Binary Search Tree

Guided By:-
Mrs. Darshana Mistry

Presented By:-
Dharita Chokshi
Disha Raval
Himani Patel

Outlines

- Tree
- Binary tree Implementation
- Binary Search Tree
- BST Operations
- Traversal
- Insertion
- Deletion
- Types of BST
- Complexity in BST
- Applications of BST

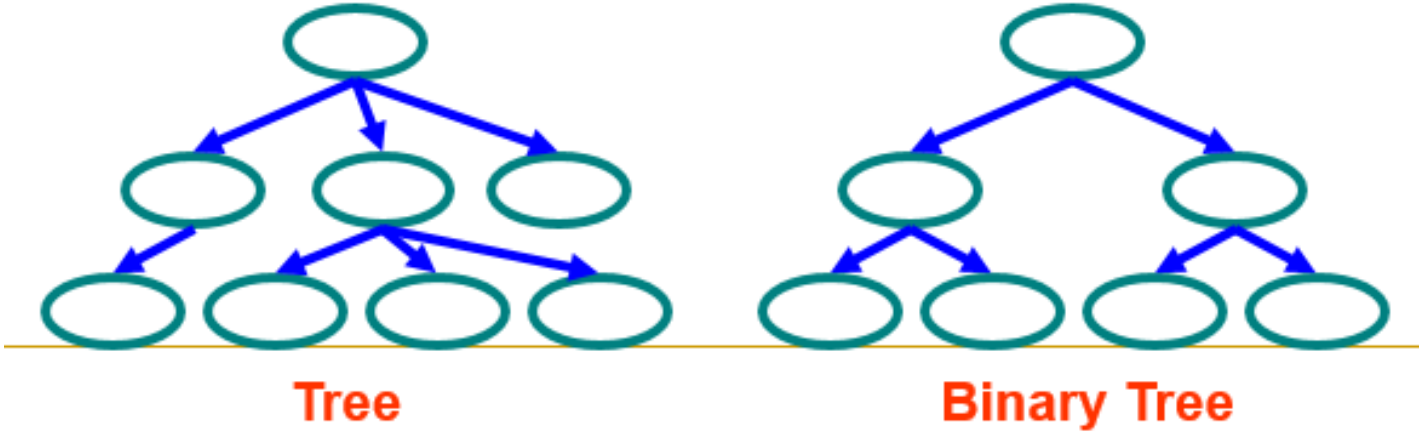
Trees

Tree

- Each node can have 0 or more children
- A node can have at most one parent

Binary tree

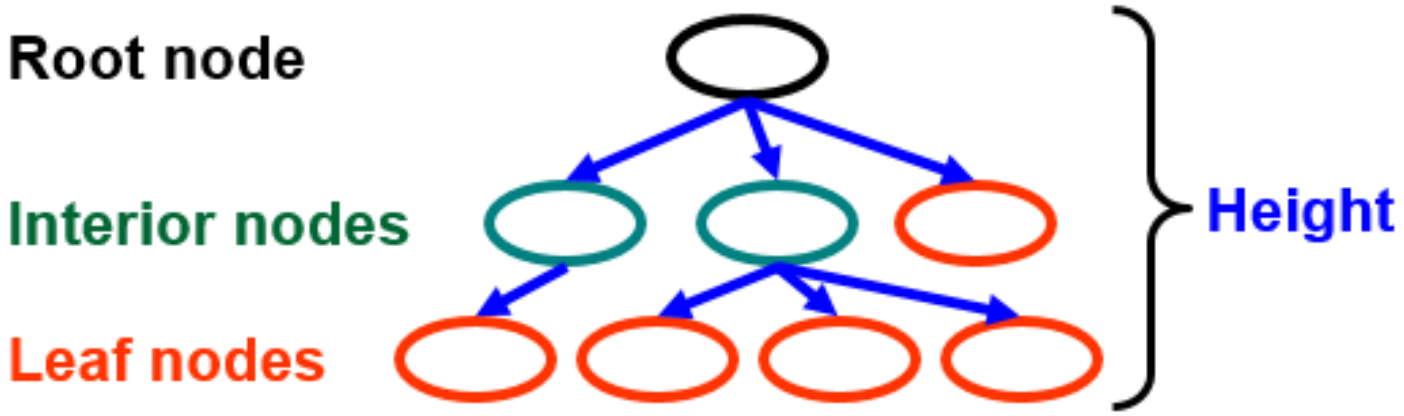
- Tree with 0–2 children per node
- Also known as Decision Making Tree



Trees

Terminology

- Root $\Rightarrow$ no parent
- Leaf $\Rightarrow$ no child
- Interior $\Rightarrow$ non-leaf
- Height $\Rightarrow$ distance from root to leaf (H)



Why is h important?

The Tree operations like insert, delete, retrieve etc. are typically expressed in terms of the height of the tree **h**.
So, it can be stated that the tree height **h** determines running time!

Binary Tree Implementation

```
Class Node
{
        int data; // Could be int, a class, etc
        Node *left, *right;          // null if empty

        void insert ( int data ) { … }
        void delete ( int data ) { … }
        Node *find ( int data ) { … }
            …
}
```

Binary Search Tree

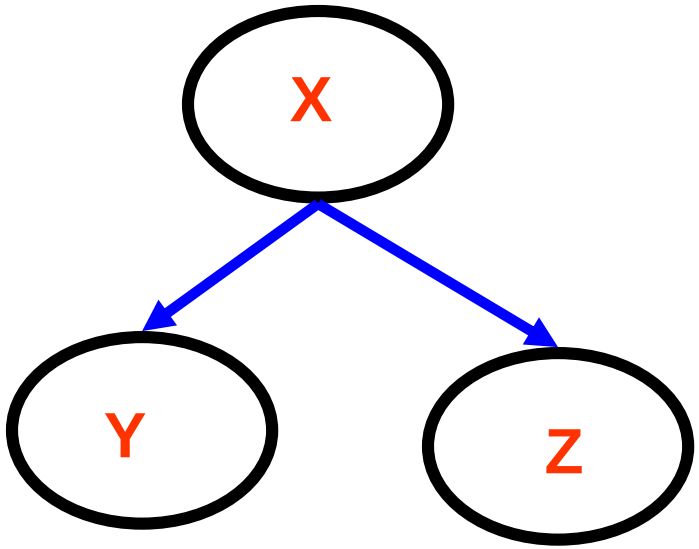
Key property is value at node

- Smaller values in left subtree
- Larger values in right subtree

Example

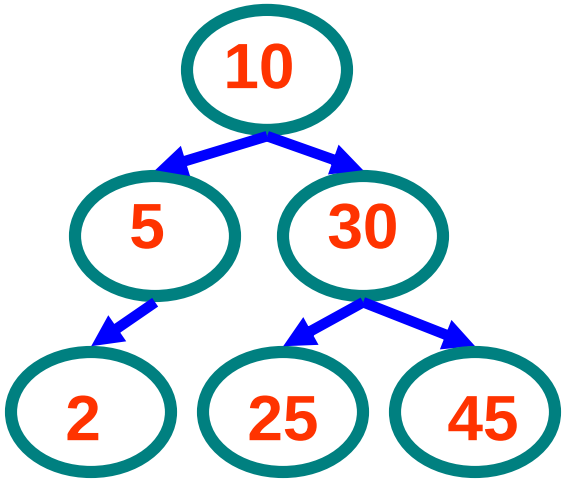
X > Y

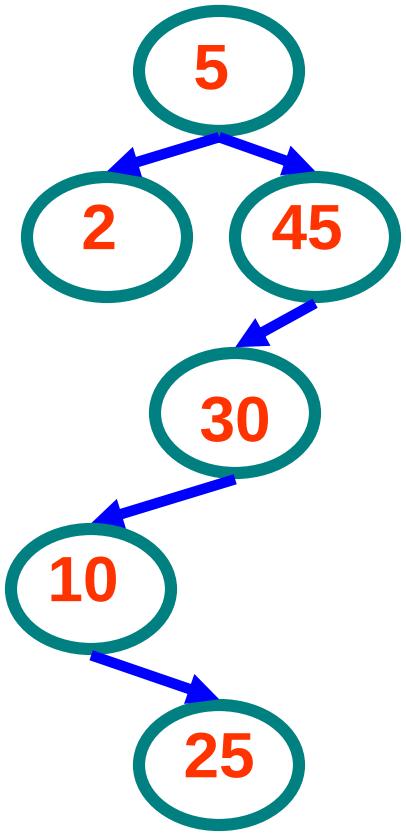
X < Z



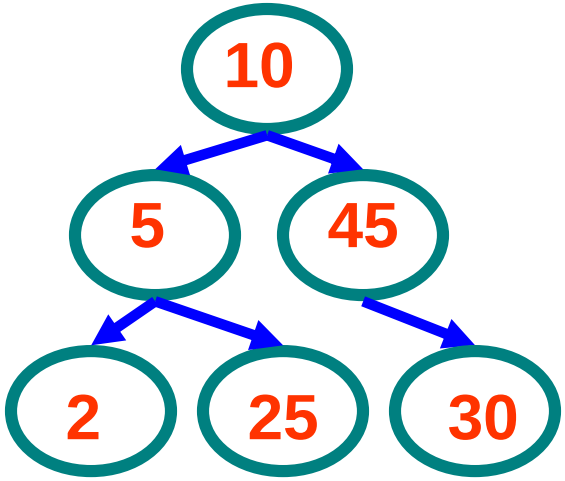
Binary Search Tree

Examples





Binary search trees



Not a binary search tree

Difference between BT and BST

A binary tree is simply a tree in which each node can have at most two children.

A binary search tree is a binary tree in which the nodes are assigned values, with the following restrictions :

1. No duplicate values.
2. The left subtree of a node can only have values less than the node
3. The right subtree of a node can only have values greater than the node and recursively defined
4. The left subtree of a node is a binary search tree.
5. The right subtree of a node is a binary search tree.

Binary Tree Search Algorithm

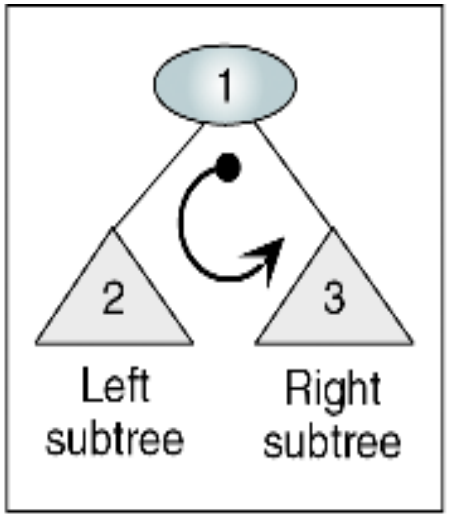
```
TREE-SEARCH(x,k)
If x==NIL or k==x.key
        return x
If k < x.key
        return TREE-SEARCH(x.left,k)
else
        return TREE-SEARCH(x.right,k)
```

