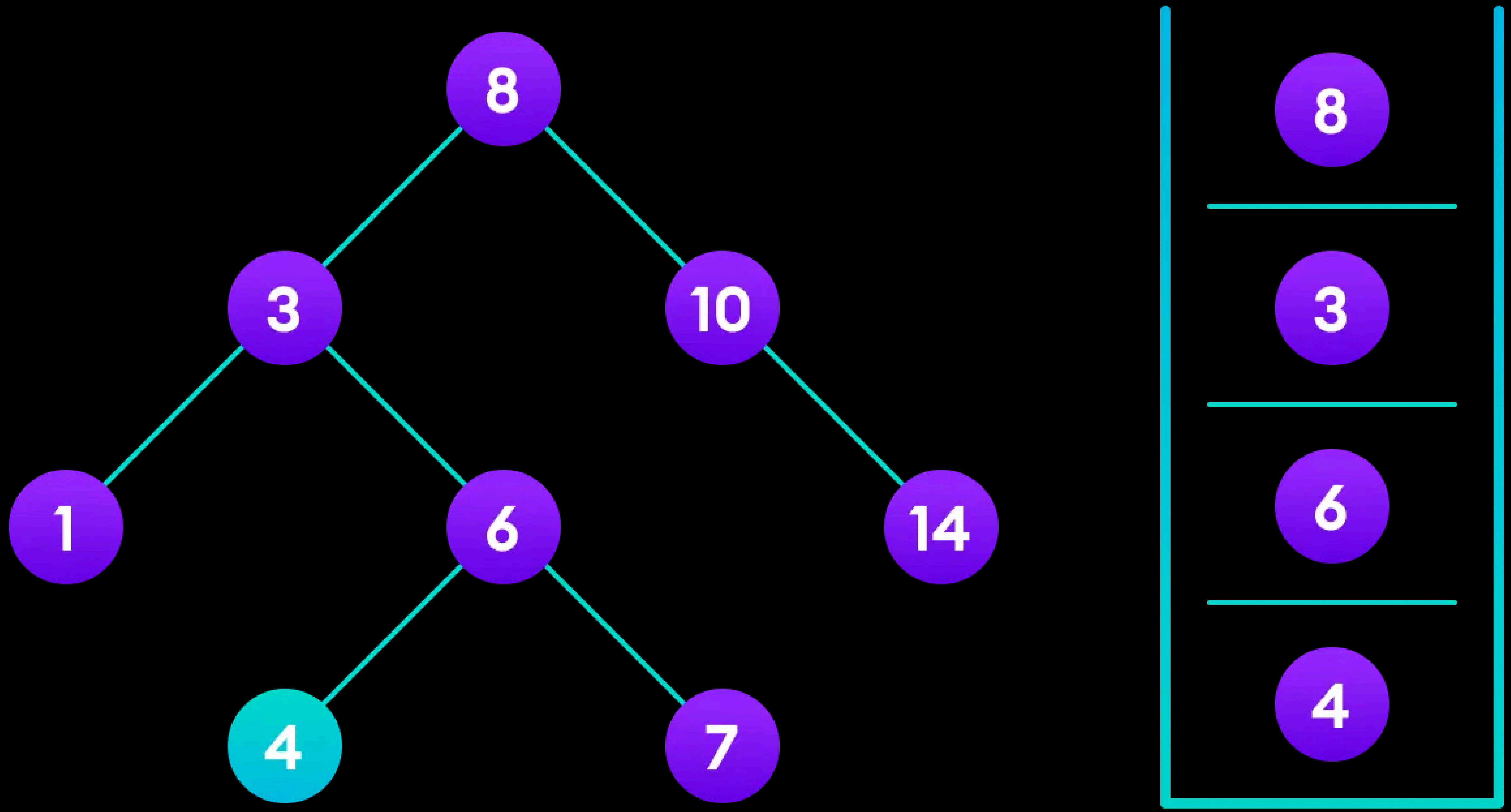


4>3 so, transverse through the right child of 8





4<6 so, transverse through the left child of 6

