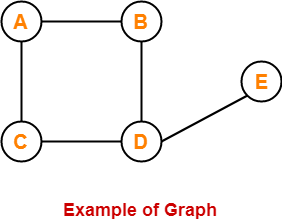
**Graphs-**

* A graph is a collection of vertices connected through a set of edges.
* The study of graphs is known as **Graph Theory**.

|  |
| --- |
| **Formal Definition**  Formally,  A graph is defined as an ordered pair of a set of vertices and a set of edges.  **G = (V, E)**  Here, V is the set of vertices and E is the set of edges connecting the vertices. |

**Example-**



1. AB, AC, BD, CD, DE

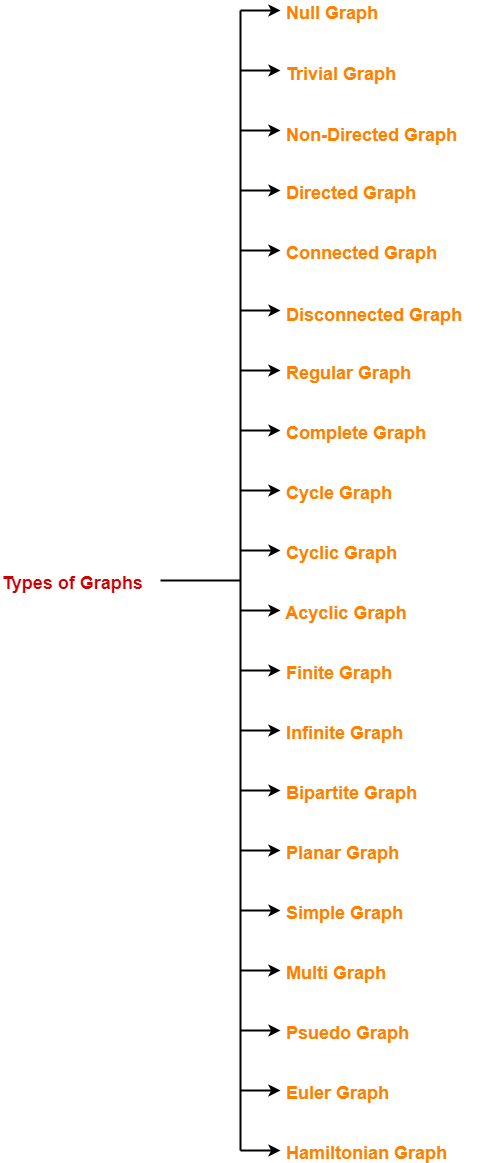
In this graph,

V = { A, B, C, D, E }

E = { AB, AC, BD, CD, DE }

**Types of Graphs-**

Various important types of graphs in graph theory are-

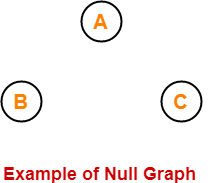


1. Null Graph
2. Trivial Graph
3. Non-directed Graph
4. Directed Graph
5. Connected Graph
6. Disconnected Graph
7. Regular Graph
8. Complete Graph
9. Cycle Graph
10. Cyclic Graph
11. Acyclic Graph
12. Finite Graph
13. Infinite Graph
14. Bipartite Graph
15. Planar Graph
16. Simple Graph
17. Multi Graph
18. Pseudo Graph
19. Euler Graph
20. Hamiltonian Graph

**1. Null Graph-**

* A graph whose edge set is empty is called a null graph.
* In other words, a null graph does not contain any edges in it.

**Example-**



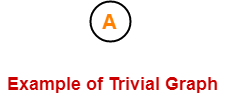
Here,

* This graph consists only of the vertices and there are no edges in it.
* Since the edge set is empty, therefore it is a null graph.

**2. Trivial Graph-**

* A graph having only one vertex in it is called a trivial graph.
* It is the smallest possible graph.

**Example-**



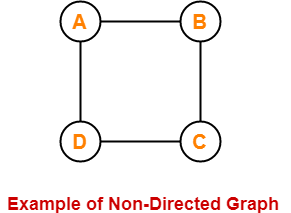
Here,

* This graph consists of only one vertex and there are no edges in it.
* Since only one vertex is present, therefore it is a trivial graph.

**3. Non-Directed Graph-**

* A graph in which all the edges are undirected is called a non-directed graph.
* In other words, the edges of an undirected graph do not contain any direction.

**Example-**



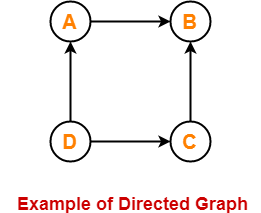
Here,

* This graph consists of four vertices and four undirected edges.
* Since all the edges are undirected, therefore it is a non-directed graph.

**4. Directed Graph-**

* A graph in which all the edges are directed is called a directed graph.
* In other words, all the edges of a directed graph contain some direction.
* Directed graphs are also called **digraphs**.

**Example-**



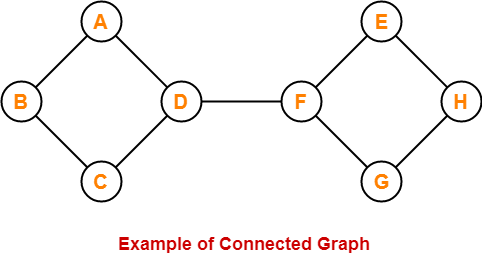
Here,

* This graph consists of four vertices and four directed edges.
* Since all the edges are directed, therefore it is a directed graph.

**5. Connected Graph-**

* A graph in which we can visit from any one vertex to any other vertex is called a connected graph.
* In a connected graph, at least one path exists between every pair of vertices.

**Example-**



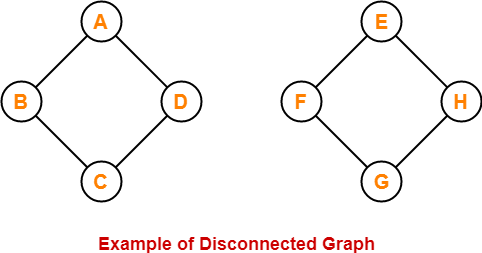
Here,

* In this graph, we can visit from any one vertex to any other vertex.
* There exists at least one path between every pair of vertices.
* Therefore, it is a connected graph.

**6. Disconnected Graph-**

* A graph in which there does not exist any path between at least one pair of vertices is called a disconnected graph.

**Example-**



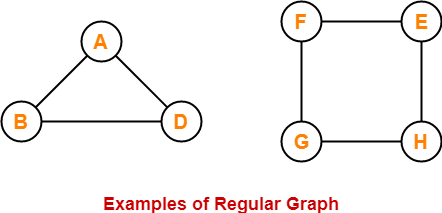
Here,

* This graph consists of two independent components which are disconnected.
* It is not possible to visit from the vertices of one component to the vertices of another component.
* Therefore, it is a disconnected graph.

**7. Regular Graph-**

* A graph in which the degree of all the vertices is the same is called a regular graph.
* If all the vertices in a graph are of degree ‘k’, then it is called a “**k-regular graph**“.

**Examples-**



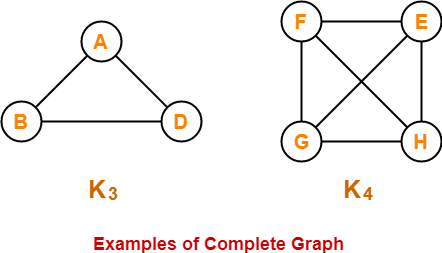
In these graphs,

* All the vertices have degree 2.
* Therefore, they are 2-Regular graphs. NC2=FAC N/FAC C \* FAC N-C

**8. Complete Graph-**

* A graph in which exactly one edge is present between every pair of vertices is called a complete graph.
* A complete graph of ‘n’ vertices contains exactly nC2 edges.
* A complete graph of ‘n’ vertices is represented as **Kn**.

**Examples-**

3C2=FAC 3/(FAC 2 \* FAC(1)

 = 3 FAC2/(FAC 2\* FAC 1)

=3

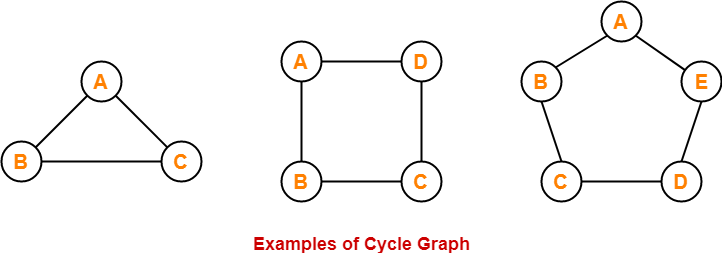
In these graphs,

* Each vertex is connected with all the remaining vertices through exactly one edge.
* Therefore, they are complete graphs.

**9. Cycle Graph-**

* A simple graph of ‘n’ vertices (n>=3) and n edges forming a cycle of length ‘n’ is called a cycle graph.
* In a cycle graph, all the vertices are of degree 2.

**Examples-**



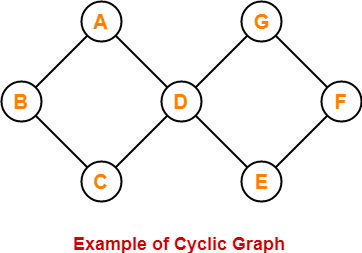
In these graphs,

* Each vertex is having degree 2.
* Therefore, they are cycle graphs.

**10. Cyclic Graph-**

* A graph containing at least one cycle is called a cyclic graph.

**Example-**



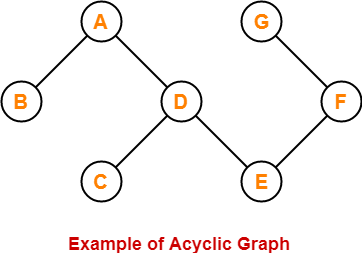
Here,

* This graph contains two cycles in it.
* Therefore, it is a cyclic graph.

**11. Acyclic Graph-**

* A graph not containing any cycle is called an acyclic graph.

**Example-**



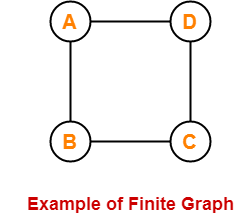
Here,

* This graph does not contain any cycle in it.
* Therefore, it is an acyclic graph.

**12. Finite Graph-**

* A graph consisting of a finite number of vertices and edges is called a finite graph.

**Example-**



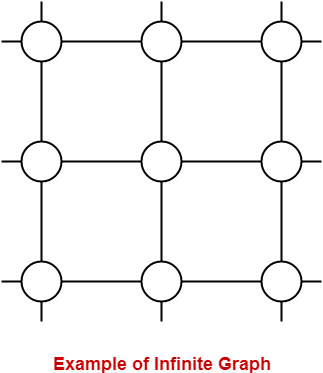
Here,

* This graph consists of a finite number of vertices and edges.
* Therefore, it is a finite graph.