BST Operations

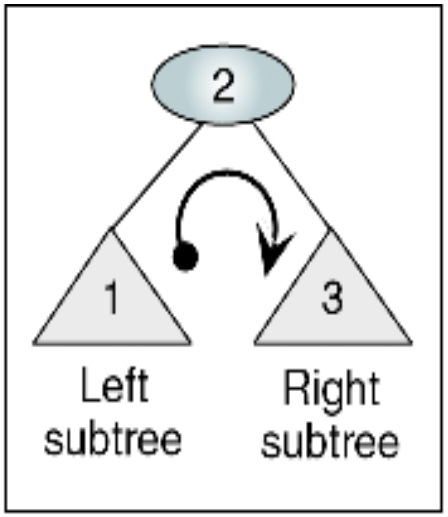
Four basic BST operations

1. Traversal
2. Search
3. Insertion
4. Deletion

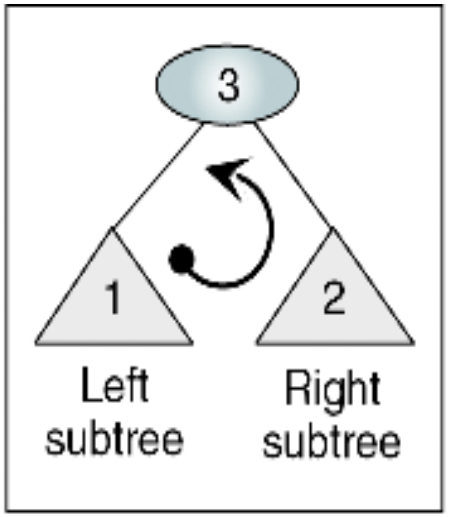
BST Traversal



(a) Preorder traversal



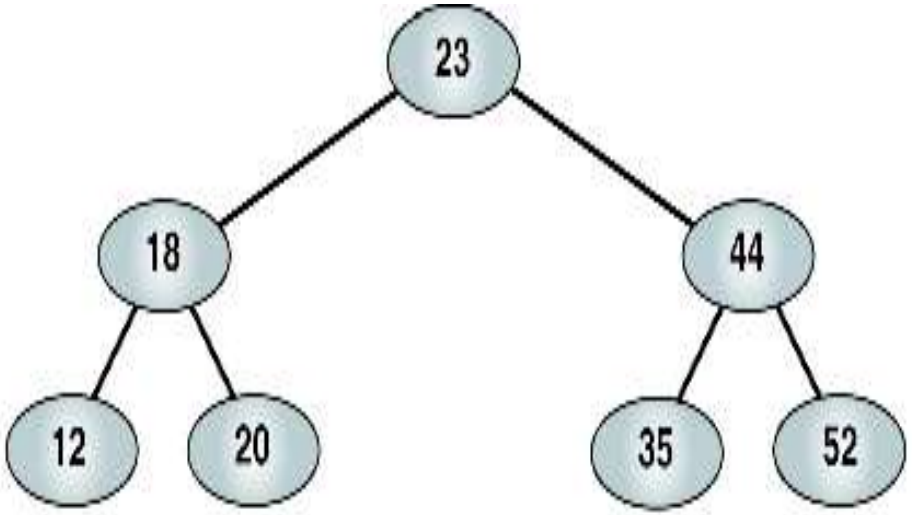
(b) Inorder traversal



(c) Postorder traversal

Preorder Traversal

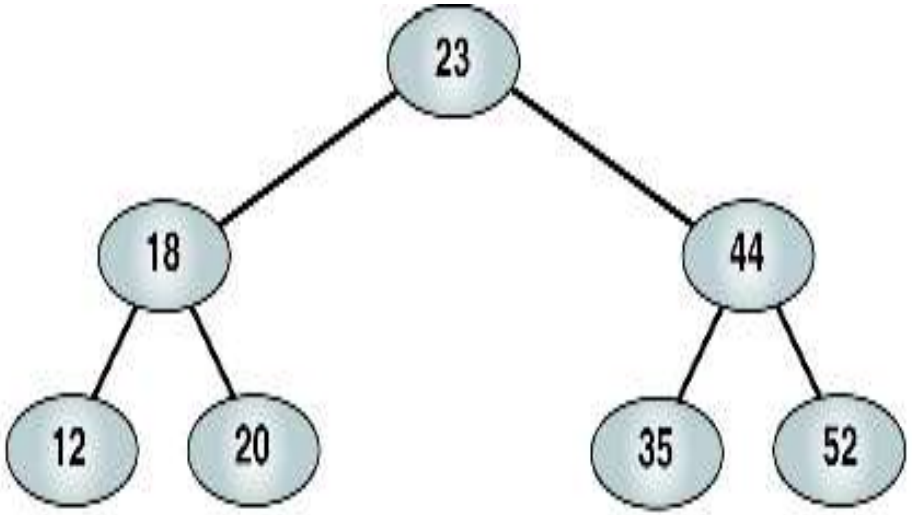
Root Left Right



23 18 12 20 44 35 52

Postorder Traversal

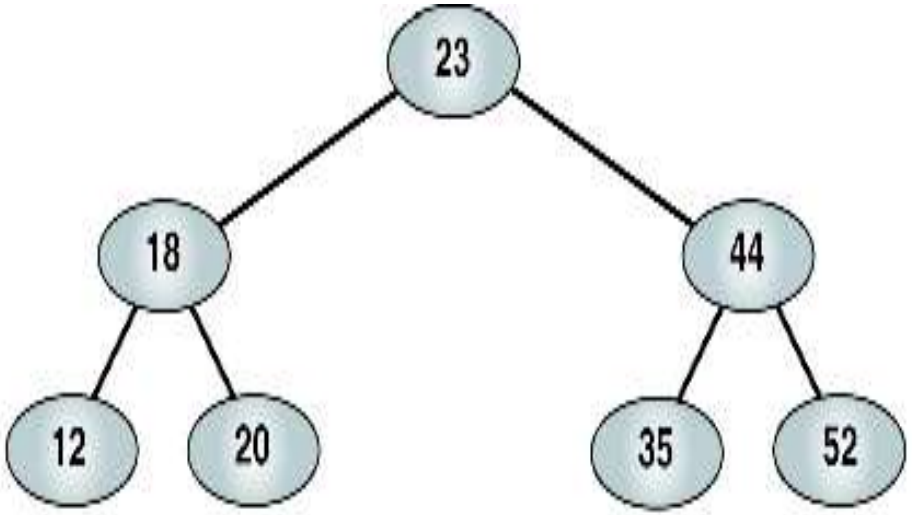
Left Right Root



12 20 18 35 52 44 23

Inorder Traversal

Left Root Right

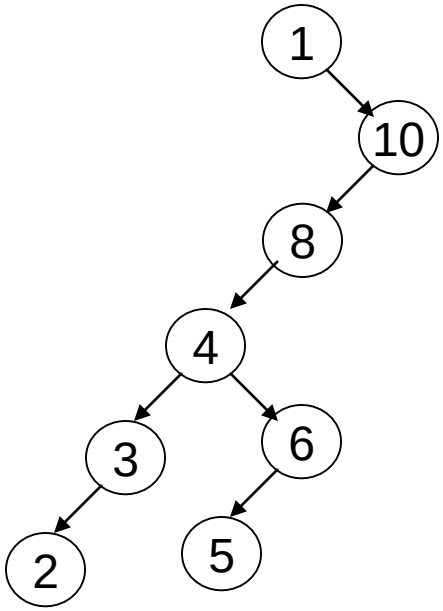


12 18 20 23 35 44 52

Produces a sequenced list

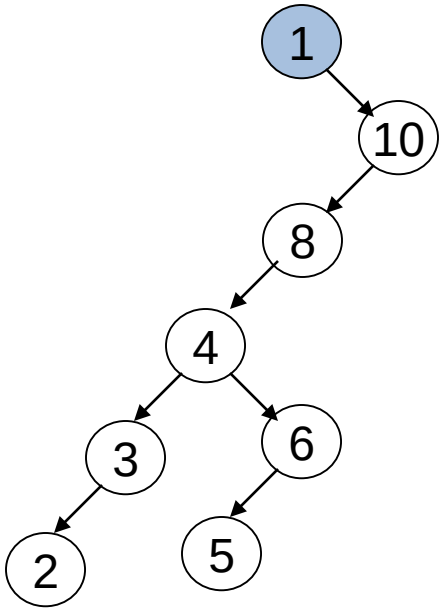
Inorder Traversal

1	10	8	4	6	3	2	5



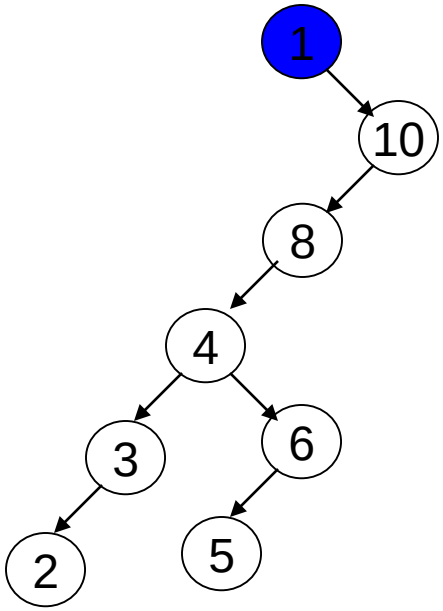
Inorder Traversal

1	10	8	4	6	3	2	5



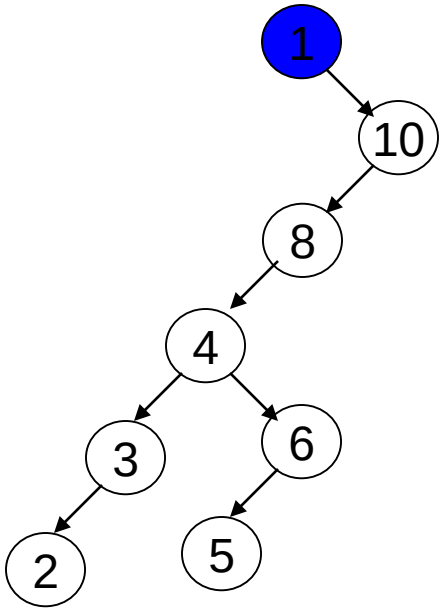
Inorder Traversal

1	10	8	4	6	3	2	5



Inorder Traversal

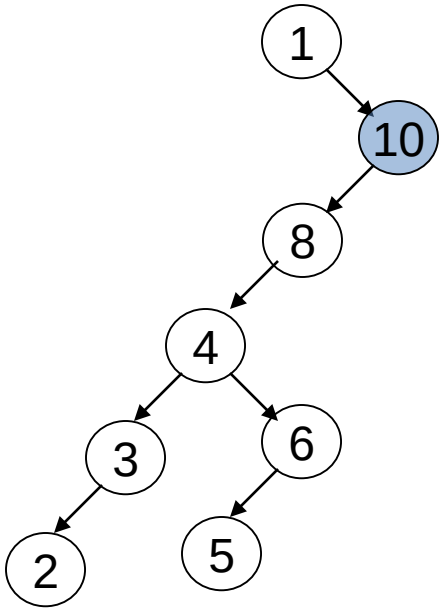
1	10	8	4	6	3	2	5



1

Inorder Traversal

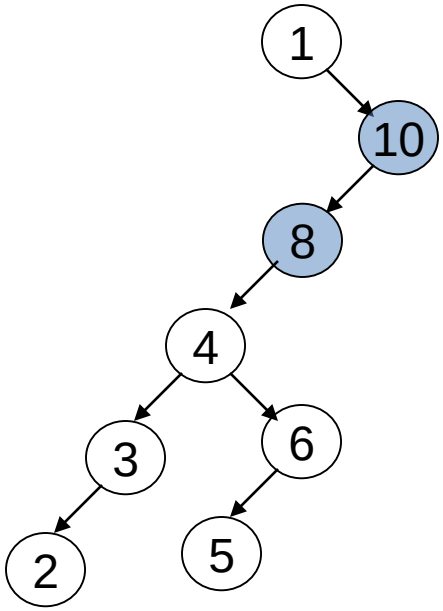
1	10	8	4	6	3	2	5



1

Inorder Traversal

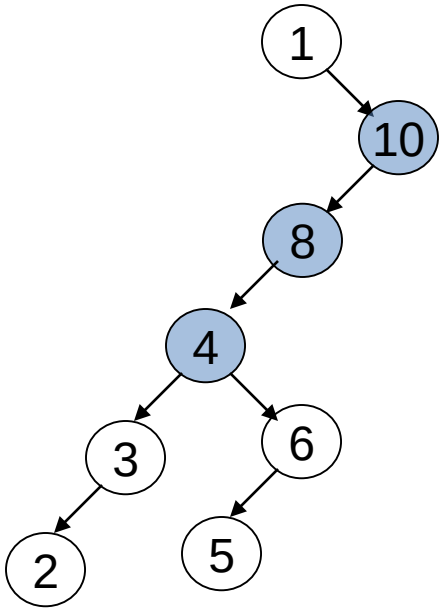
1	10	8	4	6	3	2	5



1

Inorder Traversal

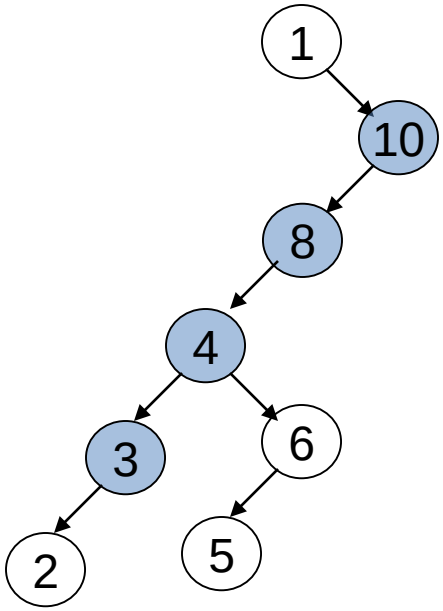
1	10	8	4	6	3	2	5



1

Inorder Traversal

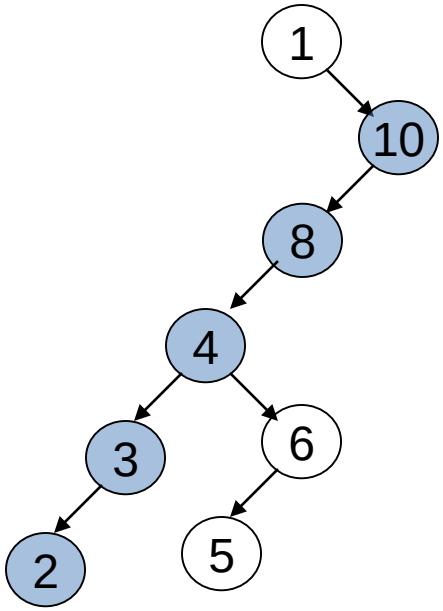
1	10	8	4	6	3	2	5



1

Inorder Traversal

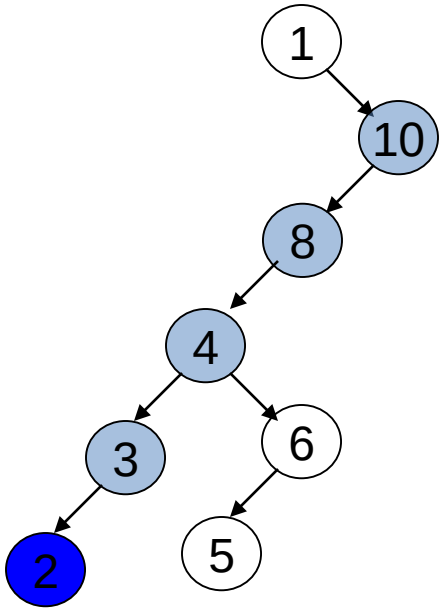
1	10	8	4	6	3	2	5



1

Inorder Traversal

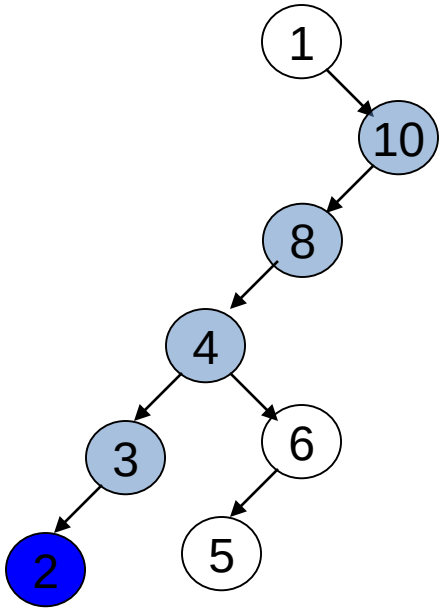
1	10	8	4	6	3	2	5



1

Inorder Traversal

