<https://www.javatpoint.com/binary-tree-vs-binary-search-tree>

Binary tree vs Binary Search tree

First, we will understand the ***binary tree*** and ***binary search tree*** separately, and then we will look at the differences between a binary tree and a binary search tree.

What is a Binary tree?

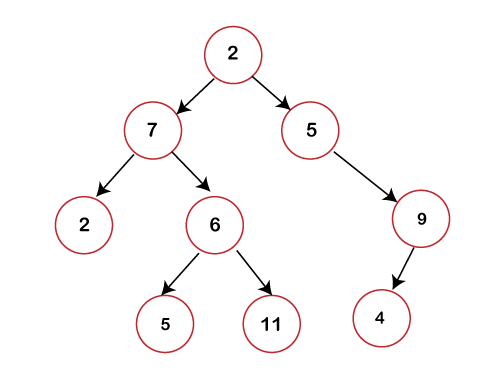
A [Binary tree](https://www.javatpoint.com/binary-tree) is a [non-linear](https://www.javatpoint.com/linear-vs-non-linear-data-structure) data structure in which a node can have either **0, 1** or **maximum 2 nodes**. Each node in a binary tree is represented either as a parent node or a child node. There can be two children of the parent node, i.e., **left child** and **right child**.

There is only one way to reach from one node to its next node in a binary tree.

**A node in a binary tree has three fields:**

* **Pointer to the left child:** It stores the reference of the left-child node.
* **Pointer to the right child:** It stores the reference of the right-child node.
* **Data element:** The data element is the value of the data which is stored by the node.

**The binary tree can be represented as:**



In the above figure, we can observe that each node contains utmost 2 children. If any node does not contain left or right child then the value of the pointer with respect to that child would be NULL.

**Basic terminologies used in a Binary tree are:**

* **Root node:** The root node is the first or the topmost node in a binary tree.
* **Parent node:** When a node is connected to another node through edges, then that node is known as a parent node. In a binary tree, parent node can have a maximum of 2 children.
* **Child node:** If a node has its predecessor, then that node is known as a ***child node***.
* **Leaf node:** The node which does not contain any child known as a ***leaf node***.
* **Internal node:** The node that has at least 2 children known as an ***internal node***.
* **Depth of a node:** The distance from the root node to the given node is known as a ***depth of a node***. We provide labels to all the nodes like root node is labeled with 0 as it has no depth, children of the root nodes are labeled with 1, children of the root child are labeled with 2.
* **Height:** The longest distance from the root node to the leaf node is the ***height of the node***.

In a binary tree, there is one tree known as a ***perfect binary tree***. It is a [tree](https://www.javatpoint.com/tree) in which all the internal nodes must contain two nodes, and all the leaf nodes must be at the same depth. In the case of a perfect binary tree, the total number of nodes exists in a binary tree can be calculated by using the following equation:

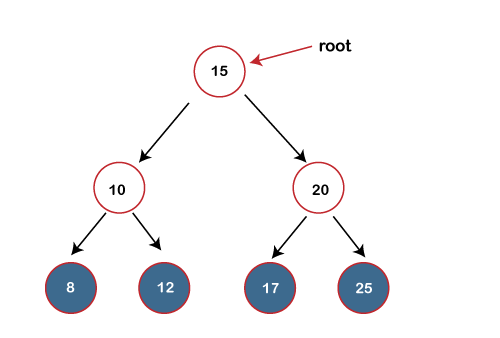
**n = 2m+1-1**

**Where n is the number of nodes, m is the depth of a node.**

What is a Binary Search tree?

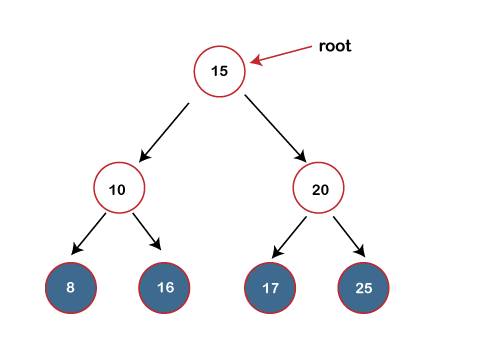
A [Binary search tree](https://www.javatpoint.com/binary-search-tree) is a tree that follows some order to arrange the elements, whereas the binary tree does not follow any order. In a Binary search tree, the value of the left node must be smaller than the parent node, and the value of the right node must be greater than the parent node.