**13. Infinite Graph-**

* A graph consisting of an infinite number of vertices and edges is called an infinite graph.

**Example-**



Here,

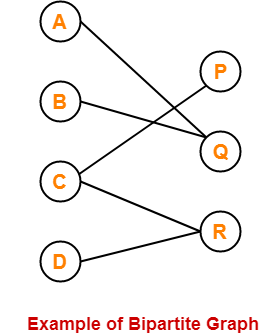
* This graph consists of an infinite number of vertices and edges.
* Therefore, it is an infinite graph.

**14. Bipartite Graph-**

A bipartite graph is a graph where-

* Vertices can be divided into two sets X and Y.
* The vertices of set X only join with the vertices of set Y.
* None of the vertices belonging to the same set join each other.

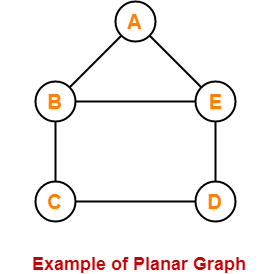
**Example-**



**15. Planar Graph-**

* A planar graph is a graph that we can draw in a plane such that no two edges of it cross each other.

**Example-**



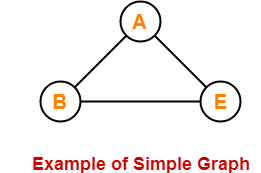
Here,

* This graph can be drawn in a plane without crossing any edges.
* Therefore, it is a planar graph.

**16. Simple Graph-**

* A graph having no self-loops and no parallel edges in it is called a simple graph.

**Example-**



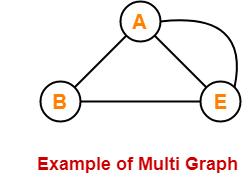
Here,

* This graph consists of three vertices and three edges.
* There are neither self-loops nor parallel edges.
* Therefore, it is a simple graph.

**17. Multi Graph-**

* A graph with no self loops but parallel edge(s) is called a multi graph.

**Example-**



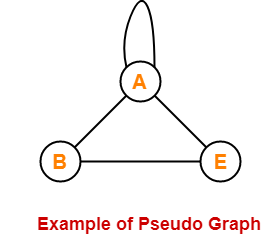
Here,

* This graph consists of three vertices and four edges out of which one edge is a parallel edge.
* There are no self loops but a parallel edge is present.
* Therefore, it is a multi-graph.

**18. Pseudo Graph-**

* A graph having no parallel edges but having self-loop (s) in it is called a pseudo graph.

**Example-**



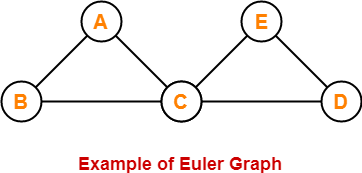
Here,

* This graph consists of three vertices and four edges out of which one edge is a self-loop.
* There are no parallel edges but a self-loop is present.
* Therefore, it is a pseudo graph.

**19. Euler Graph-**

* Euler Graph is a connected graph in which all the vertices are even degrees.

**Example-**



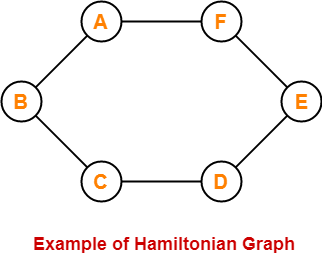
Here,

* This graph is connected.
* The degree of all the vertices is even.
* Therefore, it is an Euler graph.

**20. Hamiltonian Graph-**

* If there exists a closed walk in the connected graph that visits every vertex of the graph exactly once (except the starting vertex) without repeating the edges, then such a graph is called a Hamiltonian graph.

**Example-**



Here,

* This graph contains a closed walk ABCDEFG that visits all the vertices (except the starting vertex) exactly once.
* All the vertices are visited without repeating the edges.
* Therefore, it is a Hamiltonian Graph.

**Important Points-**

* The edge set of a graph can be empty but the vertex set of a graph can not be empty.
* Every polygon is a 2-regular Graph.
* Every complete graph of ‘n’ vertices is a (n-1)-regular graph.
* Every regular graph need not be a complete graph.

**Remember-**

The following table is useful to remember different types of graphs-

|  |  |  |
| --- | --- | --- |
|  | **Self-Loop(s)** | **Parallel Edge(s)** |
| **Graph** | Yes | Yes |
| **Simple Graph** | No | No |
| **Multi Graph** | No | Yes |
| **Pseudo Graph** | Yes | No |

LAST TOPIC

**Applications of Graph Theory-**

Graph theory has applications in diverse fields of engineering-

**1. Electrical Engineering-**

* The concepts of graph theory are used extensively in designing circuit connections.
* The types or organization of connections are named topologies.
* Some examples of topologies are star, bridge, series, and parallel topologies.

**2. Computer Science-**

Graph theory is used for the study of algorithms such as

* [**Kruskal’s Algorithm**](https://www.gatevidyalay.com/kruskals-algorithm-kruskals-algorithm-example/)
* [**Prim’s Algorithm**](https://www.gatevidyalay.com/prims-algorithm-prim-algorithm-example/)
* [**Dijkstra’s Algorithm**](https://www.gatevidyalay.com/dijkstras-algorithm-shortest-path-algorithm/)

**3. Computer Network-**

The relationships among interconnected computers in the network follow the principles of graph theory.

**4. Science-**

The following structures are represented by graphs-

* Molecular structure of a substance
* Chemical structure of a substance
* DNA structure of an organism etc

**5. Linguistics-**

The parsing tree of a language and the grammar of a language use graphs.

**6. Other Applications-**

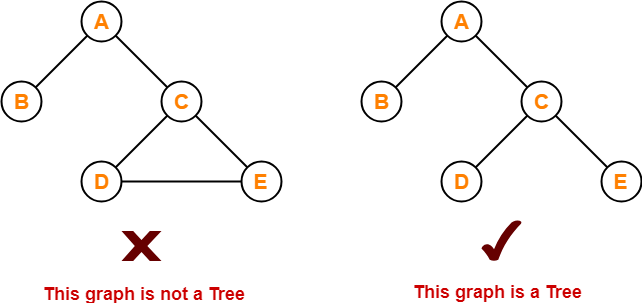
* Routes between the cities are represented using graphs.
* Hierarchical ordered information such as family trees are represented using special types of graphs called trees.

**Tree Data Structure-**

Tree data structure may be defined as

|  |
| --- |
| Tree is a non-linear data structure which organizes data in a hierarchical structure and this is a recursive definition.  **OR**  A tree is a connected graph without any circuits.  **OR**  If in a graph, there is one and only one path between every pair of vertices, then graph is called as a tree. |

**Example-**



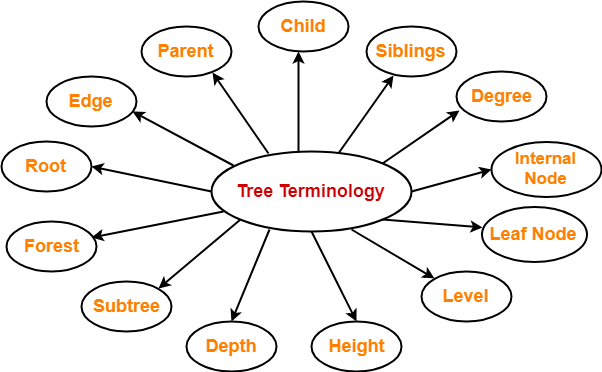
**Properties-**

The important properties of tree data structure are-

* There is one and only one path between every pair of vertices in a tree.
* A tree with n vertices has exactly (n-1) edges.
* A graph is a tree if and only if it is minimally connected.
* Any connected graph with n vertices and (n-1) edges is a tree.

**Tree Terminology-**

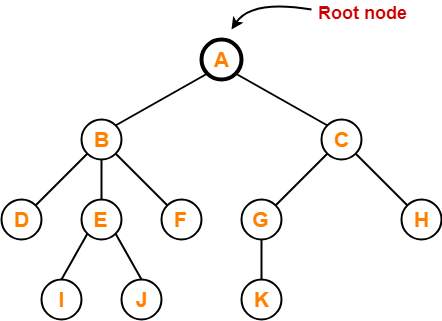
The important terms related to tree data structure are-



**1. Root-**

* The first node from where the tree originates is called as a **root node**.
* In any tree, there must be only one root node.
* We can never have multiple root nodes in a tree data structure.

**Example-**

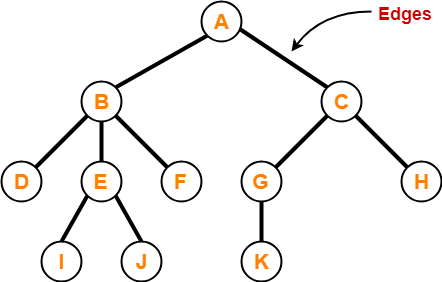


Here, node A is the only root node.

**2. Edge-**

* The connecting link between any two nodes is called as an **edge**.
* In a tree with n number of nodes, there are exactly (n-1) number of edges.

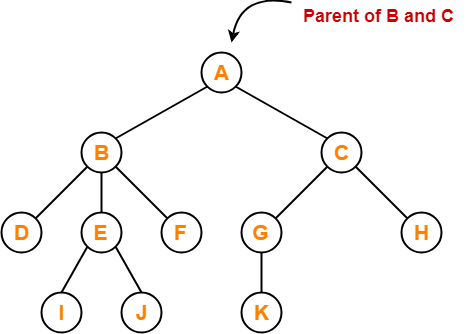
**Example-**



**3. Parent-**

* The node which has a branch from it to any other node is called as a **parent node**.
* In other words, the node which has one or more children is called as a parent node.
* In a tree, a parent node can have any number of child nodes.

**Example-**



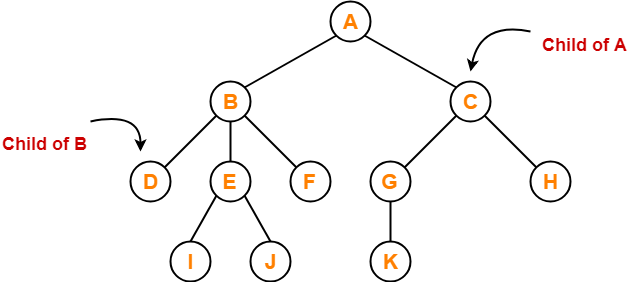
Here,

* Node A is the parent of nodes B and C
* Node B is the parent of nodes D, E and F
* Node C is the parent of nodes G and H
* Node E is the parent of nodes I and J
* Node G is the parent of node K

**4. Child-**

* The node which is a descendant of some node is called as a **child node**.
* All the nodes except root node are child nodes.