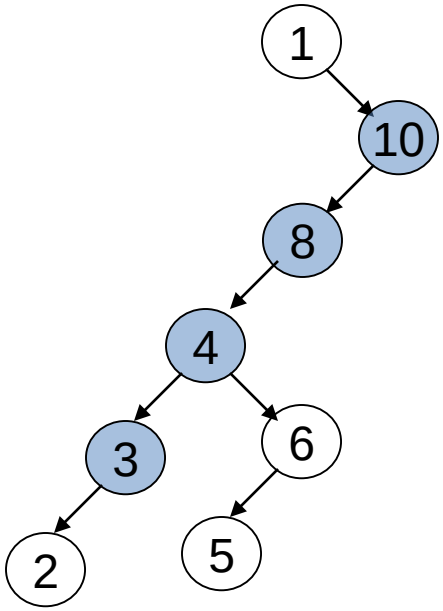
1	10	8	4	6	3	2	5



1	2

Inorder Traversal

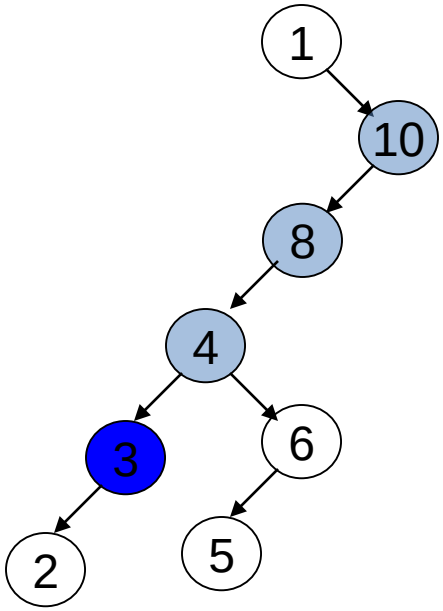
1	10	8	4	6	3	2	5



1	2

Inorder Traversal

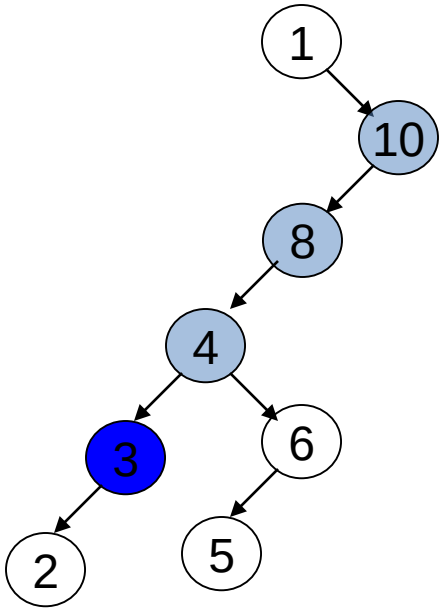
1	10	8	4	6	3	2	5



1	2

Inorder Traversal

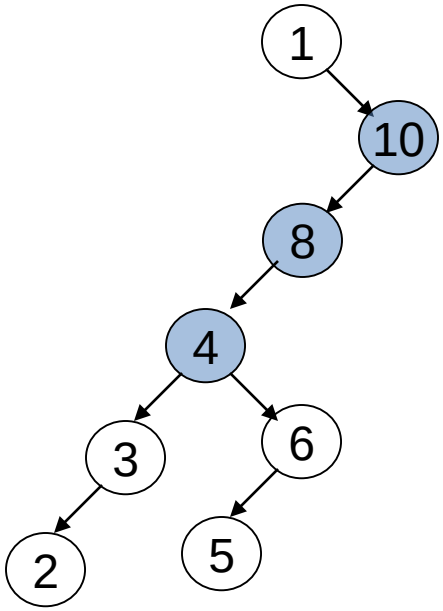
1	10	8	4	6	3	2	5



1	2	3

Inorder Traversal

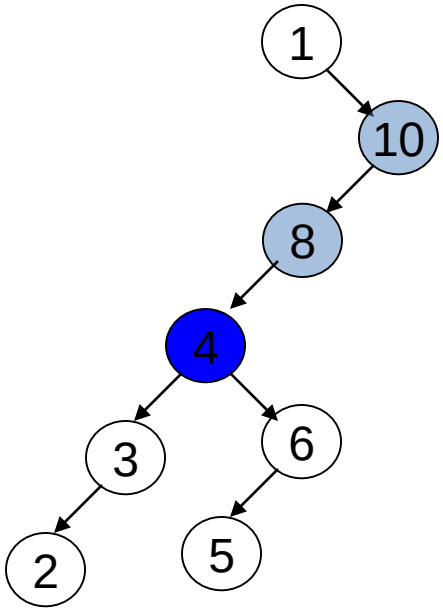
1	10	8	4	6	3	2	5



1	2	3

Inorder Traversal

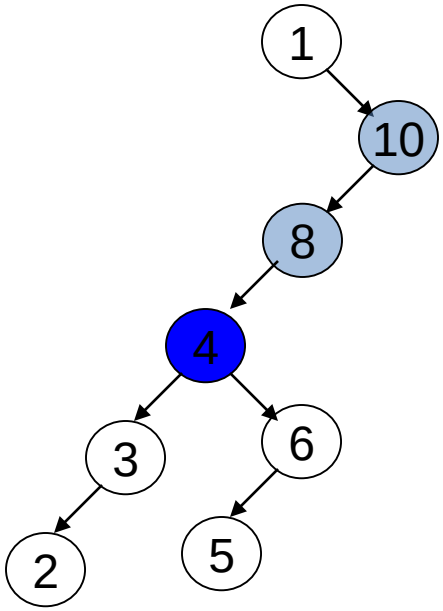
1	10	8	4	6	3	2	5



1	2	3

Inorder Traversal

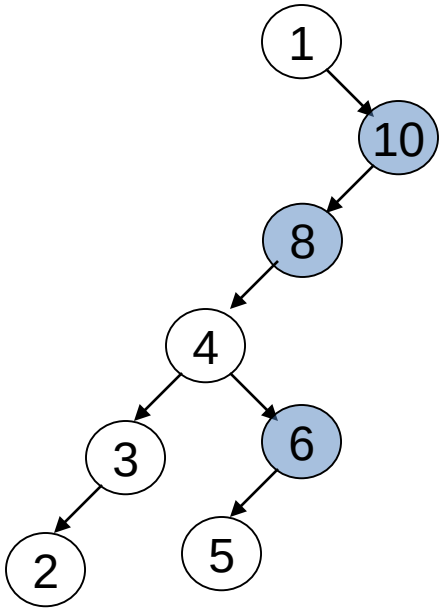
1	10	8	4	6	3	2	5



1	2	3	4

Inorder Traversal

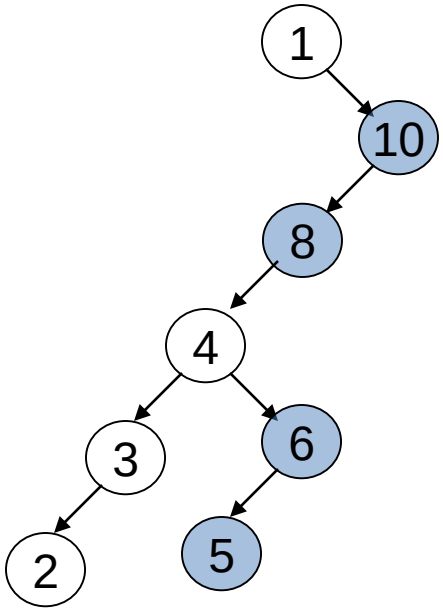
1	10	8	4	6	3	2	5



1	2	3	4

Inorder Traversal

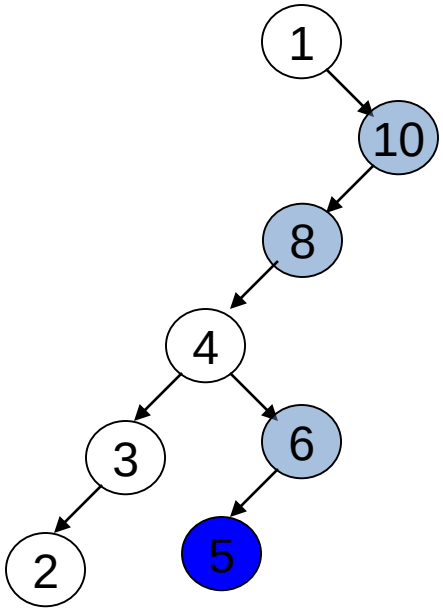
1	10	8	4	6	3	2	5



1	2	3	4

Inorder Traversal

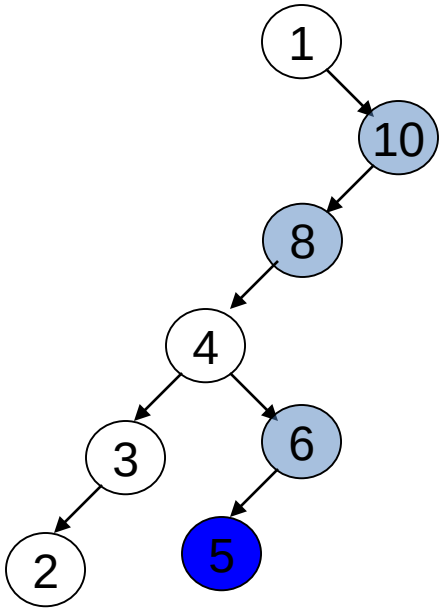
1	10	8	4	6	3	2	5



1	2	3	4

Inorder Traversal

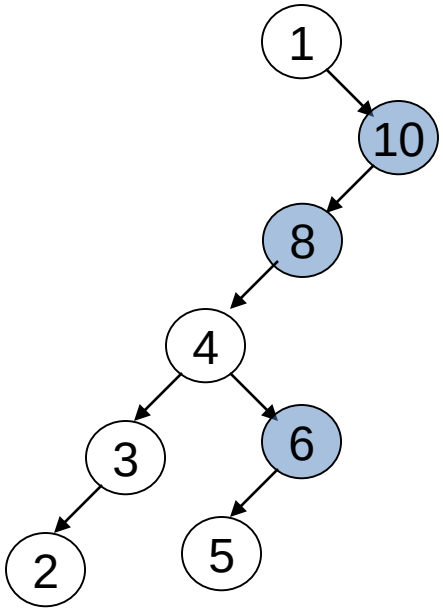
1	10	8	4	6	3	2	5



1	2	3	4	5

Inorder Traversal

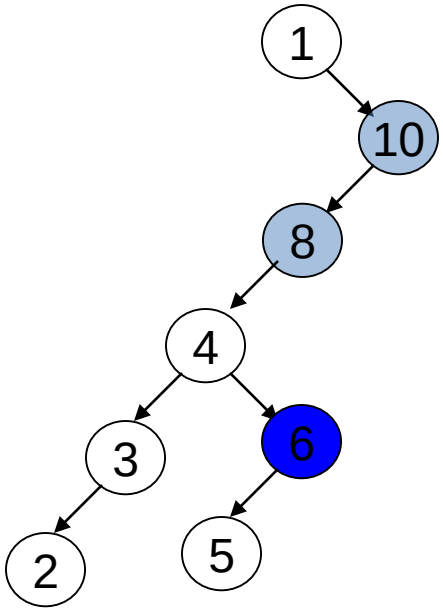
1	10	8	4	6	3	2	5



1	2	3	4	5

Inorder Traversal

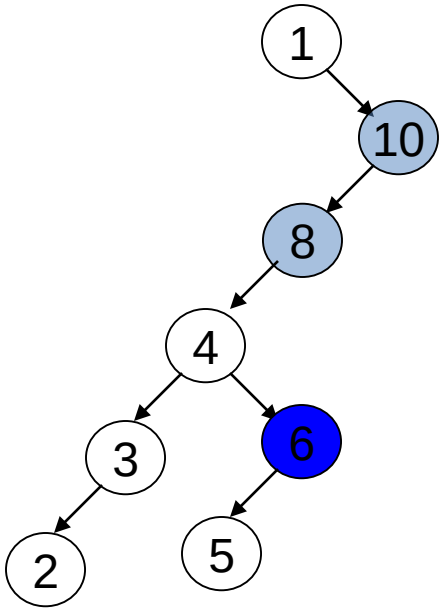
1	10	8	4	6	3	2	5



1	2	3	4	5

Inorder Traversal

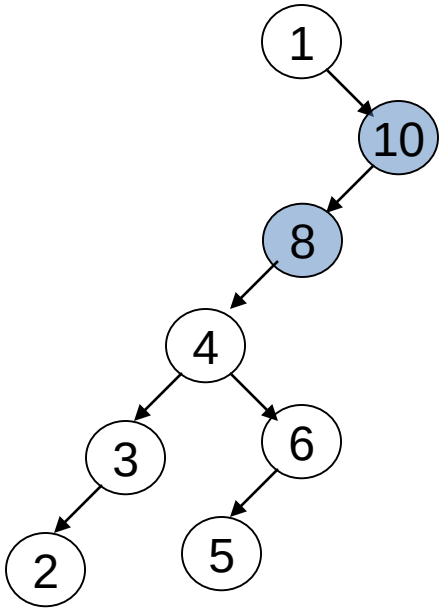
1	10	8	4	6	3	2	5



1	2	3	4	5	6

Inorder Traversal

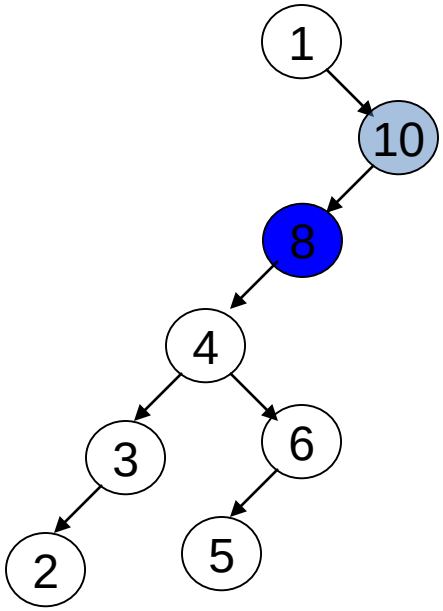
1	10	8	4	6	3	2	5



1	2	3	4	5	6

Inorder Traversal

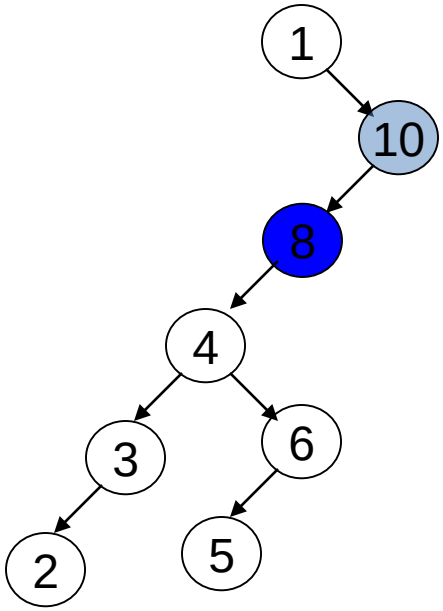
1	10	8	4	6	3	2	5



1	2	3	4	5	6

Inorder Traversal

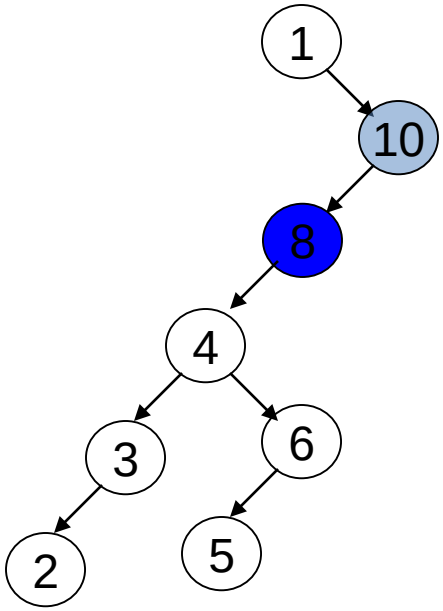
1	10	8	4	6	3	2	5



1	2	3	4	5	6

Inorder Traversal