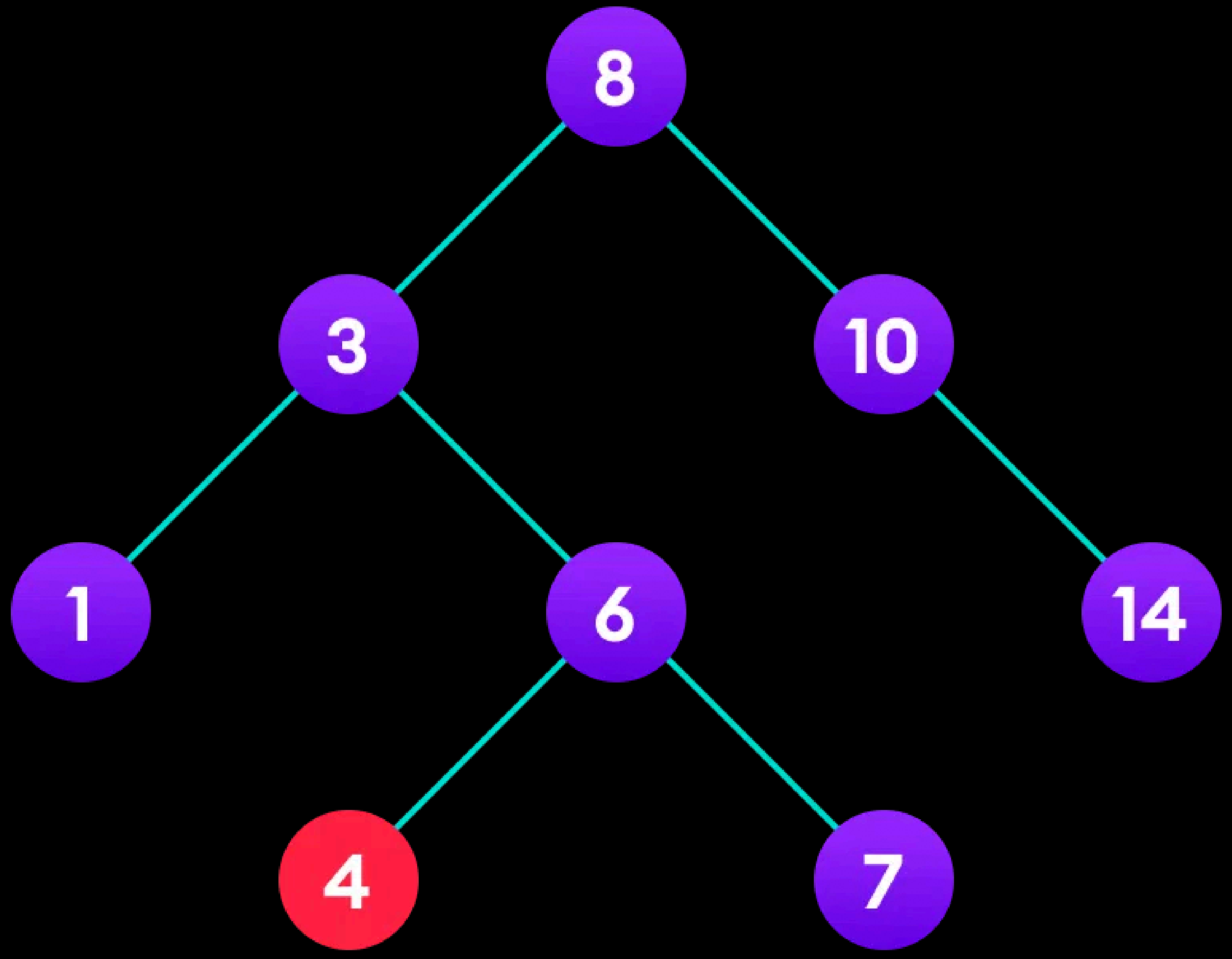


Insert 4 as a left child of 6

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.