**Example-**



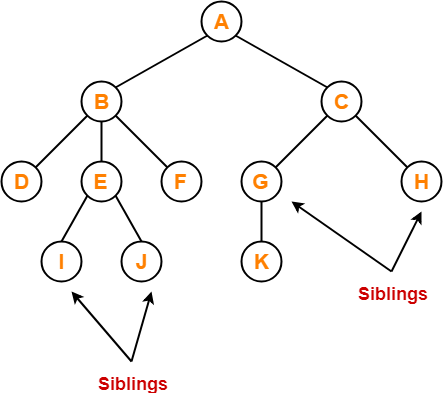
Here,

* Nodes B and C are the children of node A
* Nodes D, E and F are the children of node B
* Nodes G and H are the children of node C
* Nodes I and J are the children of node E
* Node K is the child of node G

**5. Siblings-**

* Nodes which belong to the same parent are called as **siblings**.
* In other words, nodes with the same parent are sibling nodes.

**Example-**



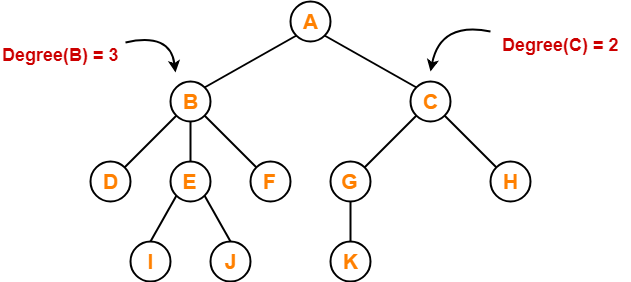
Here,

* Nodes B and C are siblings
* Nodes D, E and F are siblings
* Nodes G and H are siblings
* Nodes I and J are siblings

**6. Degree-**

* **Degree of a node** is the total number of children of that node.
* **Degree of a tree** is the highest degree of a node among all the nodes in the tree.

**Example-**



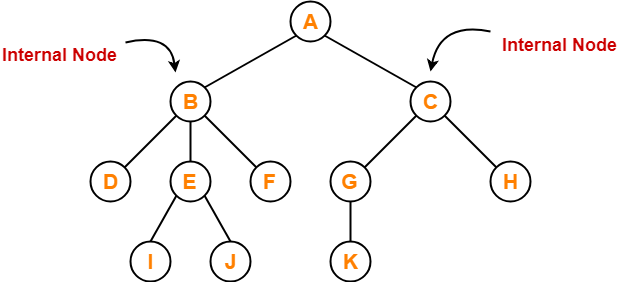
Here,

* Degree of node A = 2
* Degree of node B = 3
* Degree of node C = 2
* Degree of node D = 0
* Degree of node E = 2
* Degree of node F = 0
* Degree of node G = 1
* Degree of node H = 0
* Degree of node I = 0
* Degree of node J = 0
* Degree of node K = 0

**7. Internal Node-**

* The node which has at least one child is called as an **internal node**.
* Internal nodes are also called as **non-terminal nodes**.
* Every non-leaf node is an internal node.

**Example-**

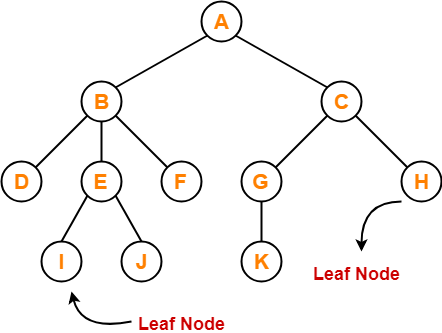


Here, nodes A, B, C, E and G are internal nodes.

**8. Leaf Node-**

* The node which does not have any child is called as a **leaf node**.
* Leaf nodes are also called as **external nodes** or **terminal nodes**.

**Example-**

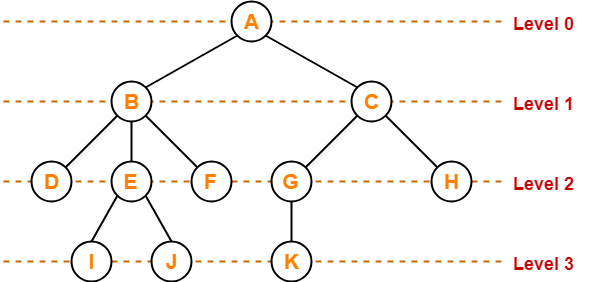


Here, nodes D, I, J, F, K and H are leaf nodes.

**9. Level-**

* In a tree, each step from top to bottom is called as **level of a tree**.
* The level count starts with 0 and increments by 1 at each level or step.

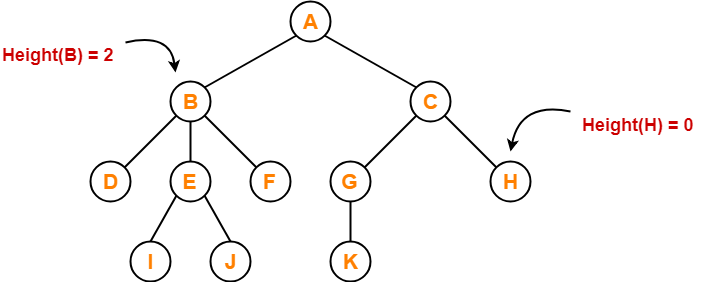
**Example-**



**10. Height-**

* Total number of edges that lies on the longest path from any leaf node to a particular node is called as **height of that node**.
* **Height of a tree** is the height of root node.
* Height of all leaf nodes = 0

**Example-**



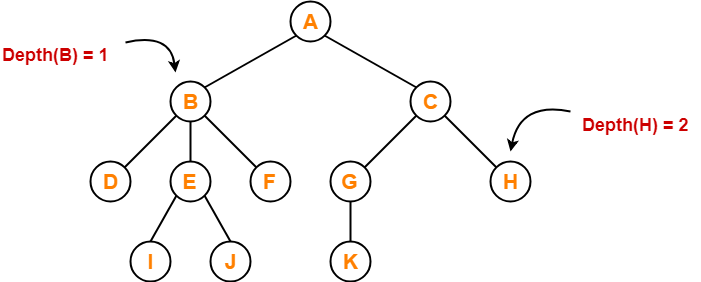
Here,

* Height of node A = 3
* Height of node B = 2
* Height of node C = 2
* Height of node D = 0
* Height of node E = 1
* Height of node F = 0
* Height of node G = 1
* Height of node H = 0
* Height of node I = 0
* Height of node J = 0
* Height of node K = 0

**11. Depth-**

* Total number of edges from root node to a particular node is called as **depth of that node**.
* **Depth of a tree** is the total number of edges from root node to a leaf node in the longest path.
* Depth of the root node = 0
* The terms “level” and “depth” are used interchangeably.

**Example-**



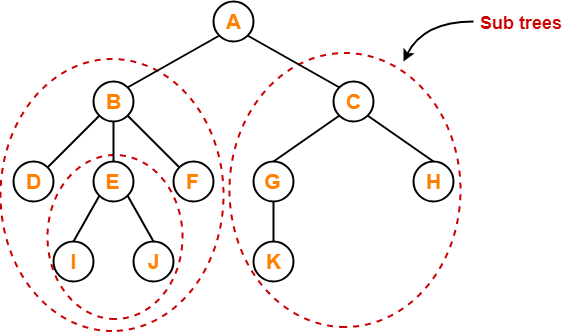
Here,

* Depth of node A = 0 Depth of a root is always ZERO
* Depth of node B = 1
* Depth of node C = 1
* Depth of node D = 2
* Depth of node E = 2
* Depth of node F = 2
* Depth of node G = 2
* Depth of node H = 2
* Depth of node I = 3
* Depth of node J = 3
* Depth of node K = 3

**12. Subtree-**

* In a tree, each child from a node forms a **subtree** recursively.
* Every child node forms a subtree on its parent node.

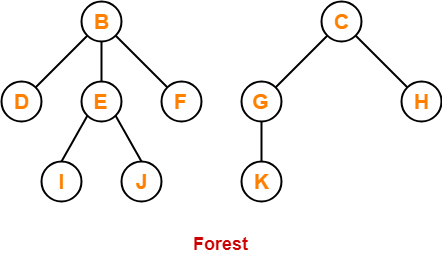
**Example-**



**13. Forest-**

A forest is a set of disjoint trees.

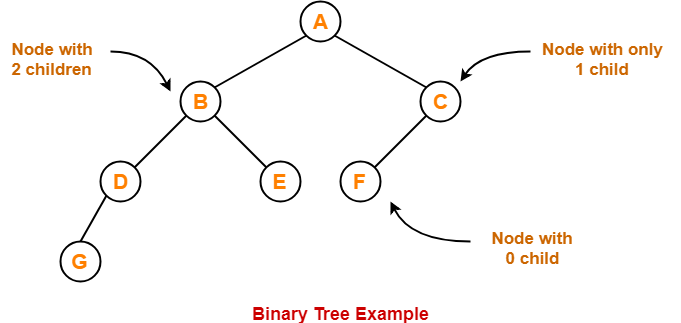
**Example-**



**Binary Tree-**

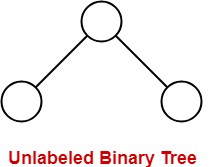
|  |
| --- |
| Binary tree is a special tree data structure in which each node can have at most 2 children.  Thus, in a binary tree,  Each node has either 0 child or 1 child or 2 children. |

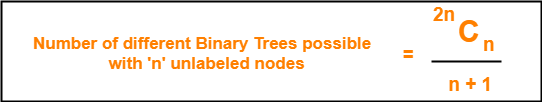
**Example-**



**Unlabeled Binary Tree-**

|  |
| --- |
| A binary tree is unlabeled if its nodes are not assigned any label. |





= 12 C 6/7= fac 12 /(fac 6 \* fac 12-6)= 12 x 11 x 10 x 9 x 8 x7 fac 6/fac 6\* fac6

=12 x 11 x 9

 = 132 x 9= 1188

**Example-**

Consider we want to draw all the binary trees possible with 3 unlabeled nodes.

Using the above formula, we have-

Number of binary trees possible with 3 unlabeled nodes

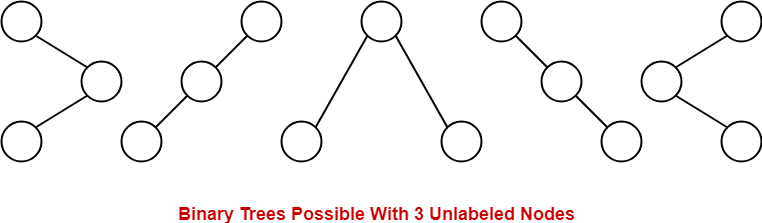
= 2 x 3C3 / (3 + 1)

= 6C3 / 4 = fac 6 /fac 3 \* fac 3 x 4= 6x5x4 fac 3/fac 3\* fac 3 \* 4 = 5

= 5

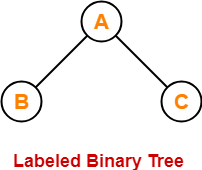
Thus,

* With 3 unlabeled nodes, 5 unlabeled binary trees are possible.
* These unlabeled binary trees are as follows-



**Labeled Binary Tree-**

|  |
| --- |
| A binary tree is labeled if all its nodes are assigned a label. |





**Example-**

Consider we want to draw all the binary trees possible with 3 labeled nodes.