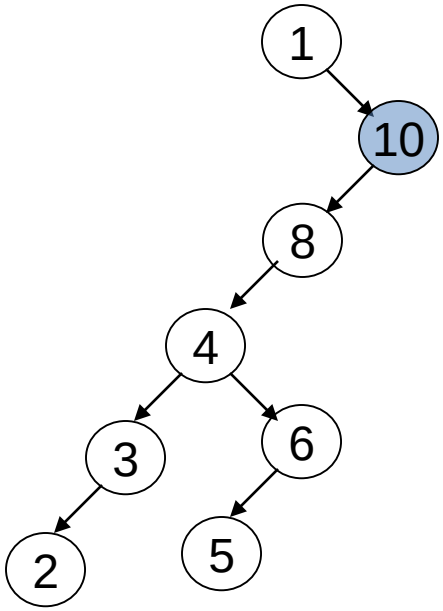
1	10	8	4	6	3	2	5



1	2	3	4	5	6	8

Inorder Traversal

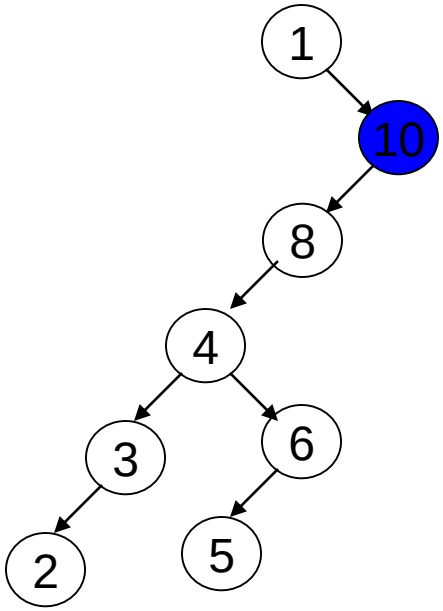
1	10	8	4	6	3	2	5



1	2	3	4	5	6	8

Inorder Traversal

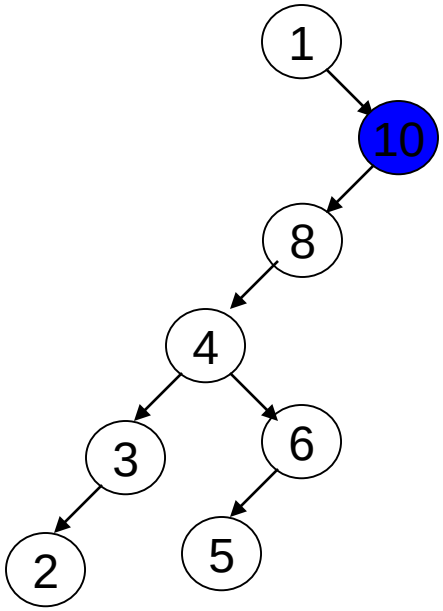
1	10	8	4	6	3	2	5



1	2	3	4	5	6	8

Inorder Traversal

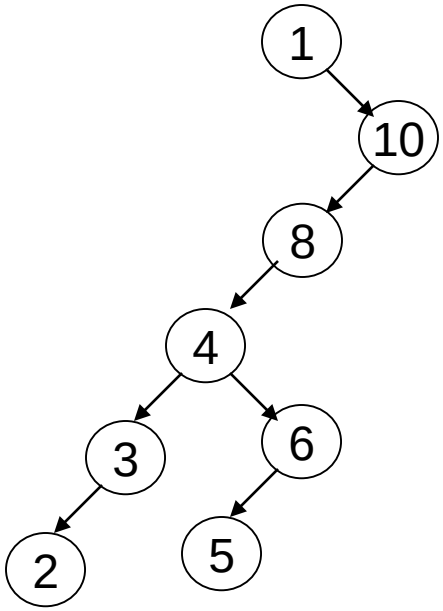
1	10	8	4	6	3	2	5



1	2	3	4	5	6	8	10

Inorder Traversal

1	10	8	4	6	3	2	5



1	2	3	4	5	6	8	10

Binary Tree Insertion

Binary Tree Insertion

1	10	8	4	6	3	2	5

Binary Tree Insertion

1	10	8	4	6	3	2	5

Binary Tree Insertion

1	10	8	4	6	3	2	5

Binary Tree Insertion

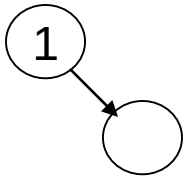
1	10	8	4	6	3	2	5

Binary Tree Insertion

1	10	8	4	6	3	2	5

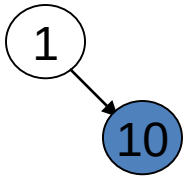
Binary Tree Insertion

1	10	8	4	6	3	2	5



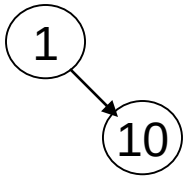
Binary Tree Insertion

1	10	8	4	6	3	2	5



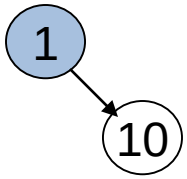
Binary Tree Insertion

1	10	8	4	6	3	2	5



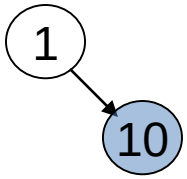
Binary Tree Insertion

1	10	8	4	6	3	2	5



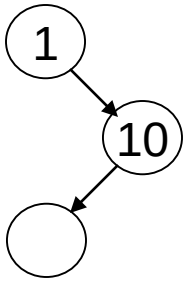
Binary Tree Insertion

1	10	8	4	6	3	2	5



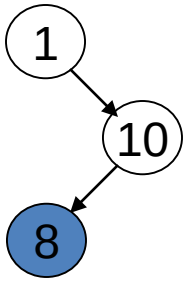
Binary Tree Insertion

1	10	8	4	6	3	2	5



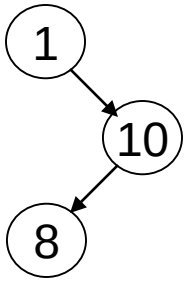
Binary Tree Insertion

1	10	8	4	6	3	2	5



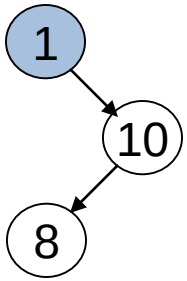
Binary Tree Insertion

1	10	8	4	6	3	2	5



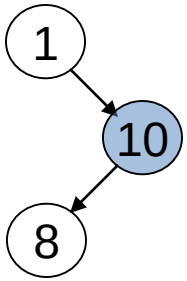
Binary Tree Insertion

1	10	8	4	6	3	2	5



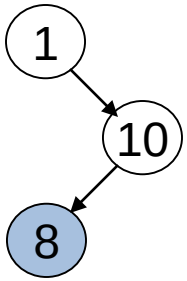