**Let's understand the concept of a binary search tree through examples.**



In the above figure, we can observe that the value of the root node is 15, which is greater than the value of all the nodes in the left subtree. The value of root node is less than the values of all the nodes in a right subtree. Now, we move to the left-child of the root node. 10 is greater than 8 and lesser than 12; it also satisfies the property of the Binary search tree. Now, we move to the right-child of the root node; the value 20 is greater than 17 and lesser than 25; it also satisfies the property of binary search tree. Therefore, we can say that the tree shown above is the binary search tree.

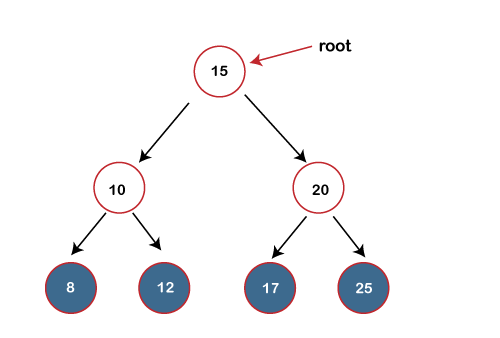
Now, if we change the value of 12 to 16 in the above binary tree, we have to find whether it is still a binary search tree or not.



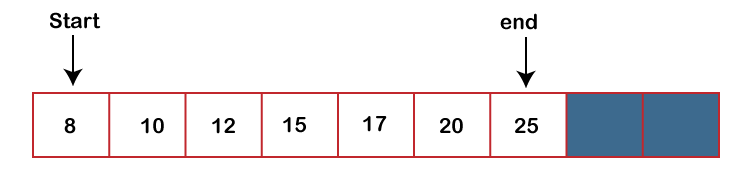
The value of the root node is 15 which is greater than 10 but lesser than 16, so it does not satisfy the property of the Binary search tree. Therefore, it is not a binary search tree.

Operations on Binary search tree

We can perform insert, delete and search operations on the binary search tree. Let's understand how a search is performed on a binary search. The binary tree is shown below on which we have to perform the search operation:



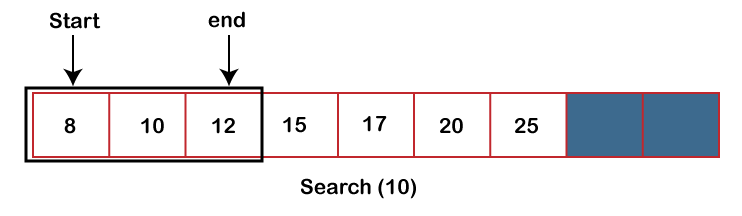
Suppose we have to search 10 in the above binary tree. To perform the binary search, we will consider all the integers in a sorted array. First, we create a complete list in a search space, and all the numbers will exist in the search space. The search space is marked by two pointers, i.e., start and end. The array of the above binary tree can be represented as



First, we will calculate the middle element and compare the middle element with the element, which is to be searched. The middle element is calculated by using n/2. The value of n is 7; therefore, the middle element is 15. The middle element is not equal to the searched element, i.e., 10.

Note: If the element is being searched is lesser than the mid element, then the searching will be performed in the left half; else, searching will be done on the right half. In the case of equality, the element is found.

As the element to be searched is lesser than the mid element, so searching will be performed on the left array. Now the search is reduced to half, as shown below:

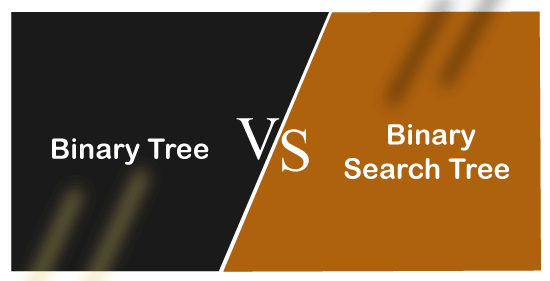


The mid element in the left array is 10, which is equal to the searched element.