Using the above formula, we have-

Number of binary trees possible with 3 labeled nodes

= { 2 x 3C3 / (3 + 1) } x 3!

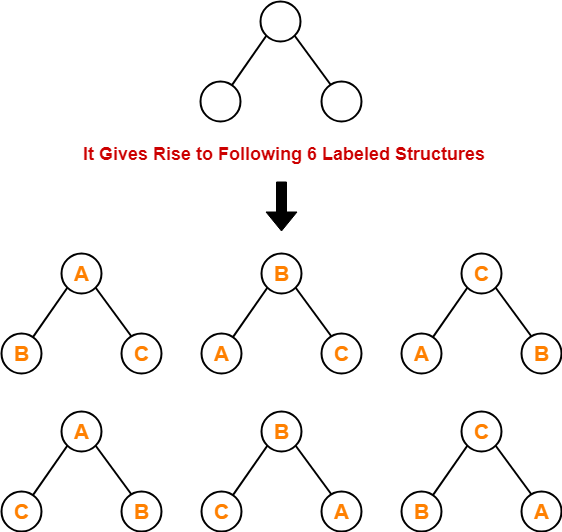
= { 6C3 / 4 } x 6

= 5 x 6

= 30

Thus,

* With 3 labeled nodes, 30 labeled binary trees are possible.
* Each unlabeled structure gives rise to 3! = 6 different labeled structures.

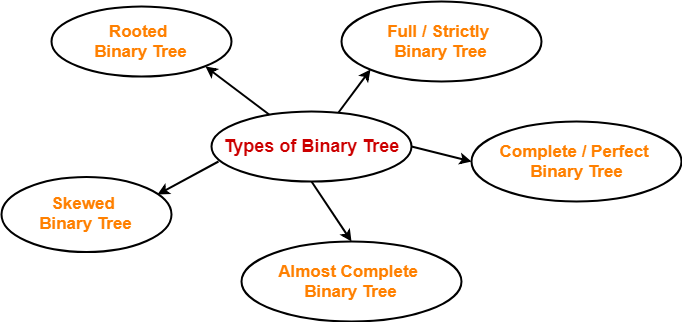


Similarly,

* Every other unlabeled structure gives rise to 6 different labeled structures.
* Thus, in total 30 different labeled binary trees are possible.

**Types of Binary Trees-**

Binary trees can be of the following types-



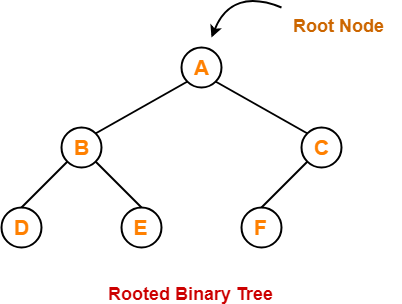
1. Rooted Binary Tree
2. Full / Strictly Binary Tree
3. Complete / Perfect Binary Tree
4. Almost Complete Binary Tree
5. Skewed Binary Tree

**1. Rooted Binary Tree-**

A **rooted binary tree** is a binary tree that satisfies the following 2 properties-

* It has a root node.
* Each node has at most 2 children.

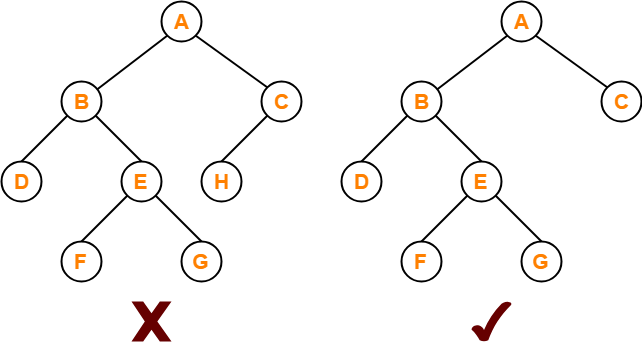
**Example-**



**2. Full / Strictly Binary Tree-**

* A binary tree in which every node has either 0 or 2 children is called as a **Full binary tree**.
* Full binary tree is also called as **Strictly binary tree**.

**Example-**



Here,

* First binary tree is not a full binary tree.
* This is because node C has only 1 child.

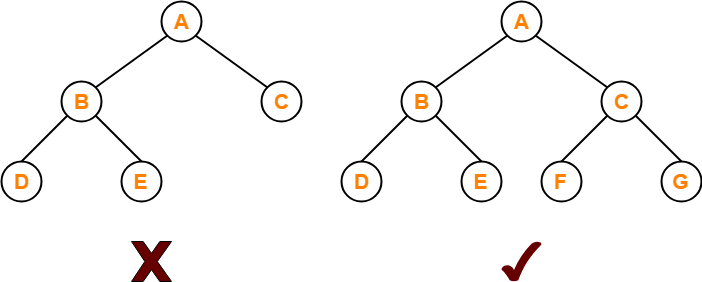
**3. Complete / Perfect Binary Tree-**

A **complete binary tree** is a binary tree that satisfies the following 2 properties-

* Every internal node has exactly 2 children.
* All the leaf nodes are at the same level.

Complete binary tree is also called as **Perfect binary tree**.

**Example-**



Here,

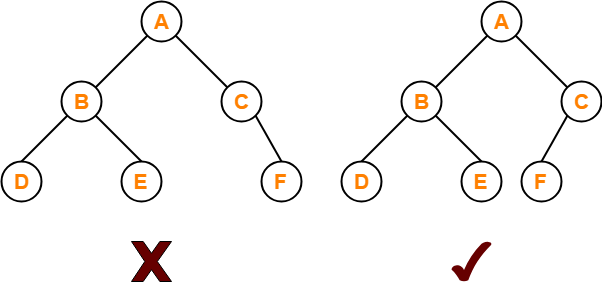
* First binary tree is not a complete binary tree.
* This is because all the leaf nodes are not at the same level.

**4. Almost Complete Binary Tree-**

An **almost complete binary tree** is a binary tree that satisfies the following 2 properties-

* All the levels are completely filled except possibly the last level.
* The last level must be strictly filled from left to right.

**Example-**



Here,

* First binary tree is not an almost complete binary tree.
* This is because the last level is not filled from left to right.

**5. Skewed Binary Tree-**

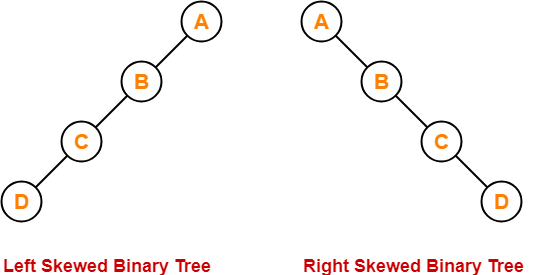
A **skewed binary tree** is a binary tree that satisfies the following 2 properties-

* All the nodes except one node has one and only one child.
* The remaining node has no child.

**OR**

A **skewed binary tree** is a binary tree of n nodes such that its depth is (n-1).

**Example-**



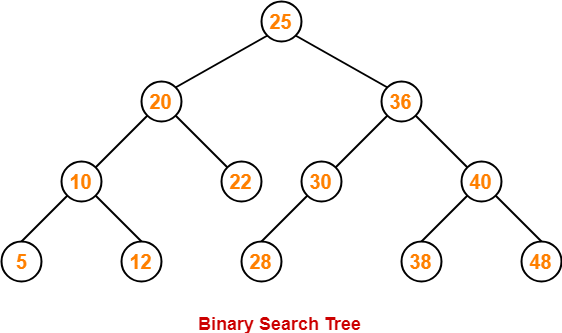
**Binary Search Tree-**

|  |
| --- |
| Binary Search Tree is a special kind of binary tree in which nodes are arranged in a specific order. |

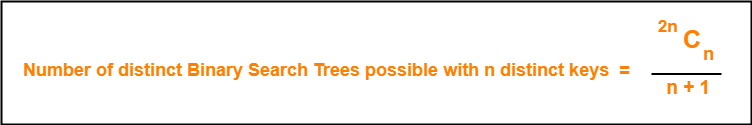
In a binary search tree (BST), each node contains-

* Only smaller values in its left sub tree
* Only larger values in its right sub tree

**Example-**



**Number of Binary Search Trees-**



**Example-**

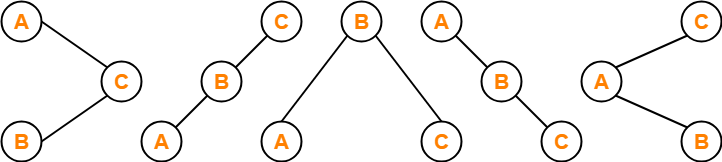
Number of distinct binary search trees possible with 3 distinct keys

= 2×3C3 / 3+1

= 6C3 / 4

= 5

If three distinct keys are A, B and C, then 5 distinct binary search trees are-



**Binary Search Tree Construction-**

Let us understand the construction of a binary search tree using the following example-

**Example-**

Construct a Binary Search Tree (BST) for the following sequence of numbers-

50, 70, 60, 20, 90, 10, 40, 100

When elements are given in a sequence,

* Always consider the first element as the root node.
* Consider the given elements and insert them in the BST one by one.

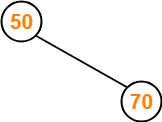
The binary search tree will be constructed as explained below-

**Insert 50-**

https://www.gatevidyalay.com/wp-content/uploads/2018/07/Binary-Search-Tree-Construction-Step-01.png

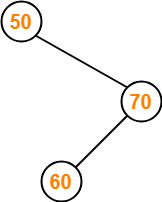
**Insert 70-**

* As 70 > 50, so insert 70 to the right of 50.



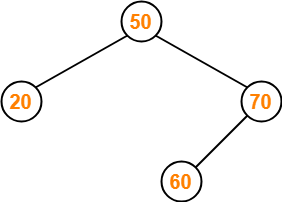
**Insert 60-**

* As 60 > 50, so insert 60 to the right of 50.
* As 60 < 70, so insert 60 to the left of 70.



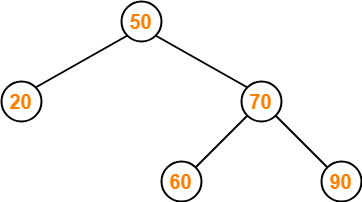
**Insert 20-**

* As 20 < 50, so insert 20 to the left of 50.