Binary Tree Insertion

1	10	8	4	6	3	2	5



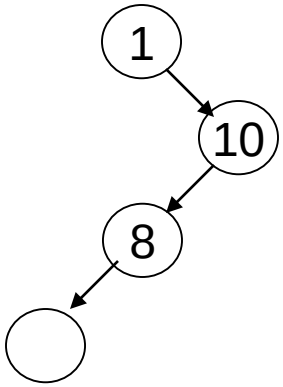
Binary Tree Insertion

1	10	8	4	6	3	2	5



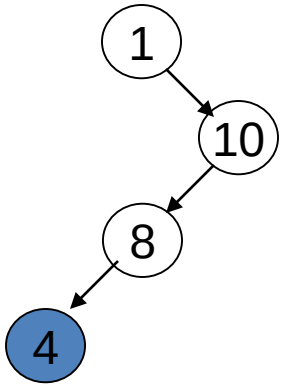
Binary Tree Insertion

1	10	8	4	6	3	2	5



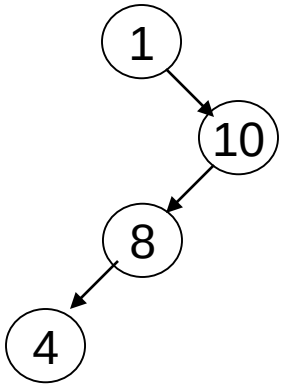
Binary Tree Insertion

1	10	8	4	6	3	2	5



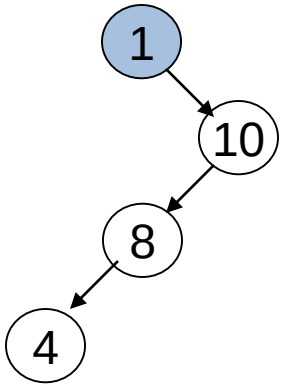
Binary Tree Insertion

1	10	8	4	6	3	2	5



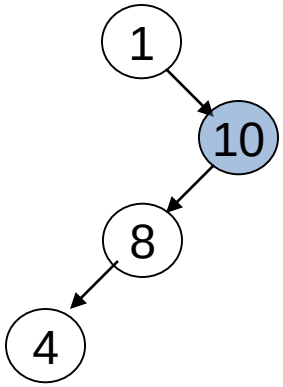
Binary Tree Insertion

1	10	8	4	6	3	2	5



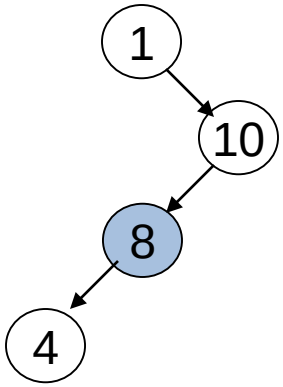
Binary Tree Insertion

1	10	8	4	6	3	2	5



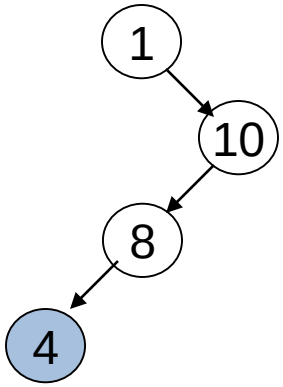
Binary Tree Insertion

1	10	8	4	6	3	2	5



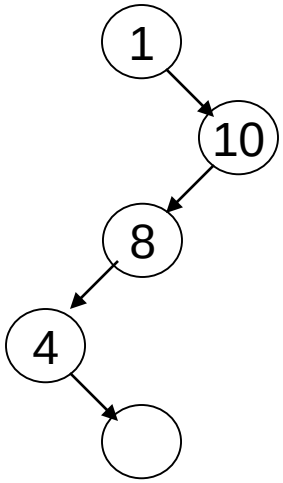
Binary Tree Insertion

1	10	8	4	6	3	2	5



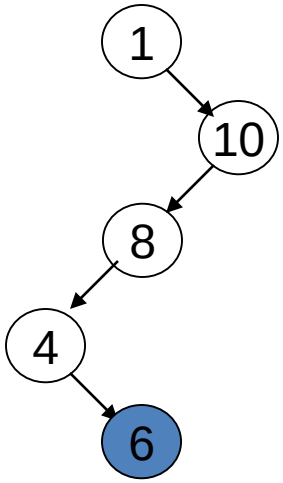
Binary Tree Insertion

1	10	8	4	6	3	2	5



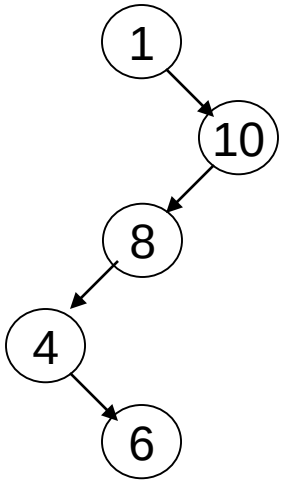
Binary Tree Insertion

1	10	8	4	6	3	2	5



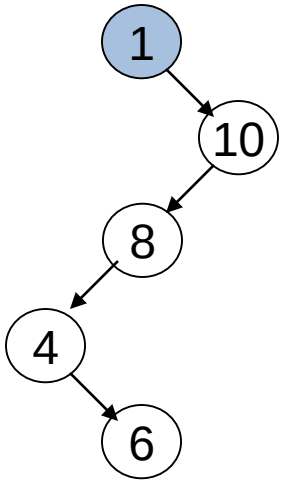
Binary Tree Insertion

1	10	8	4	6	3	2	5



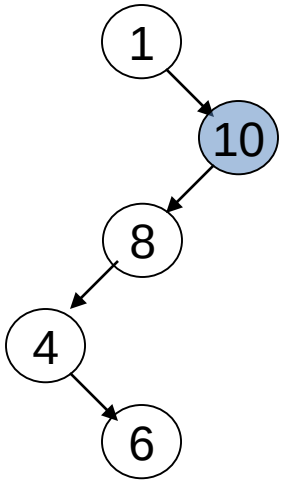
Binary Tree Insertion

1	10	8	4	6	3	2	5



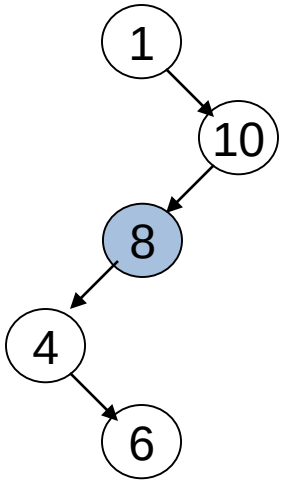
Binary Tree Insertion

1	10	8	4	6	3	2	5



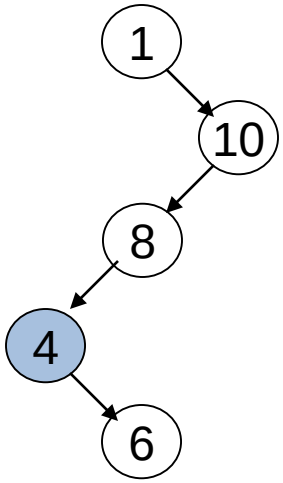
Binary Tree Insertion

1	10	8	4	6	3	2	5



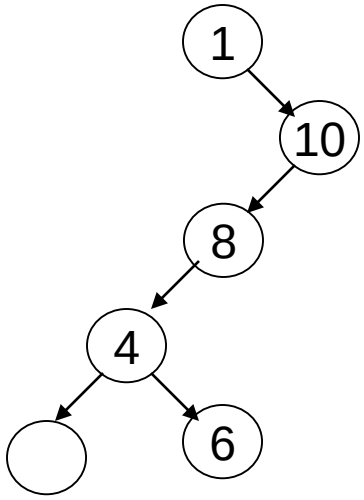
Binary Tree Insertion

1	10	8	4	6	3	2	5



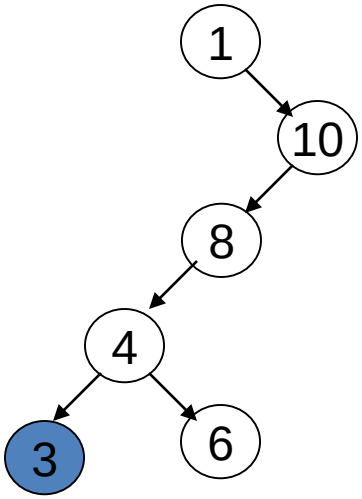
Binary Tree Insertion

1	10	8	4	6	3	2	5



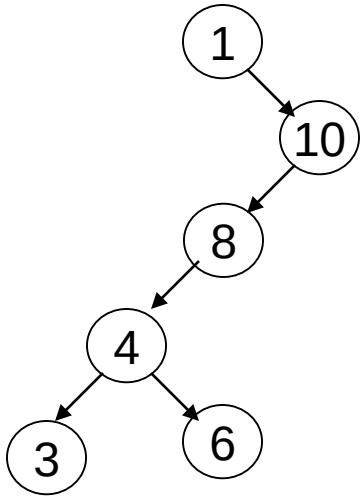
Binary Tree Insertion

1	10	8	4	6	3	2	5



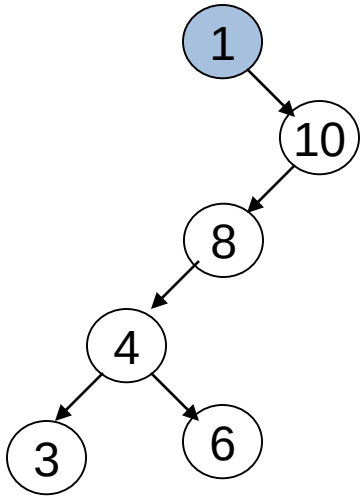