Time complexity

In a binary search, there are n elements. If the middle element is not equal to the searched element, then the search space is reduced to n/2, and we will keep on reducing the search space by n/2 until we found the element. In the whole reduction, if we move from n to n/2 to n/4 and so on, then it will take log2n steps.

**Differences between Binary tree and Binary search tree**





<https://www.datatrained.com/post/difference-between-bst-and-binary-tree/>

## ****Common Terminologies of Binary Tree****

In the difference between Bst and Binary tree, there are certain terminologies in the difference between Bst and binary tree. Let’s have a glance:

* **Root:**Topmost node in the tree is referenced to be the root.
* **Parent:** Every node in a tree (excluding the root) is connected to exactly one other node via a directed edge.
* **Child:** When going away from the root, a child is a node that is directly related to another node.
* **Leaf or External Node:** Node with no children is called leaf or external node
* **Internal Node:** Node having at least one child is referred to as an internal node.
* **Depth of a Node:** The depth of a node is the no. of edges from the root to the node.
* **Height of a Node:** No. of edges from the node to the deepest leaf. This determines the height of a node. The root’s height is the tree’s height.

## Types of Binary Tree

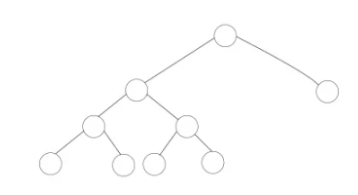
There are many different forms of and difference between Bst and binary tree, each with its own set of properties. Each of the binary tree variants is described in depth below.

Full or Proper or Strict Binary Tree

A strict binary tree is another name for a full binary tree. If each node has either 0 or 2 offspring, the tree can only be termed a complete binary tree. Except for the leaf nodes, the full binary tree may alternatively be described as a tree in which each node must have two children. In the difference between Bst and binary tree, this is a type of binary tree.

It’s a specific type of binary tree that contains either zero or two children. It indicates that every node in the binary tree should contain two child nodes, or the parent node should be the leaf node or the external node itself. In other terms, a full binary tree is a distinct binary tree in which every node has two children except the exterior node. This is an important difference between Bst and Binary Tree.

Quite a binary tree would not be a full binary tree if it just has one child. The number of leaf nodes equals the number of internal nodes plus one in this case. L=I+1, where L is the multitude of leaf nodes and I is the number of internal nodes, is the equation.



**Properties of Full Binary Tree:**

* The number of leaf nodes equals the no. of internal nodes plus one. The no. of internal nodes in the above example is 5, hence the number of leaf nodes is 6.
* The greatest number of nodes is the same as a binary tree’s number of nodes i.e., 2h+1 -1.
* The min no. of nodes in a full binary tree = 2\*h-1.
* log2(n+1) – 1 is the min-height of the full binary tree.
* The max height of the full binary tree could be evaluated as:

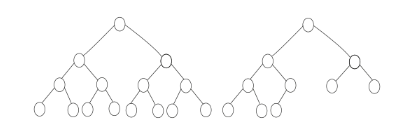
**n= 2\*h – 1**

**n+1 = 2\*h**

**h = n+1/2**

Complete Binary Tree

A complete binary tree is a sort of binary tree in which all of the tree’s levels, save the lowest, are completely filled with nodes. Except for the last level, the complete binary tree is a tree in which all of the nodes are entirely filled. All of the nodes in the final level must be as far to the left as feasible. The nodes of a complete binary tree should be inserted from the left. In addition, every node in the binary tree’s final or lowest level should be on the left side. The difference between Bst and binary tree it is a type of binary tree. Here is a structure of the complete binary tree:



**Properties of Complete Binary Tree:**

* Max No. of nodes of a complete binary tree is equal to **2h+1 – 1**.
* Min No. of nodes of a complete binary tree is equal to **2h**.
* Min height of the complete binary tree is equal to **log2(n+1) – 1**.

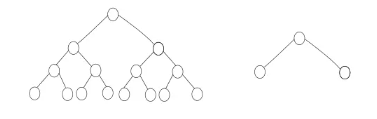
A complete binary tree is similar to a full binary tree, but there are some significant distinctions.