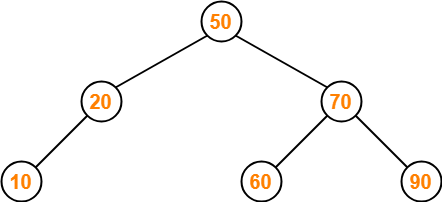
**Insert 90-**

* As 90 > 50, so insert 90 to the right of 50.
* As 90 > 70, so insert 90 to the right of 70.



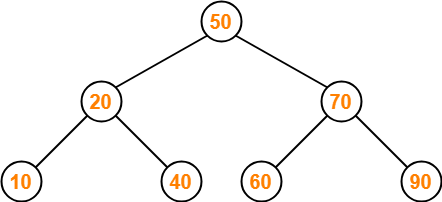
**Insert 10-**

* As 10 < 50, so insert 10 to the left of 50.
* As 10 < 20, so insert 10 to the left of 20.



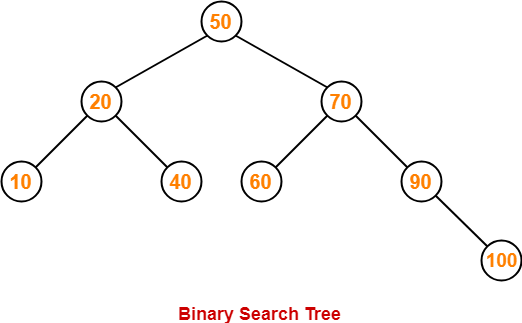
**Insert 40-**

* As 40 < 50, so insert 40 to the left of 50.
* As 40 > 20, so insert 40 to the right of 20.



**Insert 100-**

* As 100 > 50, so insert 100 to the right of 50.
* As 100 > 70, so insert 100 to the right of 70.
* As 100 > 90, so insert 100 to the right of 90.



This is the required Binary Search Tree.

**PRACTICE PROBLEMS BASED ON BINARY SEARCH TREES-**

**Problem-01:**

A binary search tree is generated by inserting in order of the following integers-

50, 15, 62, 5, 20, 58, 91, 3, 8, 37, 60, 24

50

15 62

5 20 58 91

3 8 24 37 60

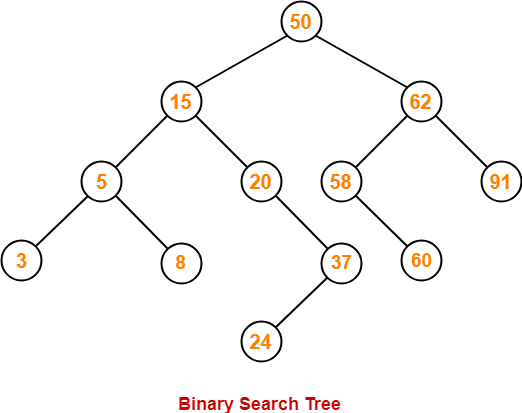
The number of nodes in the left subtree and right subtree of the root respectively is \_\_\_\_\_.

1. (4, 7)
2. (7, 4)
3. (8, 3)
4. (3, 8)

**Solution-**

Using the above discussed steps, we will construct the binary search tree.

The resultant binary search tree will be-



Clearly,

* Number of nodes in the left subtree of the root = 7
* Number of nodes in the right subtree of the root = 4

Thus, Option (B) is correct.

**Problem-02:**

How many distinct binary search trees can be constructed out of 4 distinct keys?

1. 5
2. 14
3. 24
4. 35

 2n C n /n+1

8 c 4 /5

= fac 8/ fac 4 x fac 4 x 5

= 8 x 7 x 6 x 5 fac 4/ fac 4 x fac 4 x 5

= 8 x 7 x 6 / fac 4

= 7 x 2

=14

**Solution-**

Number of distinct binary search trees possible with 4 distinct keys

= 2nCn / n+1

= 2×4C4 / 4+1

= 8C4 / 5

= 14

Thus, Option (B) is correct.

**Problem-03:**

The numbers 1, 2, …, n are inserted in a binary search tree in some order. In the resulting tree, the right subtree of the root contains p nodes. The first number to be inserted in the tree must be-

1. p
2. p+1
3. n-p
4. n-p+1

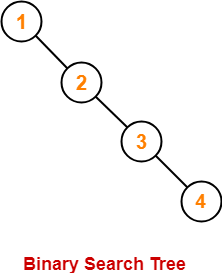
**Solution-**

Let n = 4 and p = 3.

Then, given options reduce to-

1. 3
2. 4
3. 1
4. 2

Our binary search tree will be as shown-



Clearly, first inserted number = 1.

Thus, Option (C) is correct.

**Problem-04:**

We are given a set of n distinct elements and an unlabeled binary tree with n nodes. In how many ways can we populate the tree with given set so that it becomes a binary search tree?

1. 0
2. 1
3. n!
4. C(2n, n) / n+1

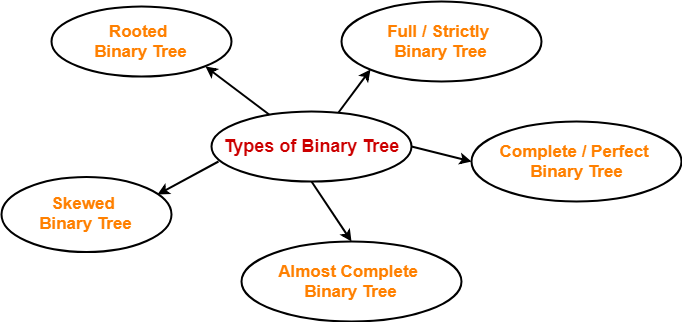
**Solution-**

Option (B) is correct.

**Binary Tree-**

We have discussed-

* Binary tree is a special tree data structure.
* In a binary tree, each node can have at most 2 children.
* There are following types of binary trees-



**Binary Tree Properties-**

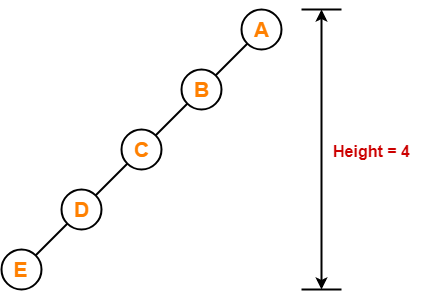
Important properties of binary trees are-

**Property-01:**

|  |
| --- |
| **Minimum number of nodes in a binary tree of height H**  **= H + 1** |

**Example-**

To construct a binary tree of height = 4, we need at least 4 + 1 = 5 nodes.



**Property-02:**

|  |
| --- |
| **Maximum number of nodes in a binary tree of height H**  **= 2H+1 – 1** |

**Example-**

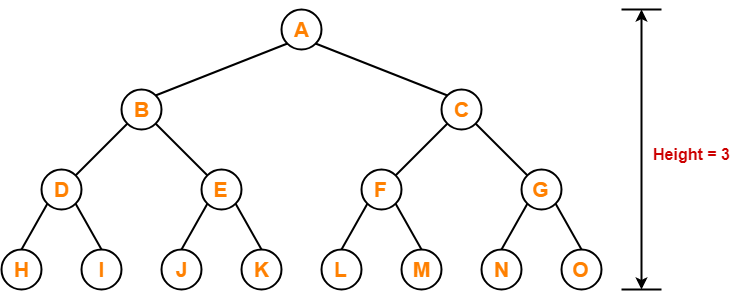
Maximum number of nodes in a binary tree of height 3

= 23+1 – 1

= 16 – 1

= 15 nodes

Thus, in a binary tree of height = 3, maximum number of nodes that can be inserted = 15.



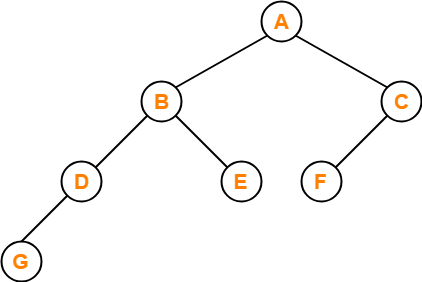
We can not insert more number of nodes in this binary tree.

**Property-03:**

|  |
| --- |
| **Total Number of leaf nodes in a Binary Tree**  **= Total Number of nodes with 2 children + 1** |

**Example-**

Consider the following binary tree-



Here,

* Number of leaf nodes = 3
* Number of nodes with 2 children = 2

Clearly, number of leaf nodes is one greater than number of nodes with 2 children.

This verifies the above relation.

|  |
| --- |
| **NOTE**  It is interesting to note that-  Number of leaf nodes in any binary tree depends only on the number of nodes with 2 children. |

**Property-04:**

|  |
| --- |
| **Maximum number of nodes at any level ‘L’ in a binary tree**  **= 2L** |

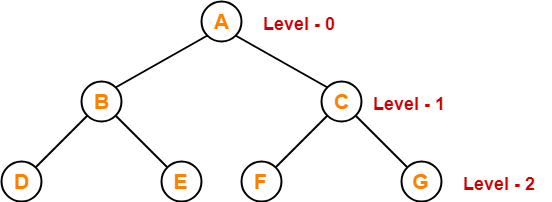
**Example-**

Maximum number of nodes at level-2 in a binary tree

= 22

= 4

Thus, in a binary tree, maximum number of nodes that can be present at level-2 = 4.



**PRACTICE PROBLEMS BASED ON BINARY TREE PROPERTIES-**

**Problem-01:**

A binary tree T has n leaf nodes. The number of nodes of degree-2 in T is \_\_\_\_\_\_?

1. log2n
2. n-1
3. n
4. 2n

**Solution-**

Using property-3, we have-

Number of degree-2 nodes

= Number of leaf nodes – 1

= n – 1

Thus, Option (B) is correct.

**Problem-02:**

In a binary tree, for every node the difference between the number of nodes in the left and right subtrees is at most 2. If the height of the tree is h > 0, then the minimum number of nodes in the tree is \_\_\_\_\_\_?

1. 2h-1 = 2 3-1= 2 2 =4
2. 2h-1 + 1 = 4+1=5
3. 2h – 1 2 3 -1 = 8-1 =7
4. 2h 8

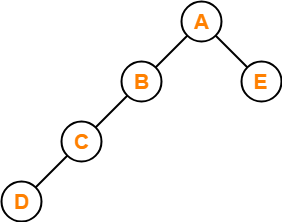
**Solution-**

Let us assume any random value of h. Let h = 3.

Then the given options reduce to-

1. 4
2. 5
3. 7
4. 8

Now, consider the following binary tree with height h = 3-



* This binary tree satisfies the question constraints.
* It is constructed using minimum number of nodes.

Thus, Option (B) is correct.