Binary Tree Insertion

1	10	8	4	6	3	2	5



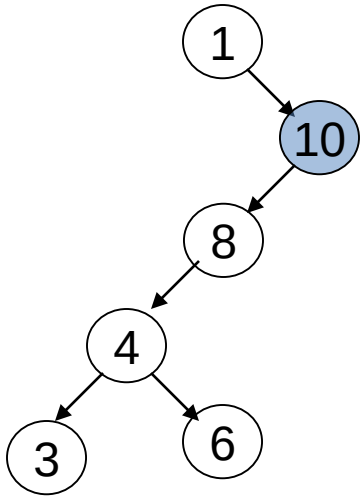
Binary Tree Insertion

1	10	8	4	6	3	2	5



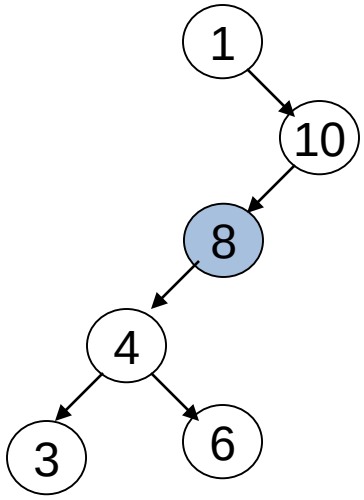
Binary Tree Insertion

1	10	8	4	6	3	2	5



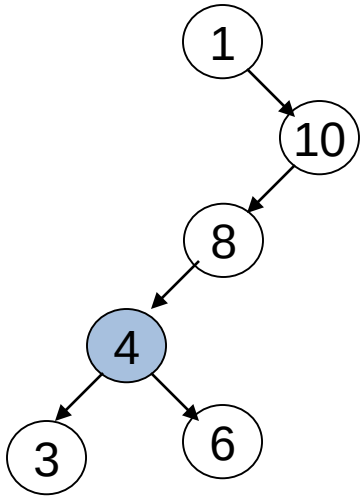
Binary Tree Insertion

1	10	8	4	6	3	2	5



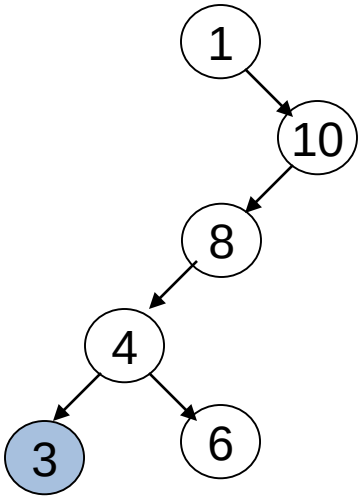
Binary Tree Insertion

1	10	8	4	6	3	2	5



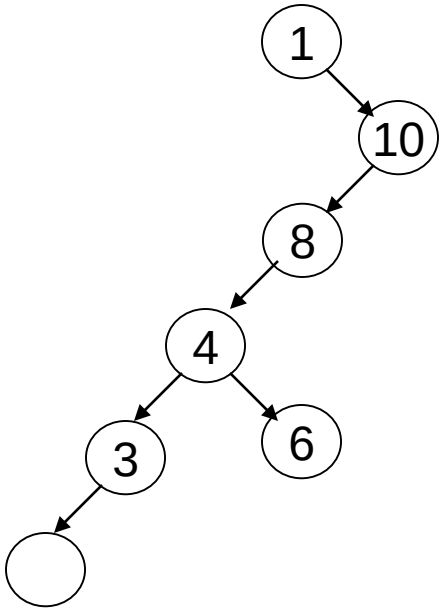
Binary Tree Insertion

1	10	8	4	6	3	2	5



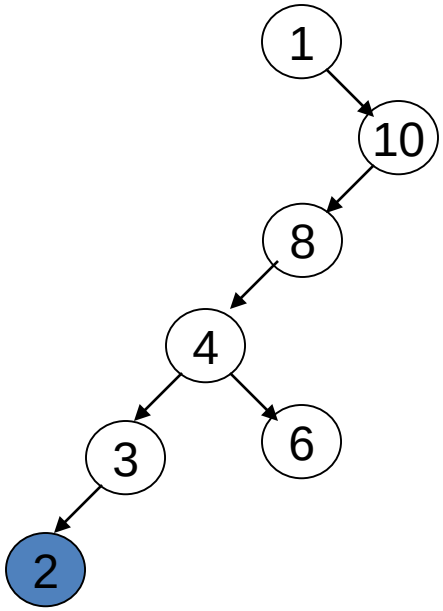
Binary Tree Insertion

1	10	8	4	6	3	2	5



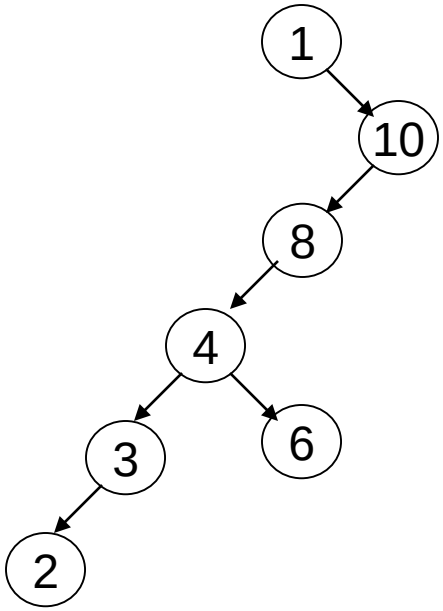
Binary Tree Insertion

1	10	8	4	6	3	2	5



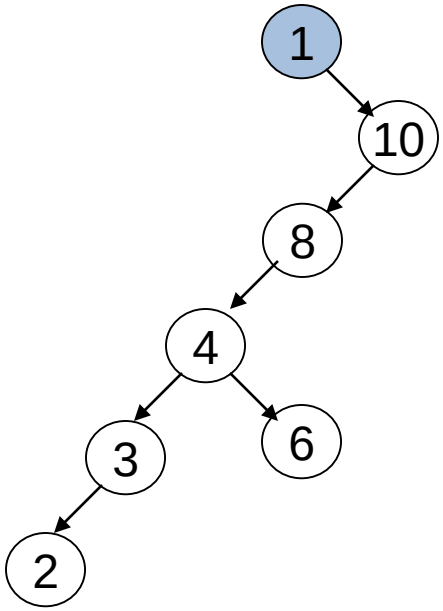
Binary Tree Insertion

1	10	8	4	6	3	2	5



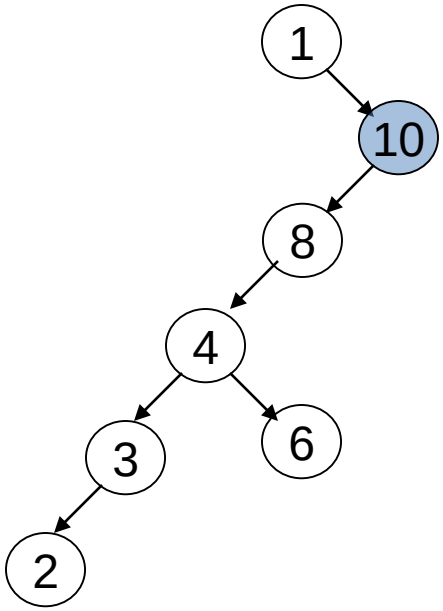
Binary Tree Insertion

1	10	8	4	6	3	2	5



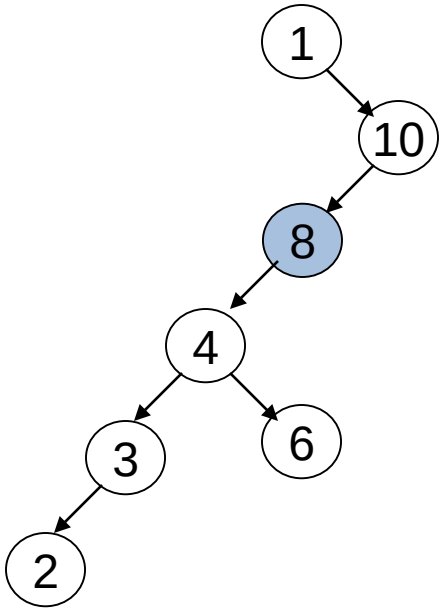
Binary Tree Insertion

1	10	8	4	6	3	2	5



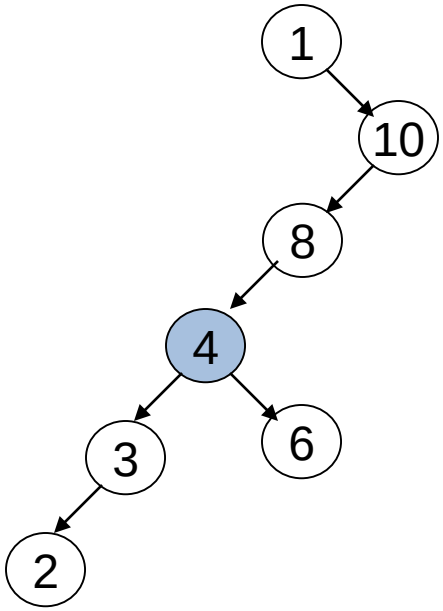
Binary Tree Insertion

1	10	8	4	6	3	2	5



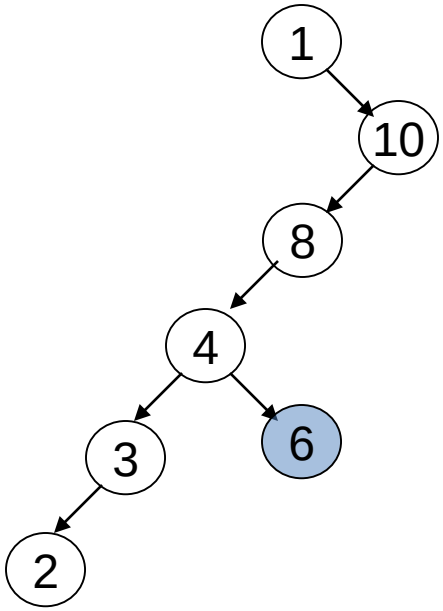
Binary Tree Insertion

1	10	8	4	6	3	2	5



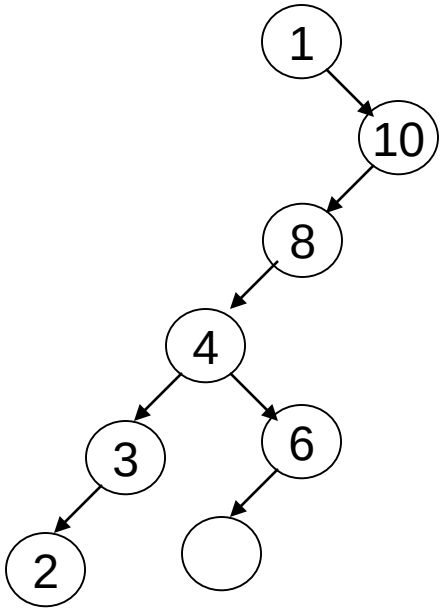
Binary Tree Insertion

1	10	8	4	6	3	2	5



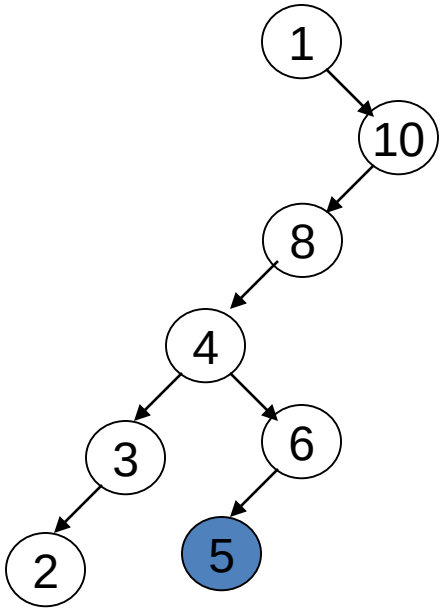
Binary Tree Insertion

1	10	8	4	6	3	2	5



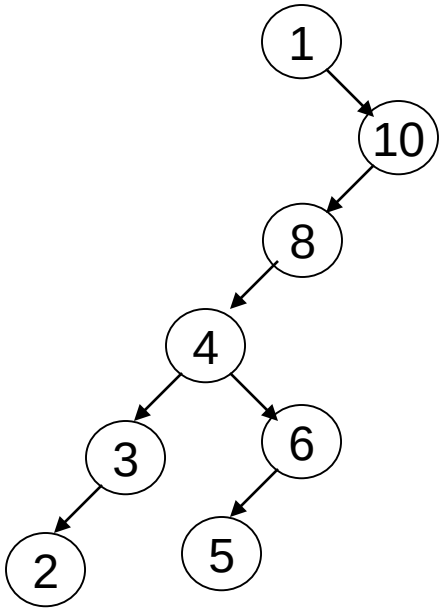
Binary Tree Insertion

1	10	8	4	6	3	2	5



Binary Tree Insertion

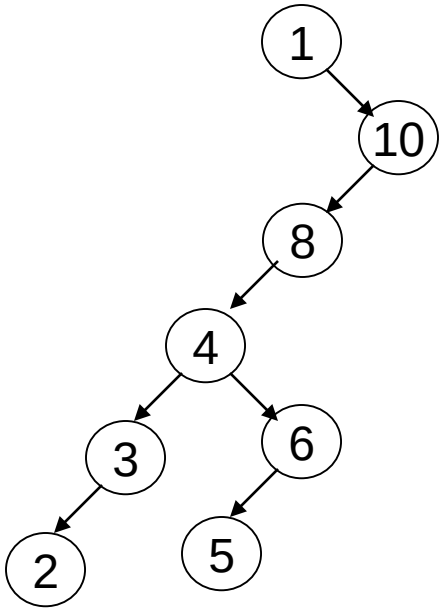
1	10	8	4	6	3	2	5



Binary Tree Deletion

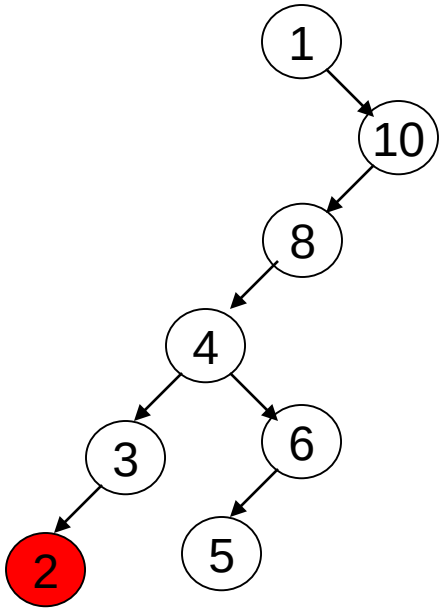
Binary Tree Deletion