* Every level must be completed in its entirety.
* The leaf components must all slant to the left.
* A complete binary tree does not have to be a full binary tree because the last leaf element may not have the right sibling.

Perfect Binary Tree

If all internal nodes have exactly two children and every exterior or leaf node is at the exact level of depth inside the tree, it is said to be ‘perfect.’ **2h – 1** node is there is a perfect binary tree of height ‘**h**‘. In the difference between Bst and binary tree, this is a type of binary tree.

* In a perfect binary tree with l leaves. There are **n = 2l-1** nodes.
* **l = 2h** and **n = 2h+1 – 1** in a perfect full binary tree, where n is the number of nodes, **h** is the tree’s height, and l is the number of leaf nodes.

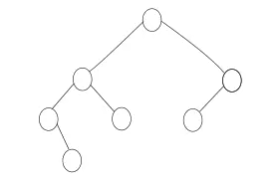


Balanced Binary Tree

When the height of a binary tree is O(logN), where ‘N’ is the number of nodes, the tree is said to be ‘balanced.’ The height of each node’s left and right subtrees should differ by no more than one in a balanced binary tree. A balanced binary search tree can be generated using a data structure like an AVL Tree or a Red-Black Tree. It’s a sort of binary tree in which each node’s height difference between the left and right subtrees is either 0 or 1. An empty tree is height-balanced. In the difference between Bst and binary tree, this is a type of binary tree.

**If a binary tree fulfills these conditions, it is height-balanced:**

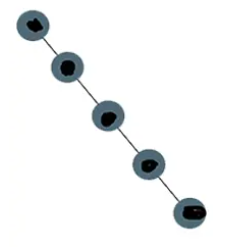
* The heights of the right and left subtrees differ by no more than 1.
* The left subtree is balanced
* The right subtree is balanced



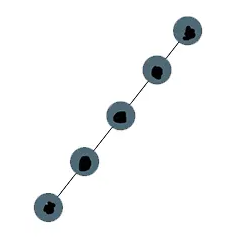
### Degenerate Binary Tree

If every internal node has just one child, the binary tree is considered to be a degenerate binary tree or pathological binary tree. In terms of performance, such trees are equivalent to linked lists. In the difference between Bst and binary tree, this is a type of binary tree.

Let’s look at a few samples of the Degenerate binary tree.



Because all of the nodes in the above tree have just one child, it is a degenerate binary tree. It’s also known as a right-skewed tree since each node only has one right child. In the difference between Bst and binary tree, this is a type of binary tree.



Because all of the nodes have only one child, the above tree is also a degenerate binary tree. It’s also known as a left-skewed tree since every node has just a left child. In the difference between Bst and binary tree, this is a type of binary tree.

## ****Benefits of a Binary Tree****

In the difference between Bst and Binary tree, trees like Bst and binary tree are very beneficial and widely used because they provide a number of significant benefits:

* The structural linkages in the data are shown in trees.
* Hierarchies are represented by trees.
* Insertion and searching are made easier using trees.
* Trees are an extremely versatile kind of data because they enable you to shift subtrees around with little effort.
* When compared to other trees, the search process in a binary tree is faster.
* To get the elements in sorted order, just two traversals are required.
* Picking up the maximum and lowest components is simple.
* Binary trees are employed in graph traversal.
* Binary trees can be used to convert various postfix and prefix expressions.

## ****What is a Binary Search Tree?****

In Bst and binary tree, a Binary Search Tree is a binary tree with a key and an optional associated value at each node. It enables extremely quick item lookup, adding, and removal. The nodes in a binary search tree are ordered as per the following properties:

* A node’s left subtree will always contain nodes with keys that are less than that node’s key.
* Nodes with keys bigger than a node’s key will always be found in the right subtree of that node.
* Binary search trees will be found in both the left and right subtrees of each node.

