Insertion

In this lesson, we'll see how to insert a new key in a BST.

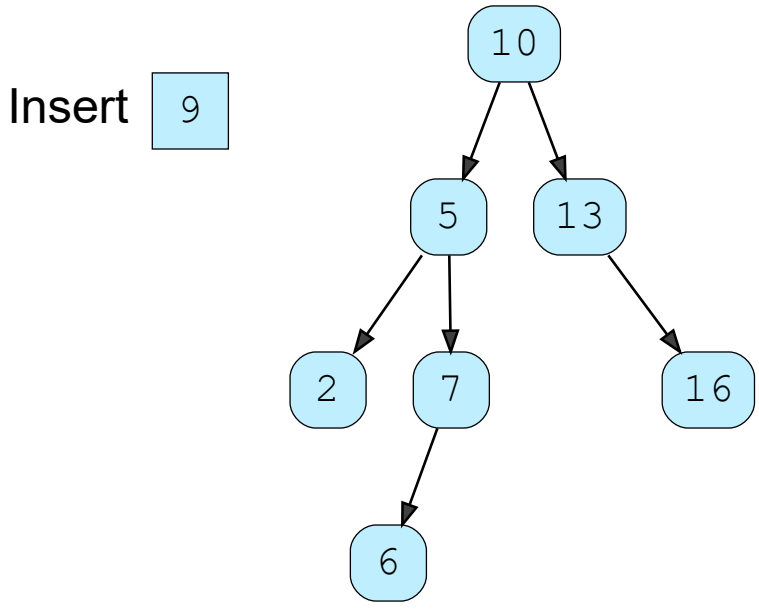
We'll cover the following

Algorithm

A new key is always inserted as a leaf. We search for the location for a new key as mentioned in the previous lesson and insert the new node when we reach the end.

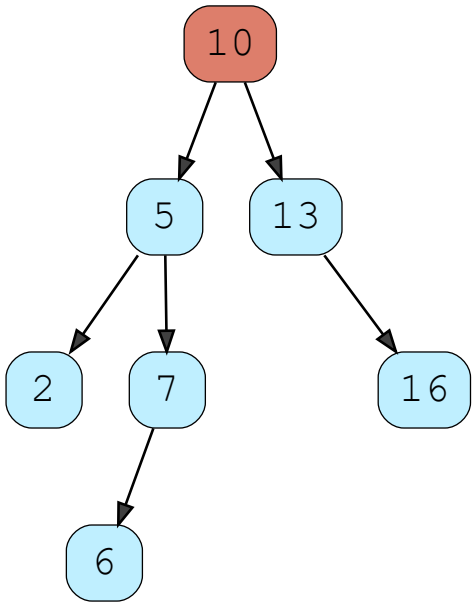
Time Complexity: In the worst case, we traverse to the deepest node. So insertion time complexity is $O(height)$ or $O(N)$.