1	10	8	4	6	3	2	5



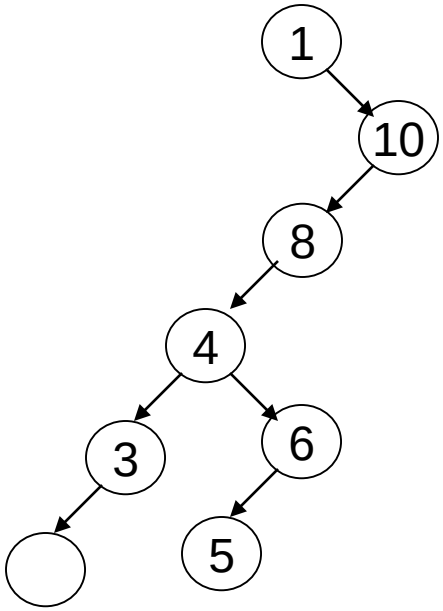
Binary Tree Deletion

1	10	8	4	6	3	2	5



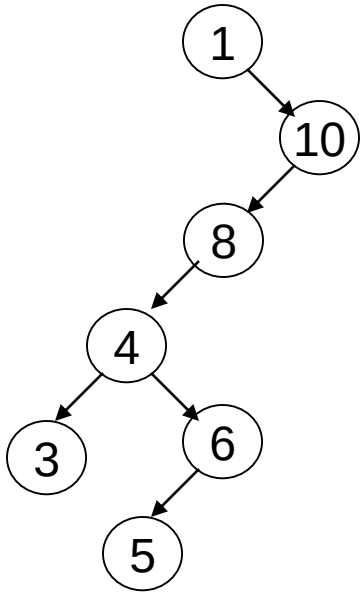
Binary Tree Deletion

1	10	8	4	6	3	2	5



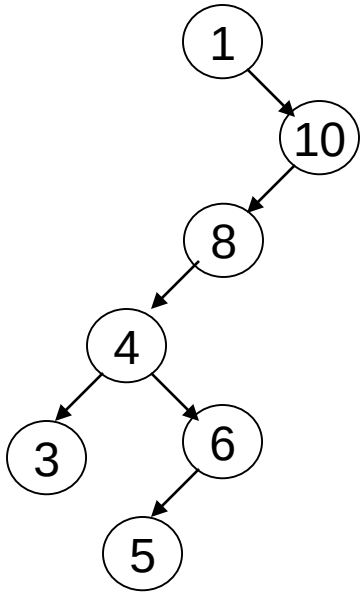
Binary Tree Deletion

1	10	8	4	6	3	2	5



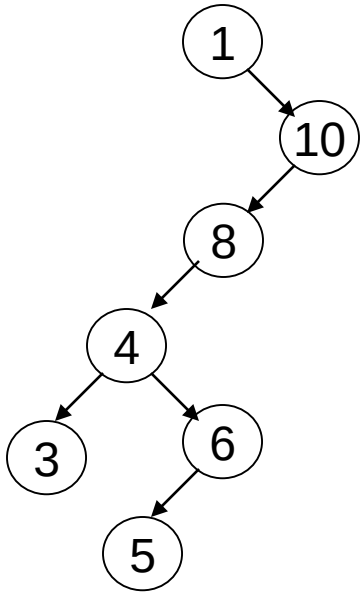
Binary Tree Deletion

1	10	8	4	6	3	5



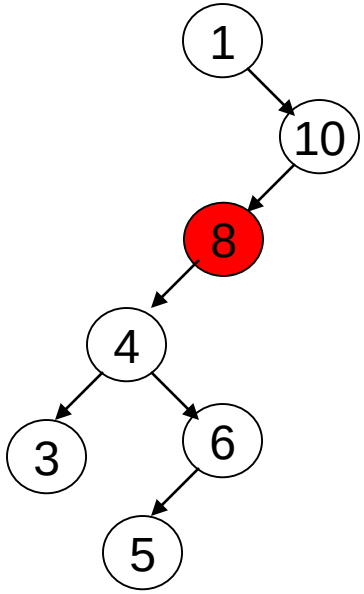
Binary Tree Deletion

1	10	8	4	6	3	5



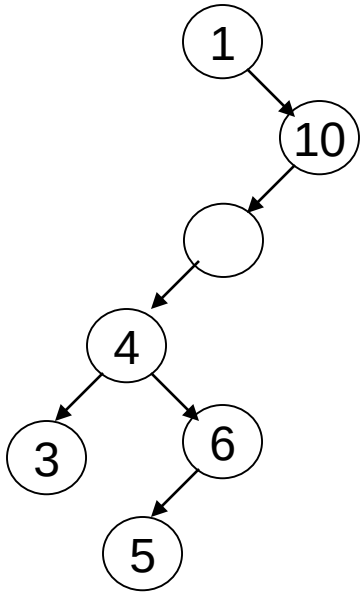
Binary Tree Deletion

1	10	8	4	6	3	5



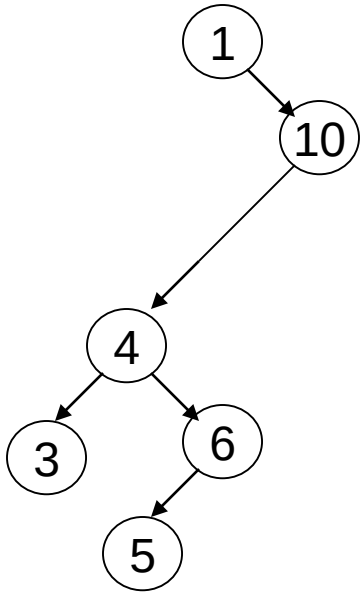
Binary Tree Deletion

1	10	8	4	6	3	5



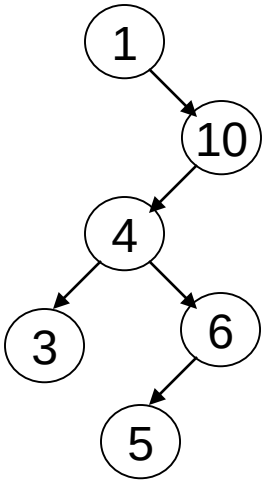
Binary Tree Deletion

1	10	8	4	6	3	5



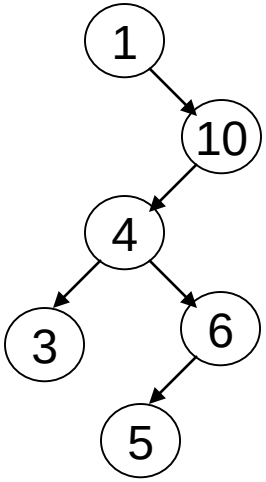
Binary Tree Deletion

1	10	8	4	6	3	5



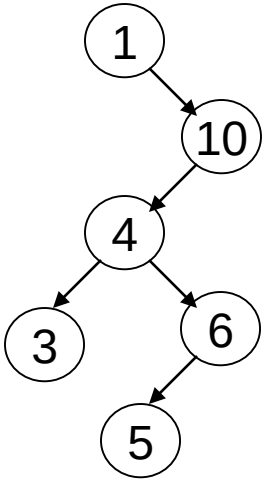
Binary Tree Deletion

1	10	4	6	3	5



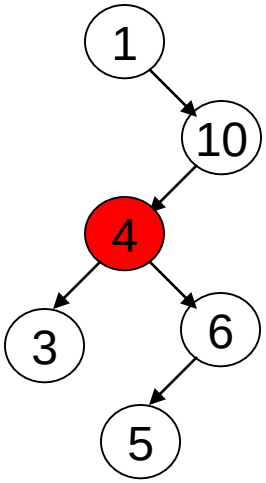
Binary Tree Deletion

1	10	4	6	3	5



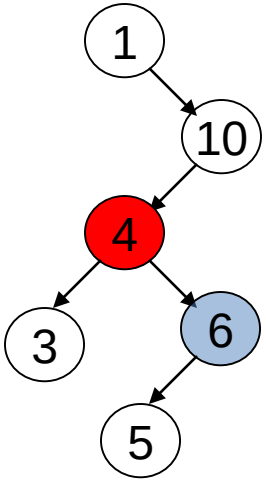
Binary Tree Deletion

1	10	4	6	3	5



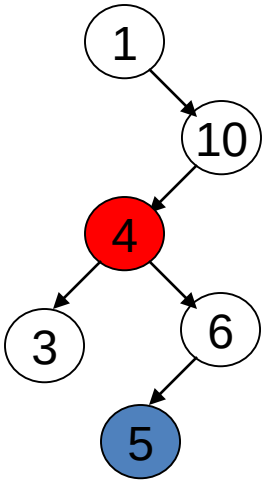
Binary Tree Deletion

1	10	4	6	3	5



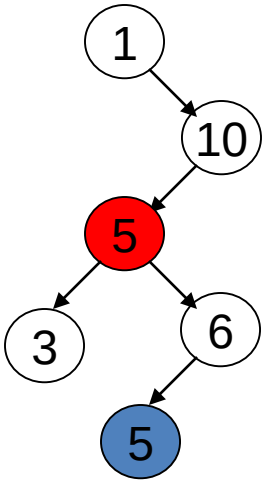
Binary Tree Deletion

1	10	4	6	3	5



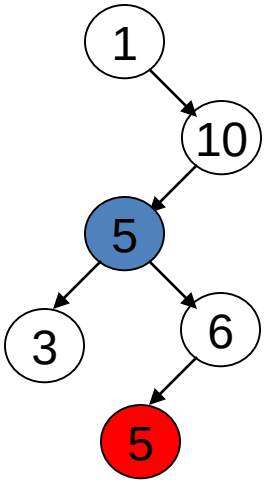
Binary Tree Deletion

1	10	4	6	3	5



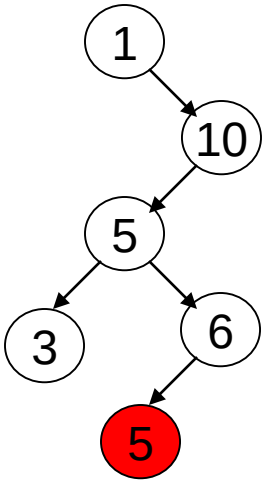
Binary Tree Deletion

1	10	4	6	3	5



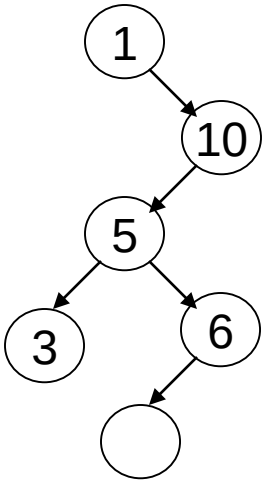
Binary Tree Deletion

1	10	4	6	3	5



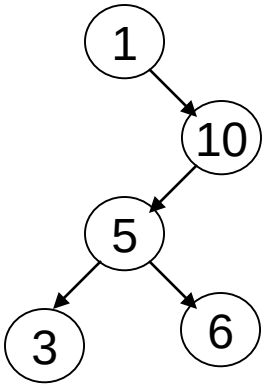
Binary Tree Deletion

1	10	4	6	3	5



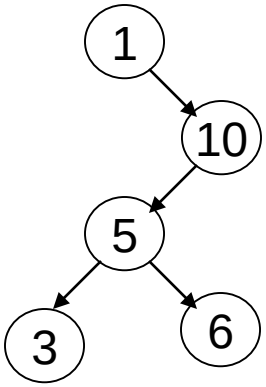