As a result, BST and binary tree are distinct as BST separates all of its sub-trees into two segments: the left and right subtrees, and may be characterized as:

**left\_subtree (keys) < node (key) ≤ right\_subtree (keys)**

### 1. Time Complexity

The following qualities, in most situations, allow insert, search, and delete operations to be performed in O(logn)O(log n)O(logn) time, where n is the number of nodes in the tree. In the worst scenario, when the tree becomes imbalanced, the time complexity of these operations is O(n)O(n)O(n).

|  |  |  |  |
| --- | --- | --- | --- |
| **Operations** | **Best case time complexity** | **Average case time complexity** | **Worst-case time complexity** |
| Insertion | O(log n) | O(log n) | O(n) |
| Deletion | O(log n) | O(log n) | O(n) |
| Search | O(log n) | O(log n) | O(n) |

### 2. Space Complexity

In both the average and worst instances, the space complexity of a binary search tree is O(n)O(n)O(n).

|  |  |
| --- | --- |
| **Operations** | **Space complexity** |
| Insertion | O(n) |
| Deletion | O(n) |
| Search | O(n) |

3. Types of Traversals

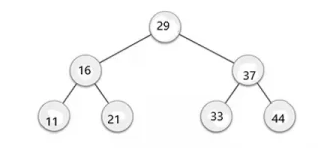
This is a common operation in Bst and Binary Tree. The Binary Search Tree can be navigated in a variety of ways:

* **Pre-order Traversal:** Pre-order traversal visits nodes in the order Parent-LeftChild-RightChild.
* **In-order Traversal:** In-order traversal visits nodes in the order LeftChild-Parent-RightChild. The tree is visited in ascending order of keys in this manner.
* **Post-order Traversal:** The traversal will traverse nodes in the order LeftChild-RightChild-Parent.

## ****Representation of the difference between Bst and Binary Tree****

In the difference between Bst and Binary tree, as BST is a set of nodes structured in such a manner that they all have the same BST attributes. A key & a value are allocated to each node. The required key is compared to the keys in BST during the search, and if found, the related value is obtained.

A visual illustration of BST is shown below.



The root node key (29) has all lower-valued keys on the left sub-tree and higher-valued keys on the right subtree, as can be seen.

## ****Basic Operations****

The basic operations of a tree-like Bst and binary tree are as follows:

* **Search:** Searches for a certain element in a tree.
* **Insert:** Inserts a new element into a tree.
* **Pre-order Traversal:** Pre-order Traversal traverses a tree in the order in which it was created.
* **In-order Traversal:** In-order Traversal traverses a tree in a sequential fashion.
* **Post-order traversal:** Traversing a tree in a post-order way is known as post-order traversal.

## ****Searching in Binary Search Tree****

In a data structure, searching implies finding or locating a certain element or node. In BST and binary tree, the binary search tree components are kept in a specified sequence, finding a node in a Binary search tree is simple. This is a common operation in Bst and binary tree and is not a difference between Bst and Binary Tree. The stages involved in finding a node in a Binary Search tree are as follows:

* To begin, compare the element to be searched with the tree’s root element.
* Return the position of the node if the root matches the target element.
* If it isn’t matched, check whether the item is smaller than the root element; if it is, go to the left subtree.
* Move to the right subtree if it is greater than the root element.
* Recursively repeat the technique above until the match is discovered.
* Return NULL if the element isn’t discovered or doesn’t exist in the tree.

### Algorithm to search in Bst and binary tree comparison

Search (root, item)

Step 1 – if (item = root → data) or (root = NULL)

return root

else if (item < root → data)

return Search(root → left, item)

else

return Search(root → right, item)

END if

Step 2 – END

## ****Insertion in Binary Search Tree****

The difference between BST and binary tree, in Bst a new key is always introduced at the leaf. To insert an element in BST, start at the root node and search for an empty spot in the left subtree of the node to be added is smaller than the root node. Otherwise, look for an empty spot in the correct subtree and fill it in. In BST, inserting is similar to searching in that we must constantly remember that the left subtree is lesser than the root and the right subtree is larger. Insertion operation is present in both Bst and binary tree and is not a difference between Bst and Binary Tree.