4 is to be deleted

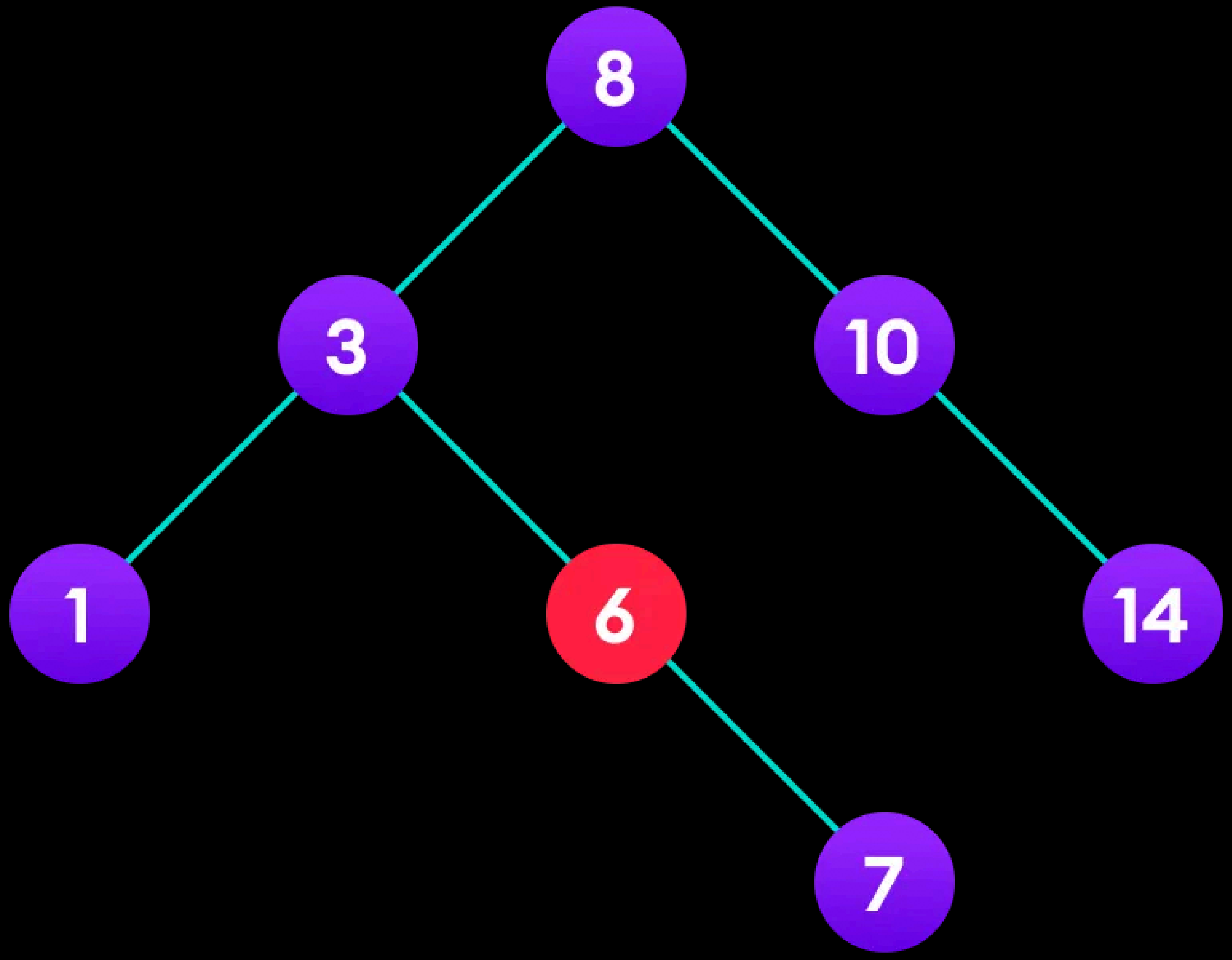
Case II

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

