
# Data Structures & Algorithms

# Tree

# Binary Search Tree

# Searching in a BST

```
bst_search1(root, item)
{
   ptr = root;
   while (ptr != NULL)  {
      if (item == ptr->data)
         return 1;   // Item present, successful search
      if (item < ptr->data)
         ptr = ptr->left;
      else
         // item > ptr->data
         ptr = ptr->right;
   }
   return 0; // Item does not present, unsuccessful search
}
```

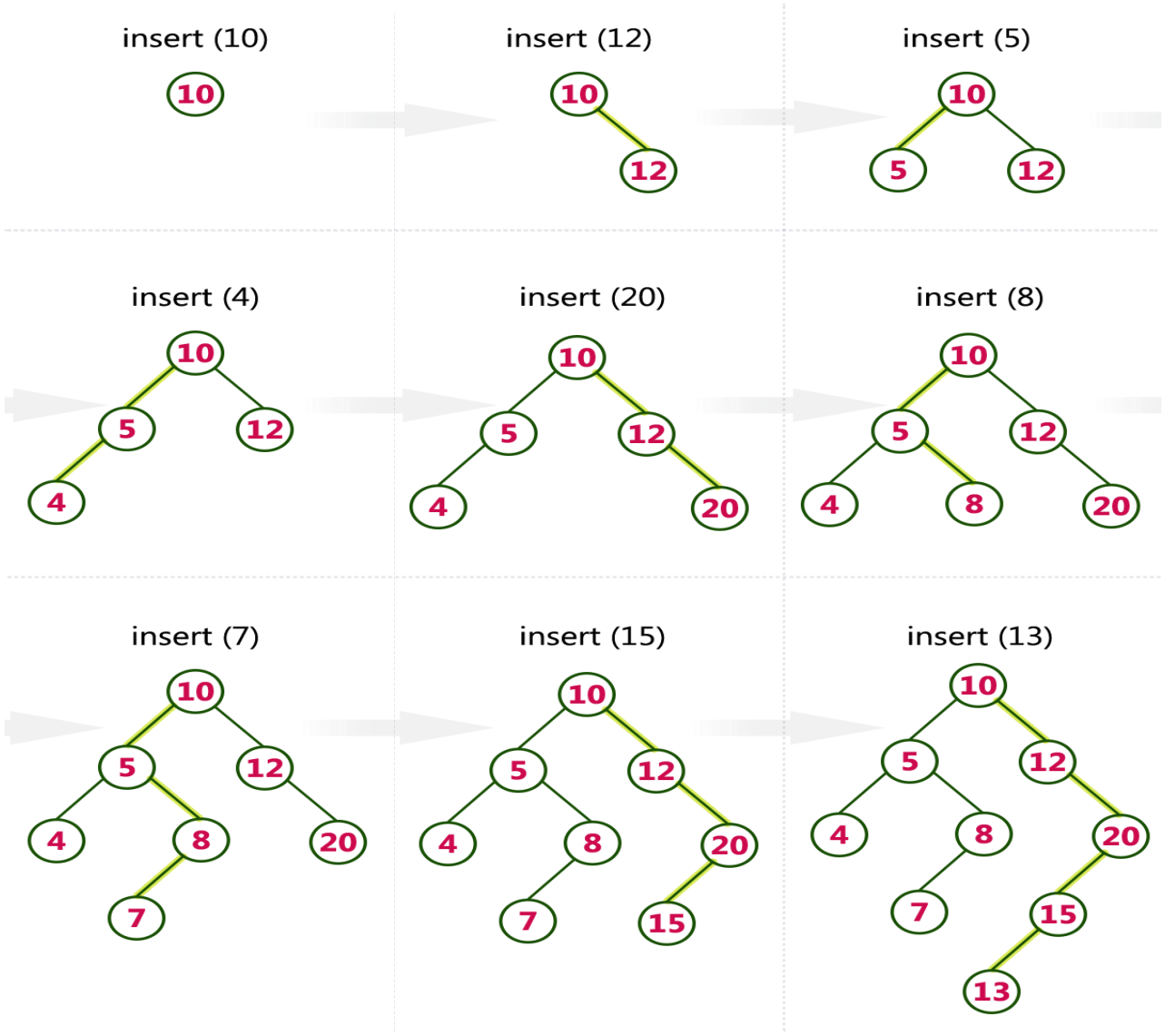
# Searching in a BST

```
bst_search2(root, item)
{
  if (root == NULL)
     return 0;
  if (item == root->data)
     return 1;
  if (item < root->data)
     return bst_search2(root->left , item);
  else // item > root->data
     return bst_search2(root->right, item);
}
```

**Time complexity : O(n) in the worst case**

# Insertion in a BST

Start with an empty binary search tree. Insert the following items, in the given order: 10, 12, 5, 4, 20, 8, 7, 15, 13



# Insertion in a BST

We assume that the elements of a BST is unique, so, the new 'item' to be inserted does not exist in the BST

```
insertBST(root, item) {
  if (root == NULL) {
     new_node = (BTnode *) malloc(sizeof(BTnode));
     new_node ->data = item;
     new_node ->left = NULL;
     new_node ->right = NULL;
     root = new_node;
     return root;
  }
  else if (item < root->data)
            insertBST(root->left, item);  // Set the new root of the subtree.
        else  // item > root->data
            insertBST(root->right, item);
}
```

**Time complexity : O(n) in the worst case**