**Problem-03:**

In a binary tree, the number of internal nodes of degree-1 is 5 and the number of internal nodes of degree-2 is 10. The number of leaf nodes in the binary tree is \_\_\_\_\_\_?

1. 10
2. 11
3. 12
4. 15

Number of leaf nodes in the Binary tree = number of internal nodes of degree-2+1

= 10 +1 =11

**Solution-**

Using property-3, we have-

Number of leaf nodes in a binary tree

= Number of degree-2 nodes + 1

= 10 + 1

= 11

Thus, Option (B) is correct.

**Problem-04:**

The height of a binary tree is the maximum number of edges in any root to leaf path. The maximum number of nodes in a binary tree of height h is \_\_\_\_\_\_?

1. 2h
2. 2h-1 – 1
3. 2h+1 – 1
4. 2h+1

**Solution-**

Using property-2, Option (C) is correct.

**Problem-05:**

A binary tree T has 20 leaves. The number of nodes in T having 2 children is \_\_\_\_\_\_?

**Solution-**

Using property-3, correct answer is 19.

# [Tree Traversal | Binary Tree Traversal](https://www.gatevidyalay.com/tree-traversal-binary-tree-traversal/)

* 1. Inorder ( R )
  2. Preorder ( R
  3. Post order( R)

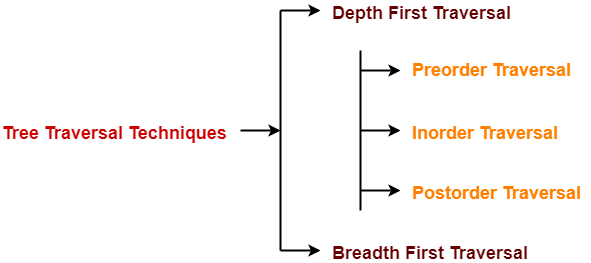
We have discussed-

* Binary tree is a special tree data structure.
* In a binary tree, each node can have at most 2 children.

## ****Tree Traversal-****

|  |
| --- |
| Tree Traversal refers to the process of visiting each node in a tree data structure exactly once. |

Various tree traversal techniques are-



## ****Depth First Traversal-****

Following three traversal techniques fall under Depth First Traversal-

1. Preorder Traversal
2. Inorder Traversal
3. Postorder Traversal

## ****1. Preorder Traversal-****

## ****Algorithm-****

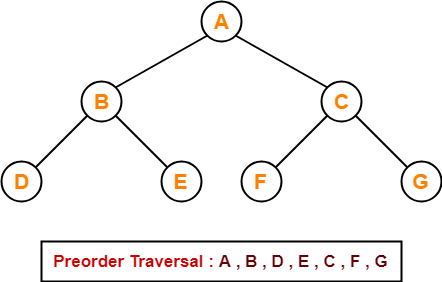
1. Visit the root
2. Traverse the left sub tree i.e. call Preorder (left sub tree)
3. Traverse the right sub tree i.e. call Preorder (right sub tree)

  ROOT LST RST

**Root**→**Left**→**Right**

## ****Example-****

Consider the following example-



LST 🡪 ROOT 🡪 RST

D🡪B🡪E🡪A🡪F🡪C🡪G

ROOT LST RST

A 🡪B🡪D🡪E🡪C🡪F🡪G

|  |
| --- |
| ****Preorder Traversal Shortcut****   Traverse the entire tree starting from the root node keeping yourself to the left.    https://www.gatevidyalay.com/wp-content/uploads/2018/07/Preorder-Traversal-Shortcut-1.png |

## ****Applications-****

* Preorder traversal is used to get prefix expression of an expression tree.
* Preorder traversal is used to create a copy of the tree.

## ****2. Inorder Traversal-****

## ****Algorithm-****

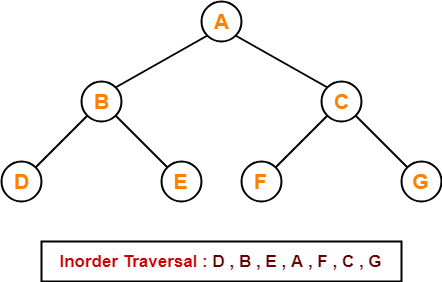
 LST 🡪 ROOT 🡪 RST

1. Traverse the left sub tree i.e. call Inorder (left sub tree)
2. Visit the root
3. Traverse the right sub tree i.e. call Inorder (right sub tree)

**Left**→**Root**→**Right**

## ****Example-****

Consider the following example-



 D🡪B🡪E🡪A🡪F🡪C🡪G

LST 🡪 RST 🡪 ROOT

D 🡪 E 🡪 B 🡪 F 🡪 G 🡪 C 🡪 A

|  |
| --- |
| ****Inorder Traversal Shortcut****   Keep a plane mirror horizontally at the bottom of the tree and take the projection of all the nodes.    https://www.gatevidyalay.com/wp-content/uploads/2018/07/Inorder-Traversal-Shortcut-1.png |

## ****Application-****

* Inorder traversal is used to get infix expression of an expression tree.

## ****3. Postorder Traversal-****

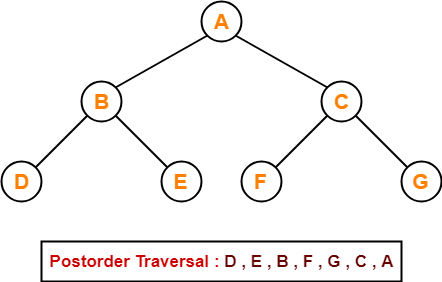
## ****Algorithm-****

1. Traverse the left sub tree i.e. call Postorder (left sub tree)
2. Traverse the right sub tree i.e. call Postorder (right sub tree)
3. Visit the root

**Left**→**Right**→**Root**

## ****Example-****

Consider the following example-



 D 🡪 E 🡪 B 🡪 F 🡪 G 🡪 C 🡪 A

|  |
| --- |
| ****Postorder Traversal Shortcut****   Pluck all the leftmost leaf nodes one by one.    https://www.gatevidyalay.com/wp-content/uploads/2018/07/Postorder-Traversal-Shortcut-1.png |

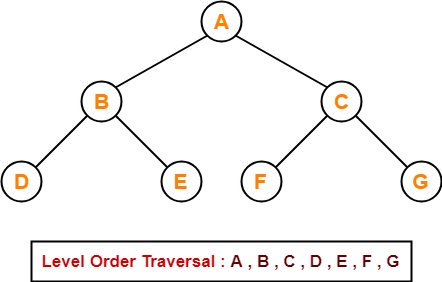
## ****Applications-****

* Postorder traversal is used to get postfix expression of an expression tree.
* Postorder traversal is used to delete the tree.
* This is because it deletes the children first and then it deletes the parent.

## ****Breadth First Traversal-****

* Breadth First Traversal of a tree prints all the nodes of a tree level by level.
* Breadth First Traversal is also called as **Level Order Traversal**.

## ****Example-****



In order🡪 Left -🡪 ROOT 🡪RIGHT

Pre order 🡪 ROOT🡪 Left -🡪 RIGHT

Post order🡪 Left 🡪 RIGHT🡪 ROOT

 .

A🡪B🡪C🡪D🡪E🡪F🡪G

## ****Application-****

* Level order traversal is used to print the data in the same order as stored in the array representation of a complete binary tree.

# [Binary Search Tree | Example | Construction](https://www.gatevidyalay.com/binary-search-trees-data-structures/)

## ****Binary Tree-****

We have discussed-

* Binary tree is a special tree data structure.
* In a binary tree, each node can have at most 2 children.
* In a binary tree, nodes may be arranged in any random order.

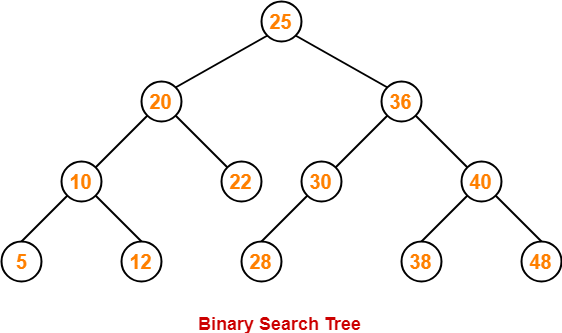
## ****Binary Search Tree-****

|  |
| --- |
| Binary Search Tree is a special kind of binary tree in which nodes are arranged in a specific order. |

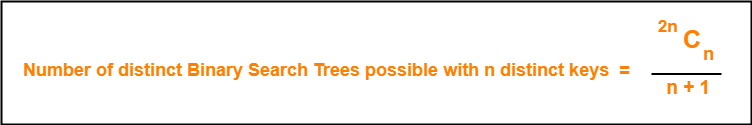
In a binary search tree (BST), each node contains-

* Only smaller values in its left sub tree
* Only larger values in its right sub tree

## ****Example-****



## ****Number of Binary Search Trees-****



## ****Example-****

Number of distinct binary search trees possible with 3 distinct keys

= 2×3C3 / 3+1

= 6C3 / 4

= FAC 6/((FAC 3 \* FAC(6-3))/4

=6 x 5 x 4 x fac 3/ (fac 3 x fac 3)/4

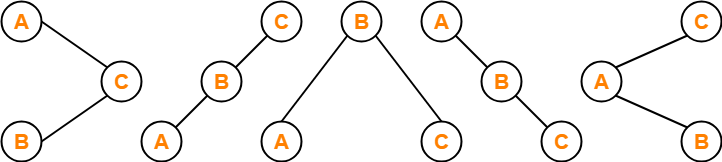
=6 x 5 x 4 / fac ¾

= 6 x 5 x 4/ 3 x 2 x ¼

= 5

= 5

If three distinct keys are A, B and C, then 5 distinct binary search trees are-



## ****Binary Search Tree Construction-****

Let us understand the construction of a binary search tree using the following example-

## ****Example-****

Construct a Binary Search Tree (BST) for the following sequence of numbers-

50, 70, 60, 20, 90, 10, 40, 100

When elements are given in a sequence,

* Always consider the first element as the root node.
* Consider the given elements and insert them in the BST one by one.

The binary search tree will be constructed as explained below-

50

20 70

### **10 40 60 90**

### **100**

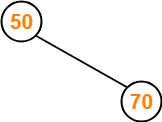
### 

### ****Insert 50-****

https://www.gatevidyalay.com/wp-content/uploads/2018/07/Binary-Search-Tree-Construction-Step-01.png

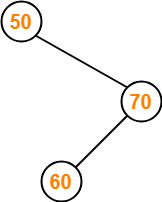
### ****Insert 70-****

* As 70 > 50, so insert 70 to the right of 50.