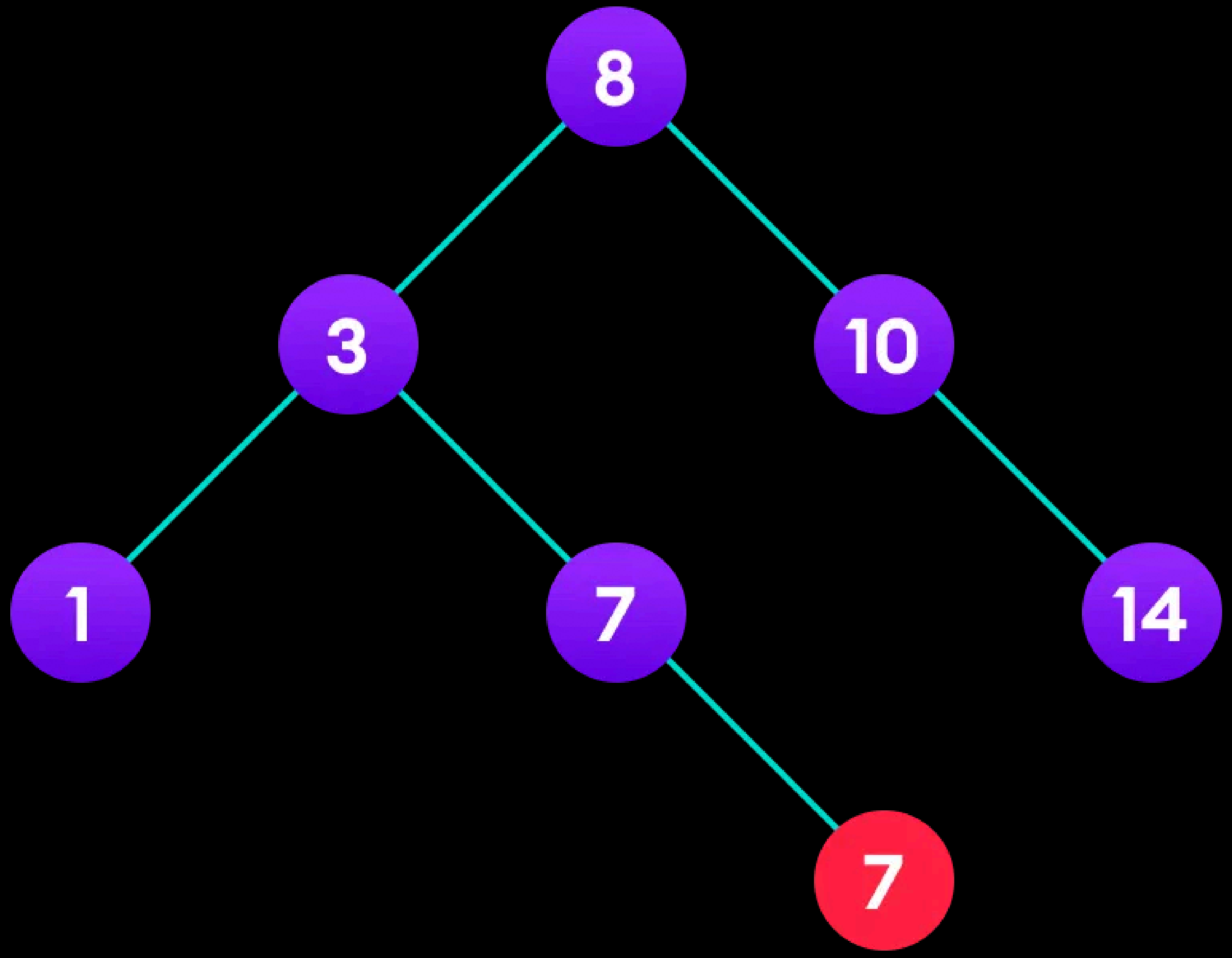
Replace that node with its child node.

Remove the child node from its original position.

