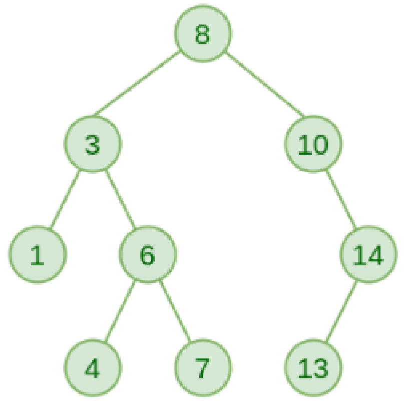
# Binary Search Tree

- Used for organizing and storing data in a sorted manner.
- Each node in a **Binary Search Tree** has at most two children, a **left** child and a **right** child.
- **left** child contain values less than the parent node and the **right** child contain values greater than the parent node.

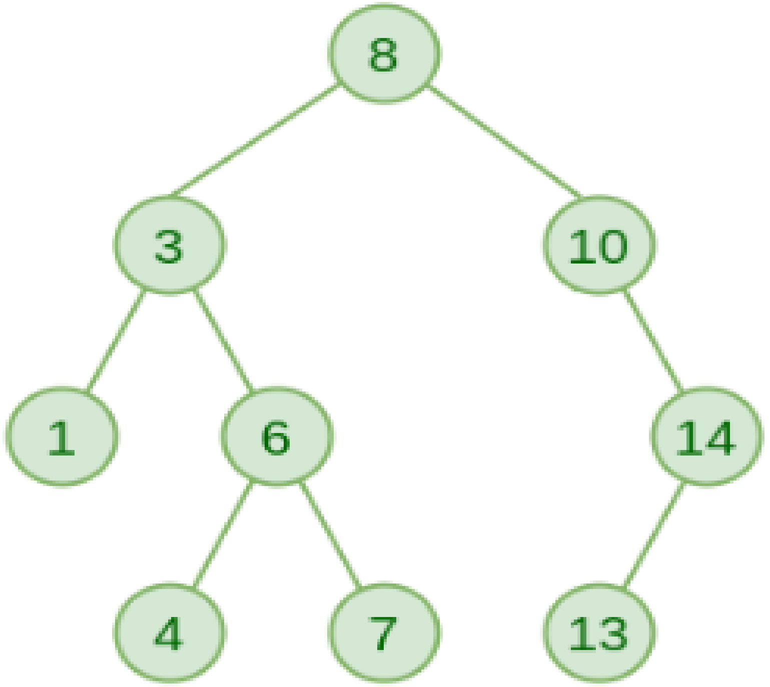


## Traversal in Binary Search Tree

## Search in Binary Search Tree

- Let’s say we want to search for the number **X,** We start at the root. Then:
- We compare the value to be searched with the value of the root.
  - If it’s equal we are done with the search if it’s smaller we know that we need to go to the left subtree because in a binary search tree all the elements in the left subtree are smaller and all the elements in the right subtree are larger.
- Repeat the above step till no more traversal is possible
- If at any iteration, key is found, return True. Else False.

## Example



Search(7)
Search(20)

# Insertion in Binary Search Tree

- A new key is always inserted at the leaf by maintaining the property of the binary search tree.
- Steps:
  1. Initilize the current node (say, **currNode or node**) with root node
  2. Compare the **key** with the current node.
  3. **Move left** if the **key** is less than or equal to the current node value.
  4. **Move right** if the **key** is greater than current node value.
  5. Repeat steps 3 and 4 until you reach a leaf node.
  6. Attach the **new key** as a left or right child based on the comparison with the leaf node’s value.

- 10,20,15,50,25,5,60,55

## Deletion in BST