Visualization



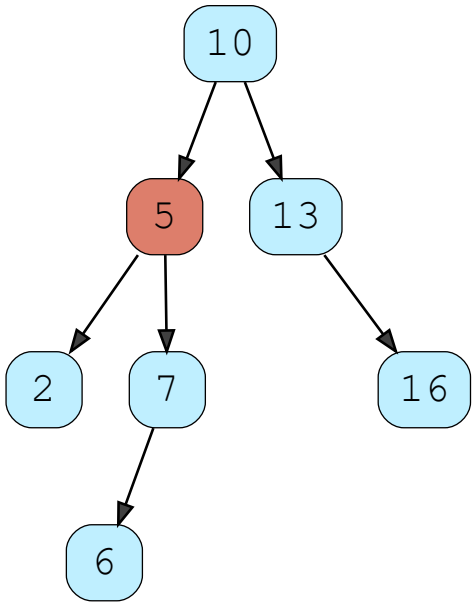
Insert 9

Insert 9



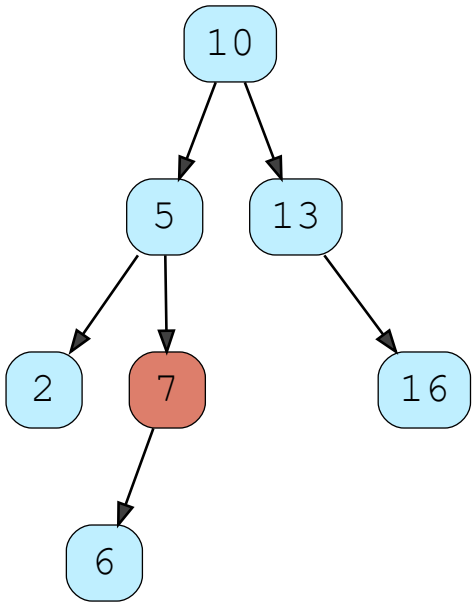
Compare with 10. 9 < 10 => go left

Insert 9



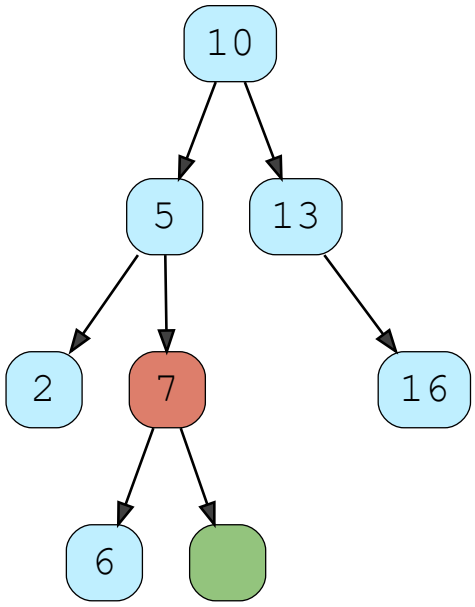
Compare with 5. 9 > 5 => go right

Insert 9

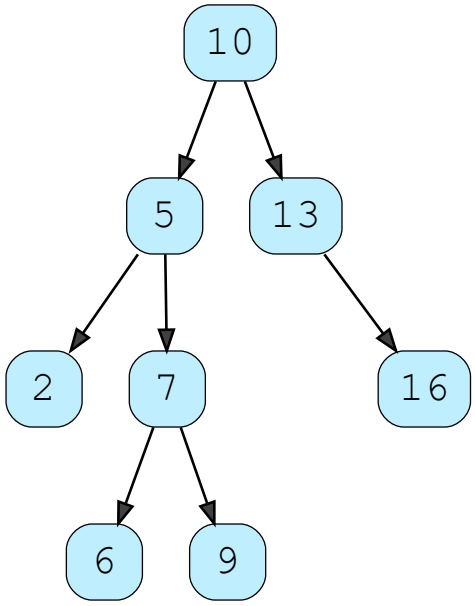


Compare with 7. 9 > 7 => go right

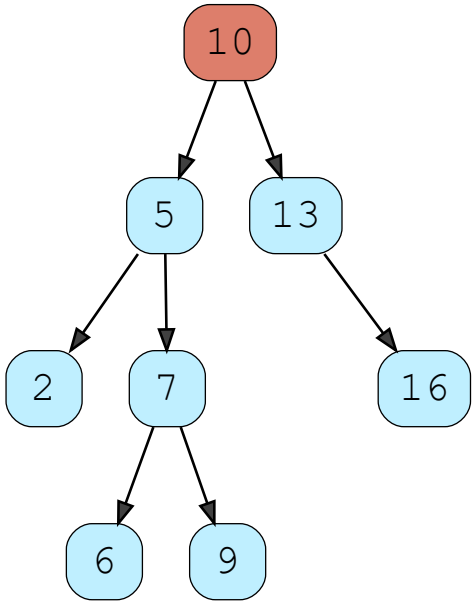
Insert 9



7-> right is NULL. Add new node here

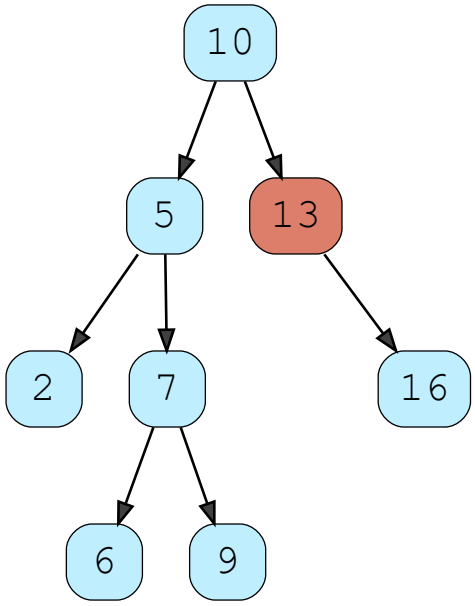


Insert 14



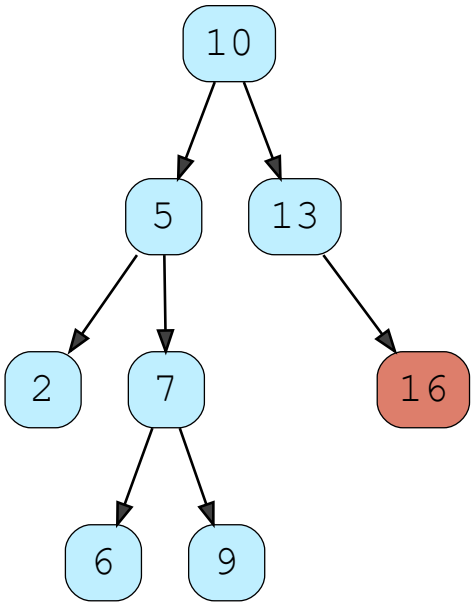
Compare with 10, 14 > 10, go right

Insert 14



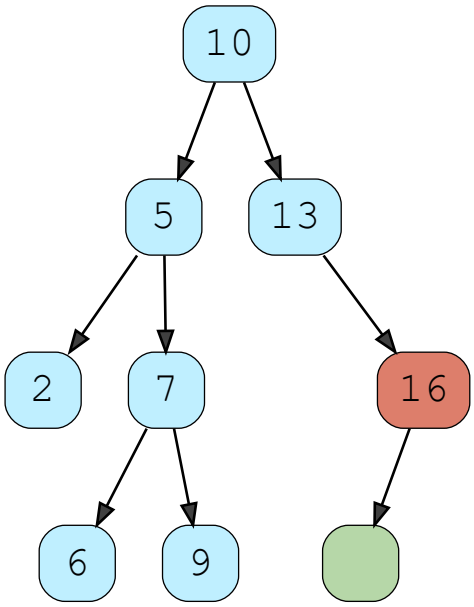
Compare with 13, 14 > 13, go right

Insert 14



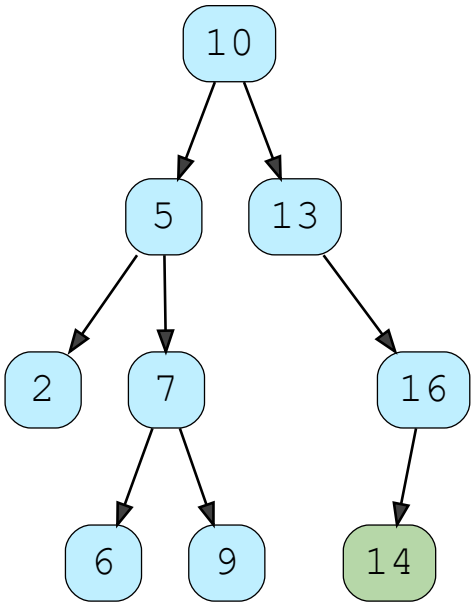
Compare with 16, 14 < 16, go left