Binary Tree Deletion

1	10	4	6	3	5

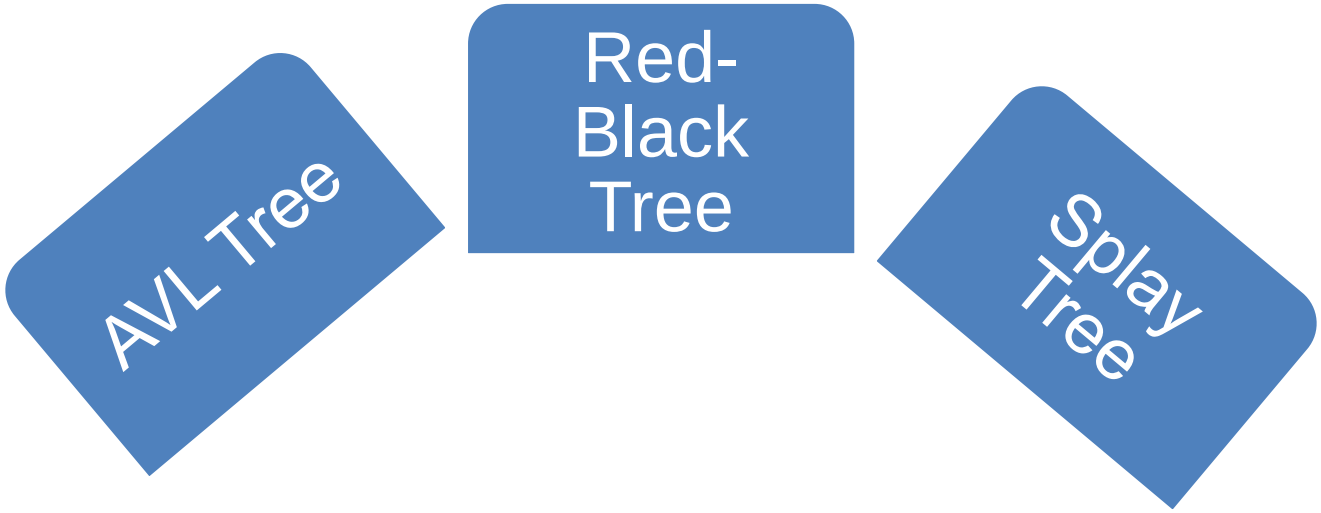


Binary Tree Deletion

1	10	6	3	5



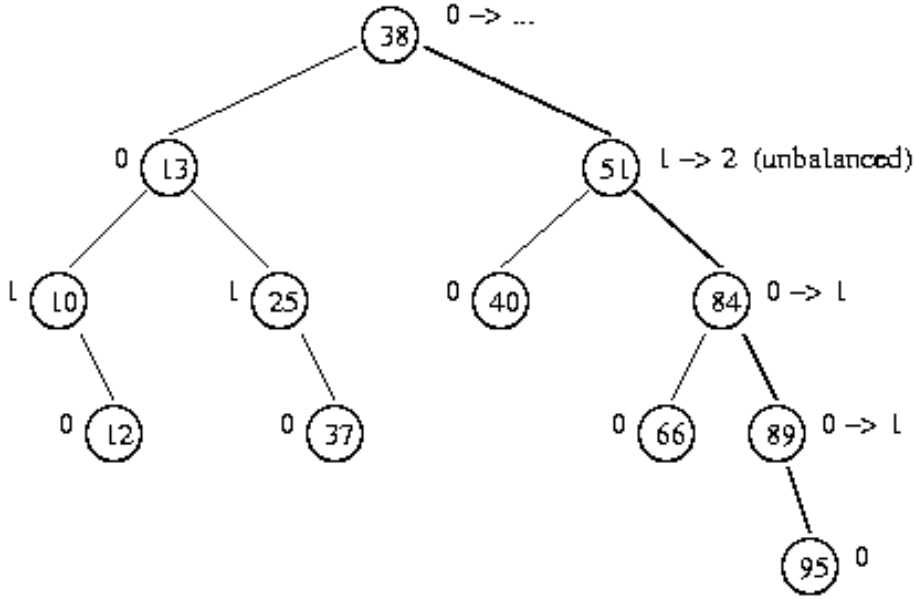
Types of BST



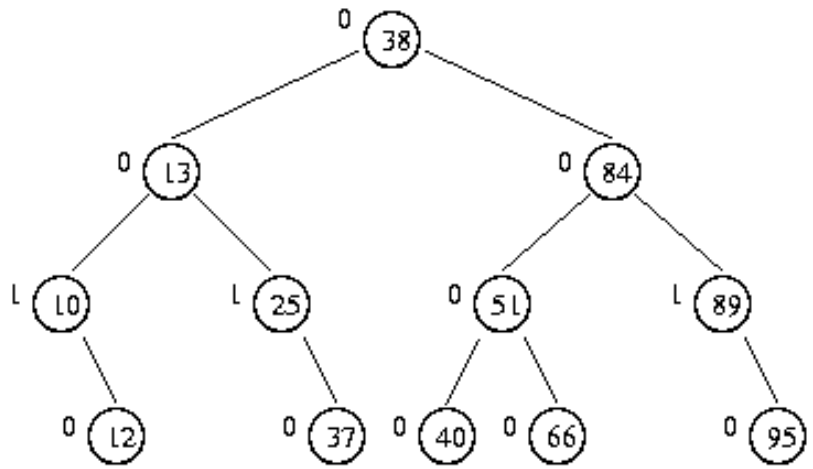
AVL Tree

AVL tree is a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees cannot be more than one for all nodes.

The tree became unballanced after inserting key 95.

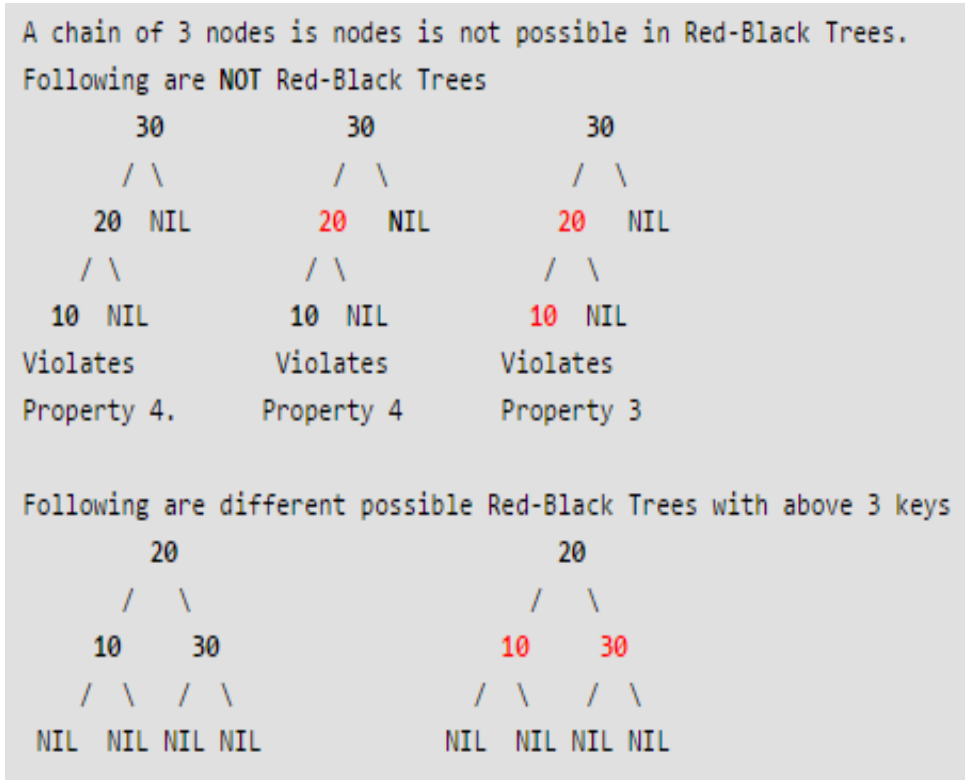


After the tree is rebalanced using rotation we have:



Red Black Tree

- Every node has a color either red or black.
- Root of tree is always black.
- There are no two adjacent red nodes (A red node cannot have a red parent or red child).
- Every path from root to a NULL node has same number of black nodes.



Splay Tree

Automatically moves frequently accessed elements nearer to the root for quick to access

Complexity in BST

Operation	Average	Worst Case	Best Case
Search	O(log n)	O(n)	O(1)
Insertion	O(log n)	O(n)	O(1)
Deletion	O(log n)	O(n)	O(1)

Applications of BST

- Used in many search applications where data is constantly entering/leaving, such as the map and set objects in many languages' libraries.
- Storing a set of names, and being able to lookup based on a prefix of the name. (Used in internet routers.)
- Storing a path in a graph, and being able to reverse any subsection of the path in O(log n) time. (Useful in travelling salesman problems).
- Finding square root of given number
- allows you to do range searches efficiently.

Thank you