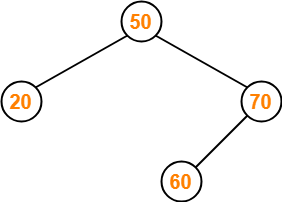
### ****Insert 60-****

* As 60 > 50, so insert 60 to the right of 50.
* As 60 < 70, so insert 60 to the left of 70.



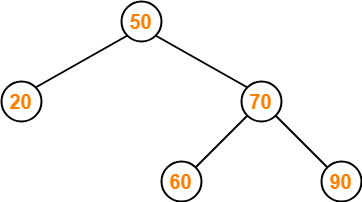
### ****Insert 20-****

* As 20 < 50, so insert 20 to the left of 50.



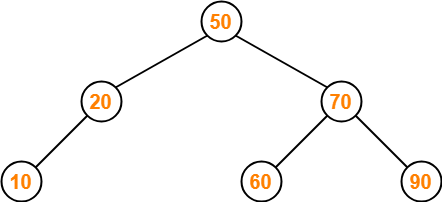
### ****Insert 90-****

* As 90 > 50, so insert 90 to the right of 50.
* As 90 > 70, so insert 90 to the right of 70.



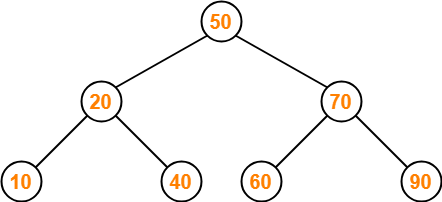
### ****Insert 10-****

* As 10 < 50, so insert 10 to the left of 50.
* As 10 < 20, so insert 10 to the left of 20.



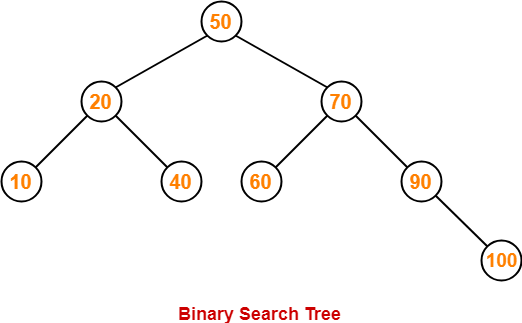
### ****Insert 40-****

* As 40 < 50, so insert 40 to the left of 50.
* As 40 > 20, so insert 40 to the right of 20.



### ****Insert 100-****

* As 100 > 50, so insert 100 to the right of 50.
* As 100 > 70, so insert 100 to the right of 70.
* As 100 > 90, so insert 100 to the right of 90.



50

20 70

### **10 40 60 90**

### **100**

### 

## ****PRACTICE PROBLEMS BASED ON BINARY SEARCH TREES-****

## ****Problem-01:****

A binary search tree is generated by inserting in order of the following integers-

50, 15, 62, 5, 20, 58, 91, 3, 8, 37, 60, 24

50

15 62

5 20 58 91

3 8 37 60

24

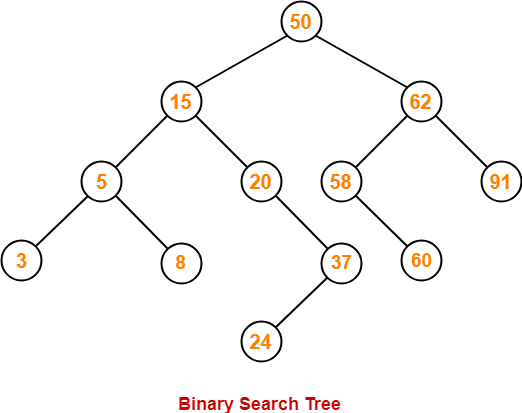
The number of nodes in the left subtree and right subtree of the root respectively is \_\_\_\_\_.

1. (4, 7)
2. (7, 4)
3. (8, 3)
4. (3, 8)

## ****Solution-****

Using the above discussed steps, we will construct the binary search tree.

The resultant binary search tree will be-



50

15 62

5 20 58 91

3 8 37 60

24

Clearly,

* Number of nodes in the left subtree of the root = 7
* Number of nodes in the right subtree of the root = 4

Thus, Option (B) is correct.

## ****Problem-02:****

How many distinct binary search trees can be constructed out of 4 distinct keys?

1. 5
2. 14
3. 24
4. 35

## ****Solution-****

 2n C n /n+1

= 8 C 4 /5

= fac 8 / (fac 4 x fac(8-4))/5

= 8 x 7 x 6 x5 x fac 4 / (fac 4 x fac4))/5

= 8 x 7 x 6 x5 / (fac4))/5

= 2 x 7

=14

Number of distinct binary search trees possible with 4 distinct keys

= 2nCn / n+1

= 2×4C4 / 4+1

= 8C4 / 5

= 14

Thus, Option (B) is correct.

## ****Problem-03:****

The numbers 1, 2, …, n are inserted in a binary search tree in some order. In the resulting tree, the right subtree of the root contains p nodes. The first number to be inserted in the tree must be-

1. p
2. p+1
3. n-p
4. n-p+1

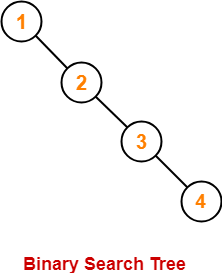
## ****Solution-****

Let n = 4 and p = 3.

Then, given options reduce to-

1. 3
2. 4
3. 1
4. 2

Our binary search tree will be as shown-



Clearly, first inserted number = 1.

Thus, Option (C) is correct.

## ****Problem-04:****

We are given a set of n distinct elements and an unlabeled binary tree with n nodes. In how many ways can we populate the tree with given set so that it becomes a binary search tree?

1. 0
2. 1
3. n!
4. C(2n, n) / n+1

## ****Solution-****

Option (B) is correct.

# [Binary Search Tree Traversal | BST Traversal](https://www.gatevidyalay.com/binary-search-tree-traversal-bst-traversal/)

## ****Binary Search Tree-****

.

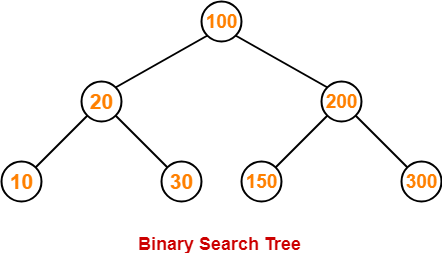
Binary search tree (BST) is a special kind of binary tree where each node contains-

* Only larger values in its right subtree.
* Only smaller values in its left subtree.

## ****BST Traversal-****

* A binary search tree is traversed in exactly the same way a binary tree is traversed.
* In other words, BST traversal is same as binary tree traversal.

Consider the following binary search tree-



Now, let us write the traversal sequences for this binary search tree-

Inorder: LEFT🡪ROOT🡪RIGHT

10🡪20🡪30🡪100🡪150🡪200🡪300

Preorder: ROOT🡪 LEFT🡪 RIGHT

100🡪20🡪10🡪30🡪200🡪150🡪300

Postorder: LEFT 🡪RIGHT🡪ROOT

10🡪30🡪20🡪150🡪300🡪200🡪100

### ****Preorder Traversal-****

100 , 20 , 10 , 30 , 200 , 150 , 300

Preorder: ROOT🡪 LEFT🡪 RIGHT

100🡪20🡪10🡪30🡪200🡪150🡪300

### ****Inorder Traversal-****

10 , 20 , 30 , 100 , 150 , 200 , 300

Inorder: LEFT🡪ROOT🡪RIGHT

10🡪20🡪30🡪100🡪150🡪200🡪300

### ****Postorder Traversal-****

10 , 30 , 20 , 150 , 300 , 200 , 100

Postorder: LEFT 🡪RIGHT🡪ROOT

10🡪30🡪20🡪150🡪300🡪200🡪100

## ****Important Notes-****

## ****Note-01:****

* Inorder traversal of a binary search tree always yields all the nodes in increasing order.

## ****Note-02:****

Unlike [**Binary Trees**](https://www.gatevidyalay.com/binary-tree-types-of-trees-in-data-structure/),

* A binary search tree can be constructed using only preorder or only postorder traversal result.
* This is because inorder traversal can be obtained by sorting the given result in increasing order.

## ****PRACTICE PROBLEMS BASED ON BST TRAVERSAL-****

## ****Problem-01:****

Suppose the numbers 7 , 5 , 1 , 8 , 3 , 6 , 0 , 9 , 4 , 2 are inserted in that order into an initially empty binary search tree. The binary search tree uses the usual ordering on natural numbers.

What is the inorder traversal sequence of the resultant tree?

1. 7 , 5 , 1 , 0 , 3 , 2 , 4 , 6 , 8 , 9
2. 0 , 2 , 4 , 3 , 1 , 6 , 5 , 9 , 8 , 7
3. 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9
4. 9 , 8 , 6 , 4 , 2 , 3 , 0 , 1 , 5 , 7

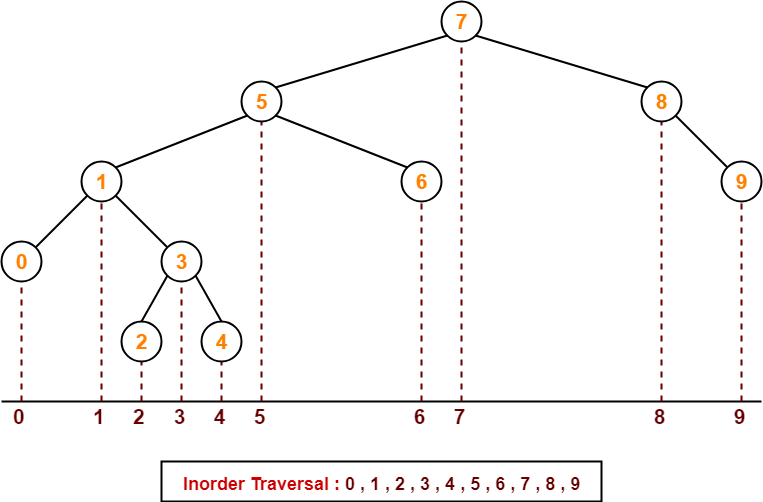
## ****Solution-****

This given problem may be solved in the following two ways-

### ****Method-01:****

* We construct a binary search tree for the given elements.
* We write the inorder traversal sequence from the binary search tree so obtained.

Following these steps, we have-



Thus, Option (C) is correct.

### ****Method-02:****

We know, inorder traversal of a binary search tree always yields all the nodes in increasing order.

Using this result,

* We arrange all the given elements in increasing order.
* Then, we report the sequence so obtained as inorder traversal sequence.

|  |
| --- |
| **Inorder Traversal :**  **0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9** |

Thus, Option (C) is correct.