Insert 14

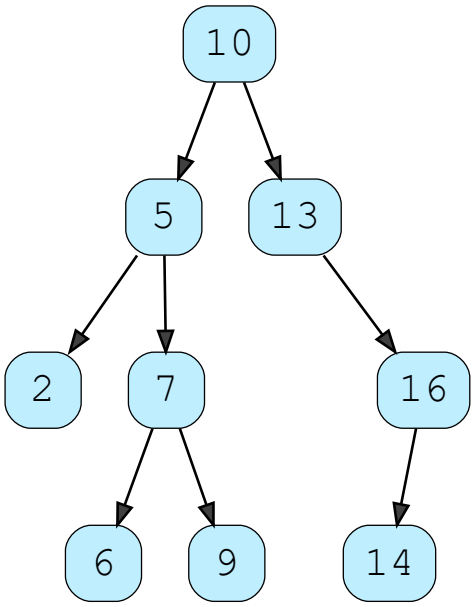


Compare with 16, 14 < 16, go left

Insert 14



Compare with 16, 14 < 16, go left



Compare with 16, 14 < 16, go left

12 of 12

Code

```
void insert(struct Node* &root, int val) {
    if (root == NULL) {
      root = new Node(val);
      return;
    }

    Node* pCrawl = root;
    Node* pCrawlParent;
    while(pCrawl) {
      pCrawlParent = pCrawl;

      if (val < pCrawl->val)
        pCrawl = pCrawl->left;
      else
        pCrawl = pCrawl->right;
    }

    if (val < pCrawlParent->val)
      pCrawlParent->left = new Node(val);
    else
      pCrawlParent->right = new Node(val);
}
```

In the next lesson, we'll see different traversals of a BST and its properties.