6 is to be deleted



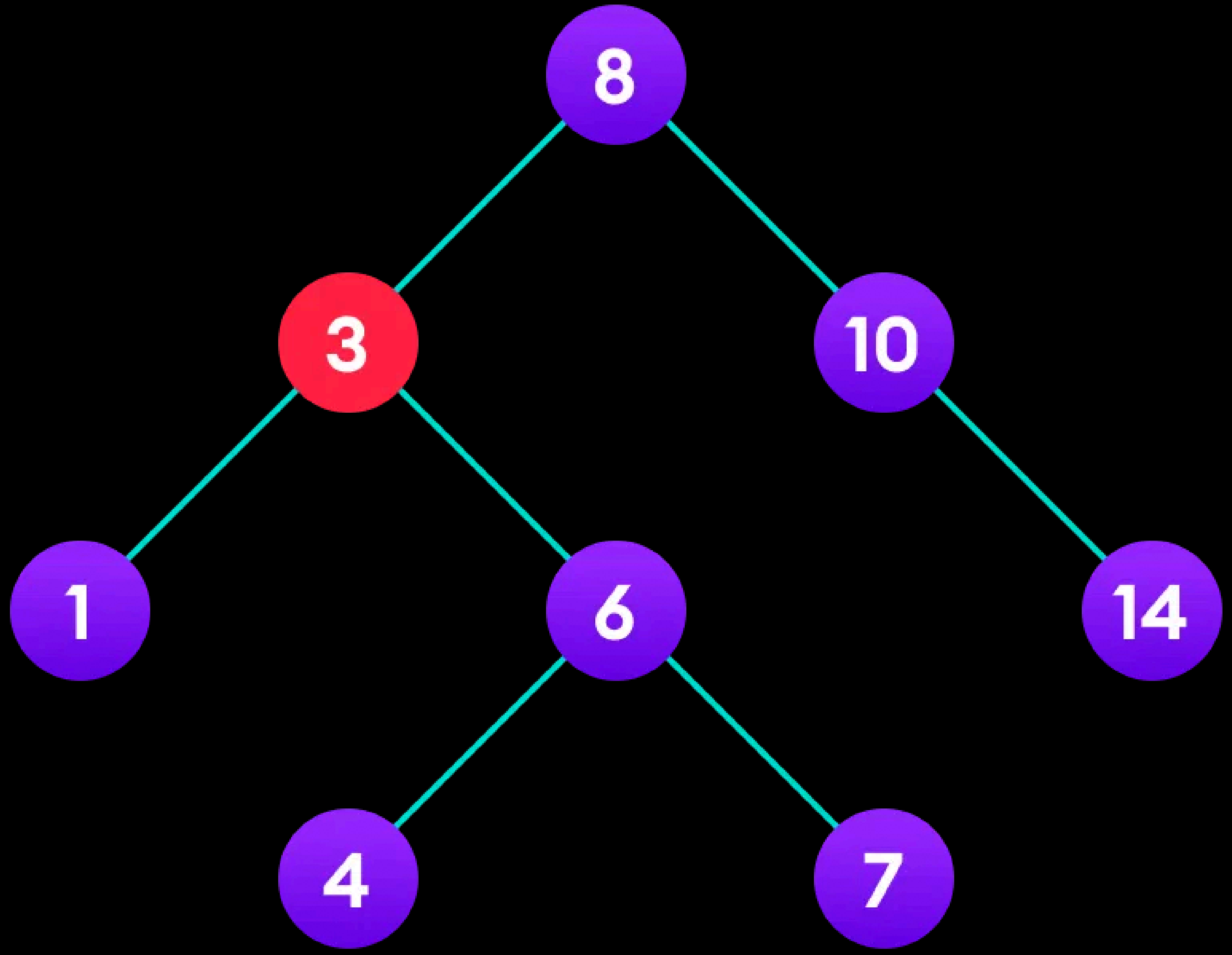
copy the value of its child to the node and delete the child