## ****Problem-02:****

The preorder traversal sequence of a binary search tree is-

30 , 20 , 10 , 15 , 25 , 23 , 39 , 35 , 42

Preorder: Root🡪Left🡪Right

What one of the following is the postorder traversal sequence of the same tree

1. 10 , 20 , 15 , 23 , 25 , 35 , 42 , 39 , 30
2. 15 , 10 , 25 , 23 , 20 , 42 , 35 , 39 , 30
3. 15 , 20 , 10 , 23 , 25 , 42 , 35 , 39 , 30
4. 15 , 10 , 23 , 25 , 20 , 35 , 42 , 39 , 30

## ****Solution-****

In this question,

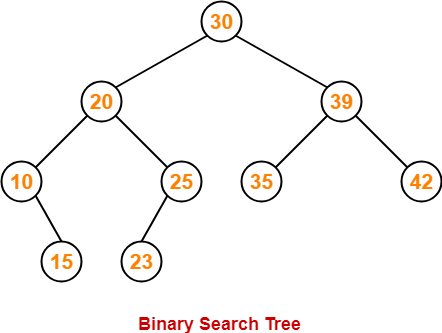
* We are provided with the preorder traversal sequence.
* We write the inorder traversal sequence by arranging all the numbers in ascending order.

Then-

* Postorder Traversal : 30 , 20 , 10 , 15 , 25 , 23 , 39 , 35 , 42
* Inorder Traversal : 10 , 15 , 20 , 23 , 25 , 30 , 35 , 39 , 42

Now,

* We draw a binary search tree using these traversal results.
* The binary search tree so obtained is as shown-



Now, we write the postorder traversal sequence-

|  |
| --- |
| **Postorder Traversal :**  **15 , 10 , 23 , 25, 20, 35, 42, 39, 30** |

Thus, Option (D) is correct.

# [Binary Search Tree Insertion | BST Deletion](https://www.gatevidyalay.com/binary-search-tree-insertion-bst-deletion/)

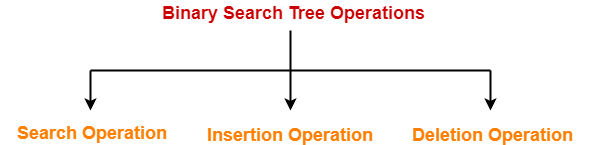
## ****Binary Search Tree-****

Binary search tree (BST) is a special kind of binary tree where each node contains-

* Only larger values in its right subtree.
* Only smaller values in its left subtree.

## ****Binary Search Tree Operations-****

Commonly performed operations on binary search tree are-



1. Search Operation
2. Insertion Operation
3. Deletion Operation

## ****1. Search Operation-****

|  |
| --- |
| Search Operation is performed to search a particular element in the Binary Search Tree. |

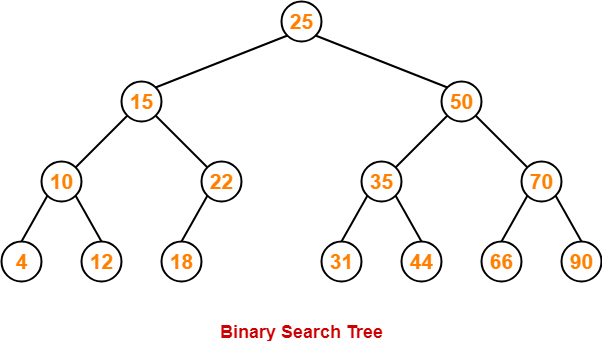
## ****Rules-****

For searching a given key in the BST,

* Compare the key with the value of root node.
* If the key is present at the root node, then return the root node.
* If the key is greater than the root node value, then recur for the root node’s right subtree.
* If the key is smaller than the root node value, then recur for the root node’s left subtree.

## ****Example-****

Consider key = 45 has to be searched in the given BST-



* We start our search from the root node 25.
* As 45 > 25, so we search in 25’s right subtree.
* As 45 < 50, so we search in 50’s left subtree.
* As 45 > 35, so we search in 35’s right subtree.
* As 45 > 44, so we search in 44’s right subtree but 44 has no subtrees.
* So, we conclude that 45 is not present in the above BST.

## ****2. Insertion Operation-****

|  |
| --- |
| Insertion Operation is performed to insert an element in the Binary Search Tree. |

## ****Rules-****

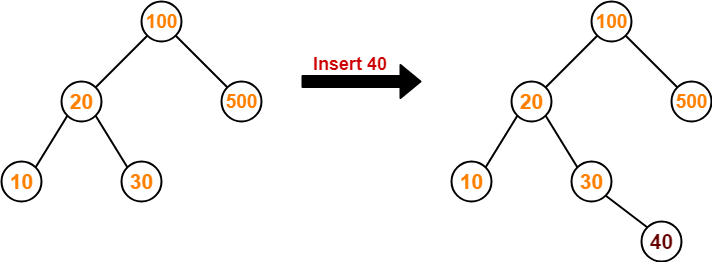
The insertion of a new key always takes place as the child of some leaf node.

For finding out the suitable leaf node,

* Search the key to be inserted from the root node till some leaf node is reached.
* Once a leaf node is reached, insert the key as child of that leaf node.

## ****Example-****

Consider the following example where key = 40 is inserted in the given BST-



* We start searching for value 40 from the root node 100.
* As 40 < 100, so we search in 100’s left subtree.
* As 40 > 20, so we search in 20’s right subtree.
* As 40 > 30, so we add 40 to 30’s right subtree.

## ****3. Deletion Operation-****

|  |
| --- |
| Deletion Operation is performed to delete a particular element from the Binary Search Tree. |

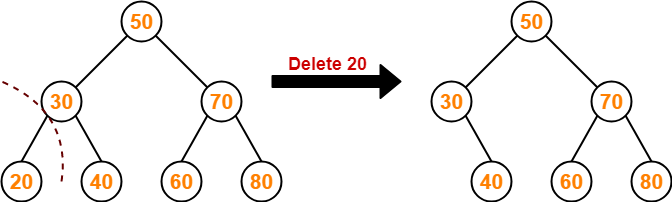
When it comes to deleting a node from the binary search tree, following three cases are possible-

### ****Case-01: Deletion Of A Node Having No Child (Leaf Node)-****

Just remove / disconnect the leaf node that is to deleted from the tree.

### ****Example-****

Consider the following example where node with value = 20 is deleted from the BST-

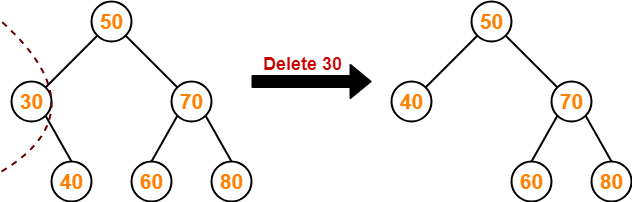


### ****Case-02: Deletion Of A Node Having Only One Child-****

Just make the child of the deleting node, the child of its grandparent.

### ****Example-****

Consider the following example where node with value = 30 is deleted from the BST-



### ****Case-02: Deletion Of A Node Having Two Children-****

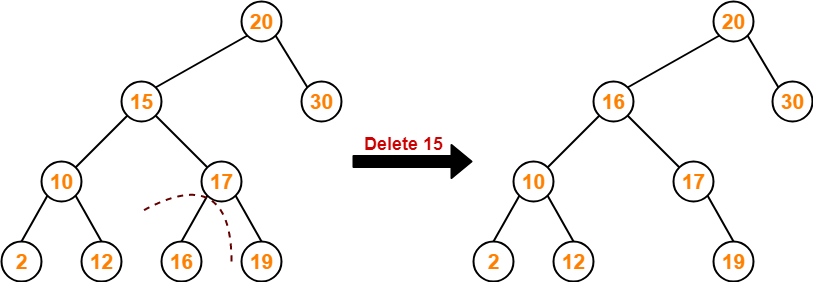
A node with two children may be deleted from the BST in the following two ways-

### ****Method-01:****

* Visit to the right subtree of the deleting node.
* Pluck the least value element called as inorder successor.
* Replace the deleting element with its inorder successor.

### ****Example-****

Consider the following example where node with value = 15 is deleted from the BST-

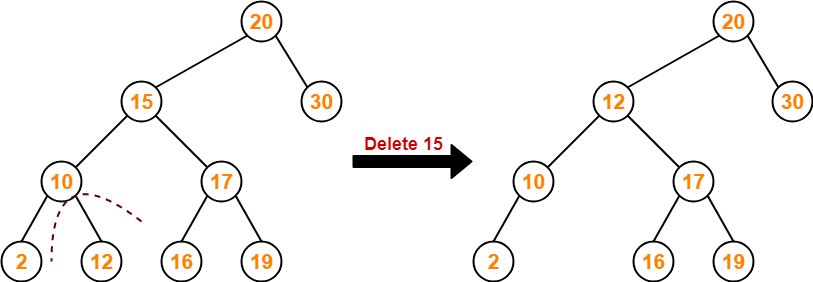


### ****Method-02:****

* Visit to the left subtree of the deleting node.
* Pluck the greatest value element called as inorder predecessor.
* Replace the deleting element with its inorder predecessor.

### ****Example-****

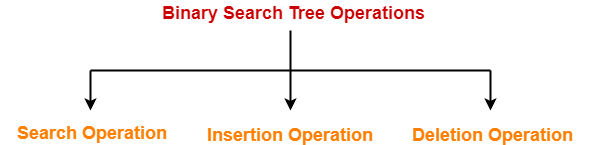
Consider the following example where node with value = 15 is deleted from the BST-



# [Time Complexity of Binary Search Tree](https://www.gatevidyalay.com/time-complexity-of-bst-binary-search-tree/)

## ****Binary Search Tree-****

Commonly performed operations on binary search tree are-



1. Search Operation
2. Insertion Operation
3. Deletion Operation

## ****Time Complexity-****

* Time complexity of all BST Operations = O(h).
* Here, h = Height of binary search tree

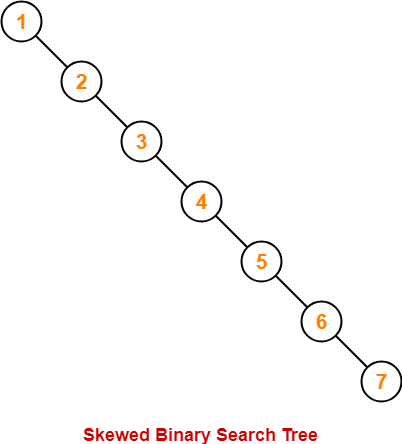
Now, let us discuss the worst case and best case.

### ****Worst Case-****

In worst case,

* The binary search tree is a skewed binary search tree.
* Height of the binary search tree becomes n.
* So, Time complexity of BST Operations = O(n).

In this case, binary search tree is as good as unordered list with no benefits.



### ****Best Case-****

In best case,

* The binary search tree is a balanced binary search tree.
* Height of the binary search tree becomes log(n).
* So, Time complexity of BST Operations = O(logn).