### Algorithm for Insertion in Binary Search Tree: BST and Binary Tree distinction (Iterative solution)

void insert(int data) {

struct node \*tempNode = (struct node \*)malloc(sizeof(struct node));

struct node \*current;

struct node \*parent;

tempNode->data = data;

tempNode->leftChild = NULL;

tempNode->rightChild = NULL;

// if tree is empty

if (root == NULL) {

root = tempNode;

} else {

current = root;

parent = NULL;

while (1) {

parent = current;

// go to left of the tree

if (data < parent->data) {

current = current->leftChild;

// insert to the left

if (current == NULL) {

parent->leftChild = tempNode;

return;

}

} // go to right of the tree

else {

current = current->rightChild;

// insert to the right

if (current == NULL) {

parent->rightChild = tempNode;

return;

}

}

}

}

}

## ****Deletion in Binary Search Tree: BST and Binary Tree distinction****

In the difference between Bst and Binary Tree, there are several scenarios for removing a node from a BST and binary tree, such as deleting a root or deleting a leaf node. We must also consider the root node after removing a root. If we wish to delete a leaf node, we may simply delete it; however, we must replace the root’s value with another node if we want to delete a root. Deletion operation is present in both Bst and binary tree and it is not a difference between Bst and Binary Tree. Consider the following scenario:

* **Case A = Node with zero children:** this is the simplest circumstance; all you have to do is remove the node on the right or left that has no more children. This was Case A difference between Bst and binary tree deletion operation.
* **Case B = Single-child node:** after deleting the node, just connect its child node to the removed value’s parent node. This was the Case B difference between Bst and binary tree deletion operation.
* The most challenging circumstance is **Case C Node with two children**, which is based on the following two principles.
  + **C1 – In Order Predecessor:** Delete the node with two children and replace it with the highest value on the destroyed node’s left-subtree. This was the Case C1 difference between Bst and binary tree deletion operation.
  + **C2 – In Order Successor:** Delete the node with two children and replace it with the highest value on the destroyed node’s right-subtree. This was the Case C2 difference between Bst and binary tree deletion operation.

## What is the difference between BST and Binary Tree?

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Binary Tree** | **Binary Search Tree** |
| **Definition** | A Binary Tree is a non-linear data structure with nodes that can have two, one, or zero nodes. Each node is made up of a right pointer, a left pointer, and a data element. | The BST, or Binary Search Tree, is an ordered Binary Tree with structurally arranged nodes. Each subtree must be a component of the structure in question. |
| **Operations Performance** | The Binary Tree may be used to accomplish operations such as insertion, deletion, and traversal. | As the Binary Trees are sorted, they may be used for efficient and quick binary search, deletion, and insertion. |
| **Structure** | It does not need a well-organized structure of the nodes in a tree. | It has a well-defined structure.  The right subtree’s value should be higher than the node’s, while the left subtree should be lower. |
| **Types** | There are various forms of binary trees.  The Full Binary Tree, Complete Binary Tree, and Extended Binary Tree are the most prevalent. | There are several forms of Binary Search Trees. Splay Trees, AVL Trees, T-Trees, Tango Trees, and other varieties are among the most popular. |
| **Speed** | The deletion, insertion, and searching procedures are slower in the Binary tree than in the Binary Search Tree because it is unordered. | The Binary Search Tree performs quicker deletion, insertion, and searching of items since it has ordered properties. |
| **Hierarchy** | Binary Trees are a type of hierarchical data structure. It is made up of nodes, which are a collection of elements. | The nodes are placed in relative order in this Binary Tree version. |
| **Time Taken** | When compared to a Binary Search Tree, any operation on a Binary Tree takes longer. As a result, inserting, searching, and deleting data takes O(n) time. | A Binary Search Tree maintains its sorting. As a result, insert, search, and delete operations are quicker. Thus, it takes about O(log n) time to complete.  In the event of lookups, a user can primarily use the BST (because all the keys stay arranged in sorted order). |
| **Duplicate Values** | Duplicate node values are allowed in the Binary Tree. | Duplicate node values are not allowed in the Binary Search Tree. |
| **Usage** | In any tree structure, the Binary Tree works well for efficient and rapid lookup of information and data. | When it comes to element deletion, insertion, and searching, the Binary Search Tree excels. |

So now you must be well familiar with the difference between BST and Binary Tree.