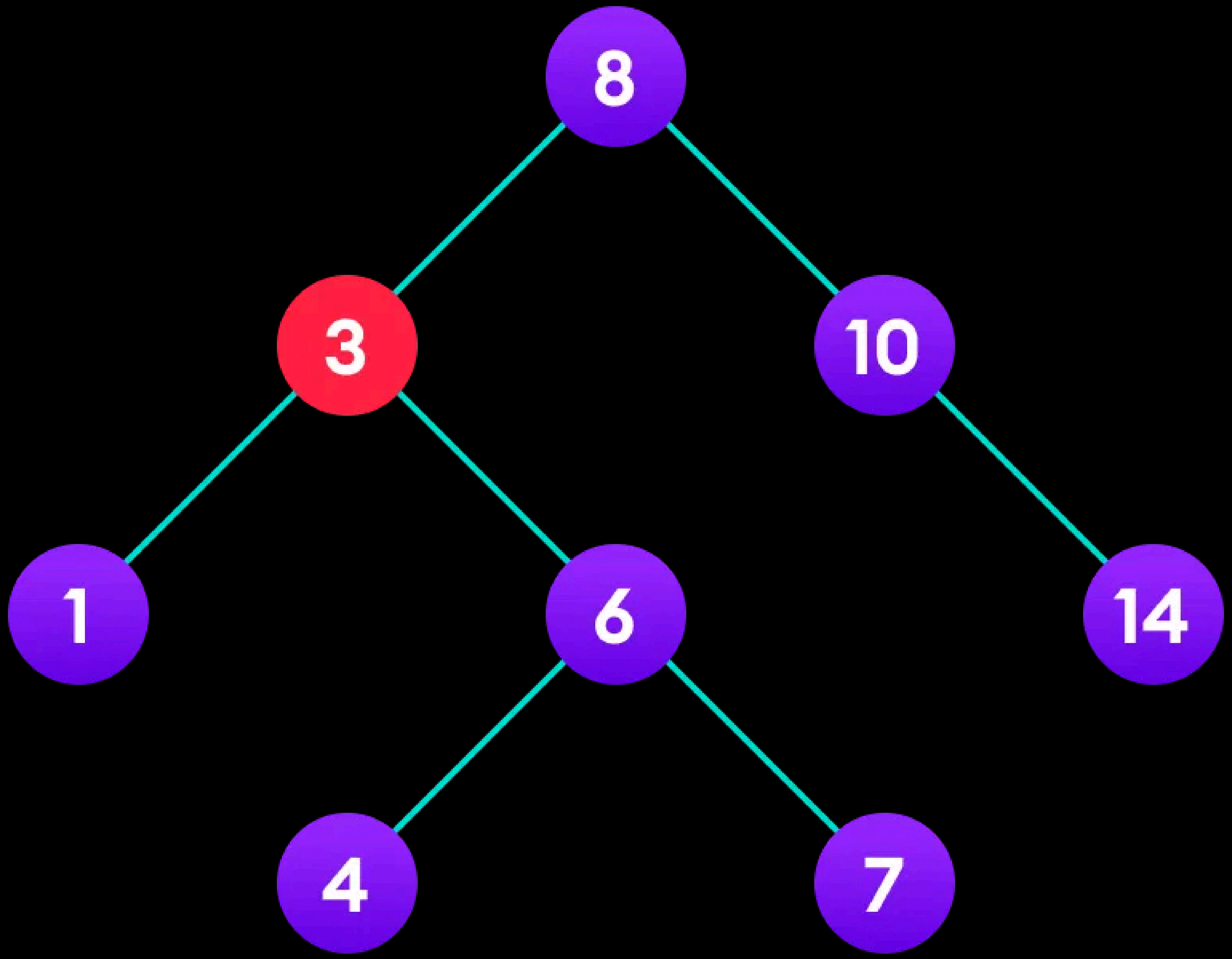


Case III
In the third case, the node to be deleted has two children. In such a case follow the steps below:

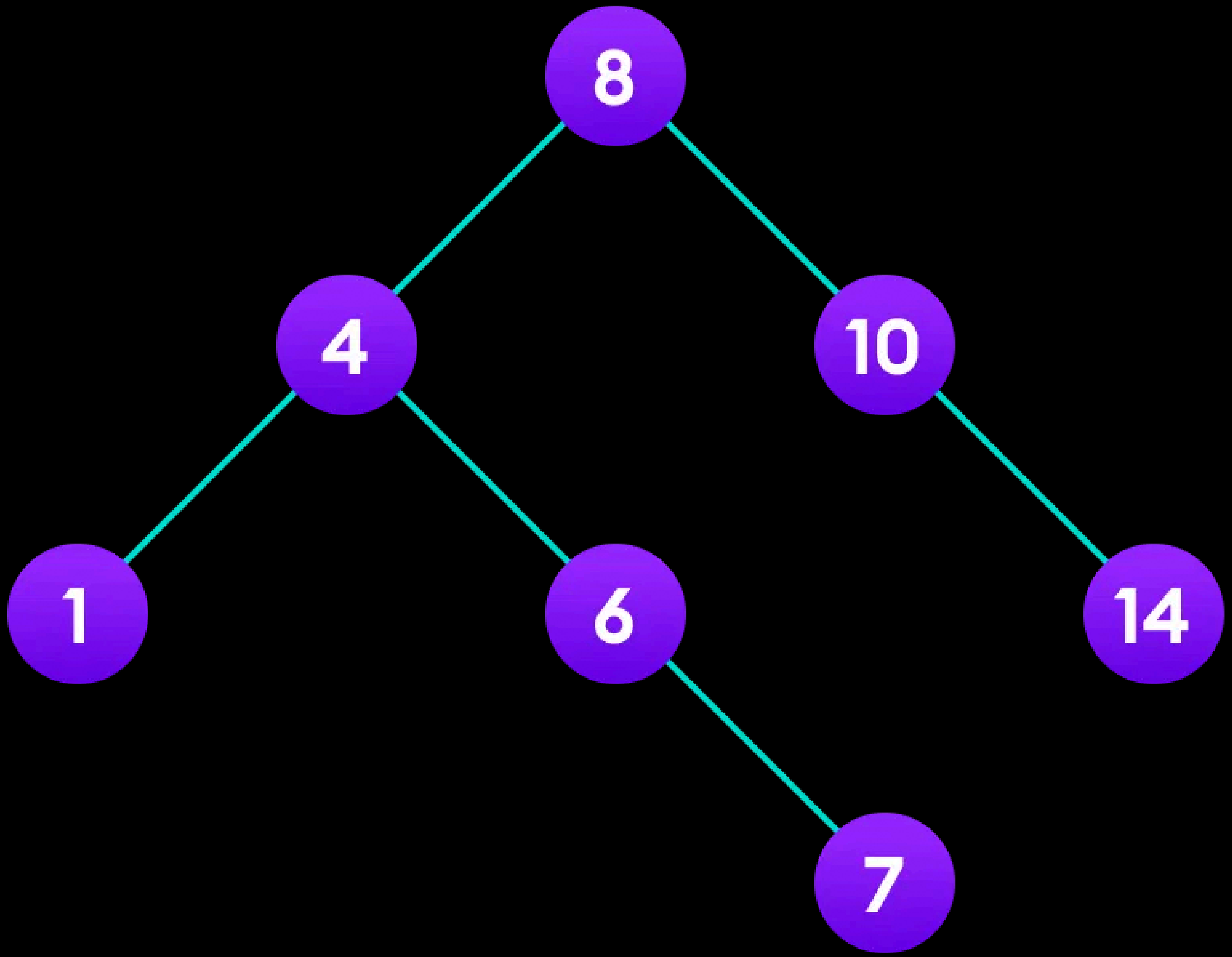
Get the inorder successor of that node.
Replace the node with the inorder successor.
Remove the inorder successor from its original position.

3 is to be deleted





Copy the value of the inorder successor (4) to the node



Delete the inorder successor

Thanks

All code and examples can be found on GitHub https://github.com/ESSAMMOHAMED1