Laboratory 5: Binary Trees

CSC205A Data structures and Algorithms Laboratory B. Tech. 2015

Vaishali R Kulkarni

Department of Computer Science and Engineering
Faculty of Engineering and Technology
M. S. Ramaiah University of Applied Sciences
Email: vaishali.cs.et@msruas.ac.in
Tel: +91-80-4906-5555 (2212) WWW: www.msruas.ac.in

Introduction and Purpose of Experiment

- Linear organization used on arrays, vectors, stacks and queues become inefficient in some applications. Then we choose the structures which provide non-linear organization.
- Binary tree is a non-linear data structure used in many applications. This experiment introduces binary search trees and its applications.

Aim and objectives

Aim:

- To design and develop binary search tree ADT

Objectives:

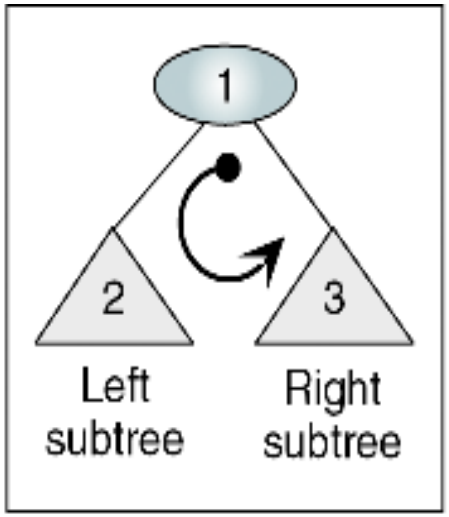
At the end of this lab, the student will be able to

- Design binary tree ADT
- Use binary tree ADT to illustrate the binary tree operations
- Use binary tree ADT and illustrate binary tree traversals

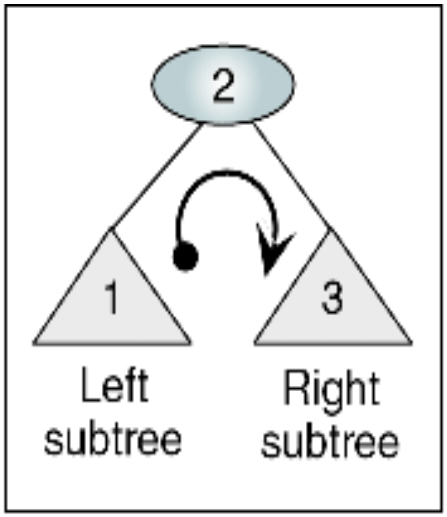
Basic BST Operations

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

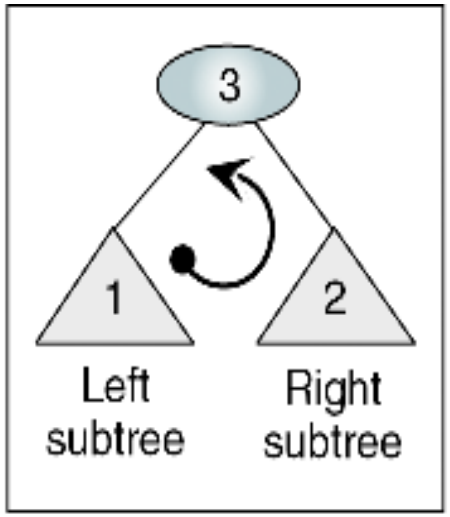
BST Traversal



(a) Preorder traversal



(b) Inorder traversal



(c) Postorder traversal

Inorder Traversal

- Inorder_tree(x)

```
1. If x != NULL
2.       then   Inorder_tree(left of x)
3.              print x
4.              Inorder_tree(right of x)
```

Preorder Traversal

- Preorder_tree(x)

```
1. If x != NULL
2.     then   print x
3.            Preorder_tree(left of x)
4.            Preorder_tree(right of x)
```

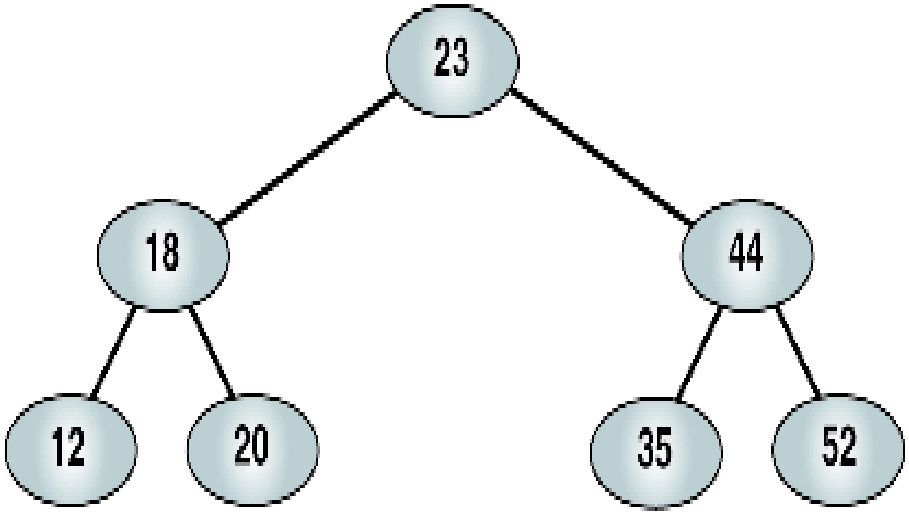
Postorder Traversal

- Postorder_tree(x)

```
1. If x != NULL
2.       then   Postorder_tree(left of x)
3.              Postorder_tree(right of x)
4.              print x
```

Preorder Traversal

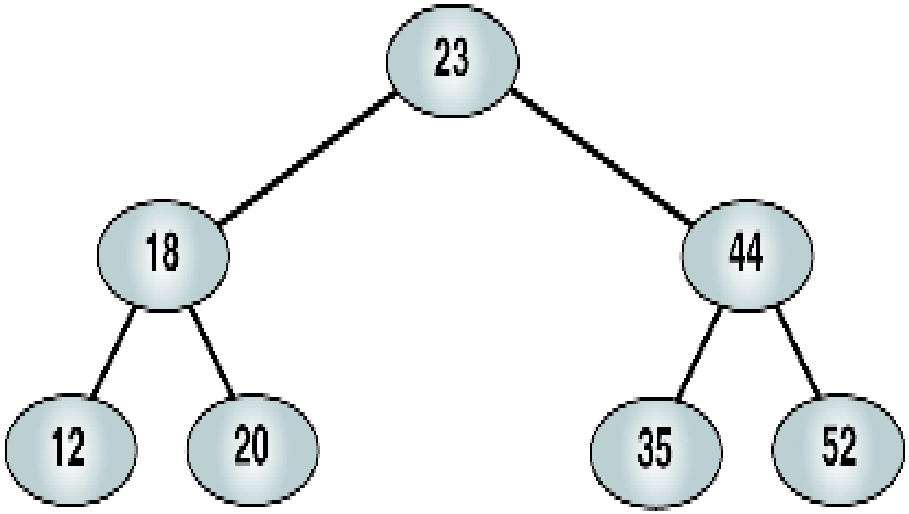
Root Left Right



23 18 12 20 44 35 52

Postorder Traversal

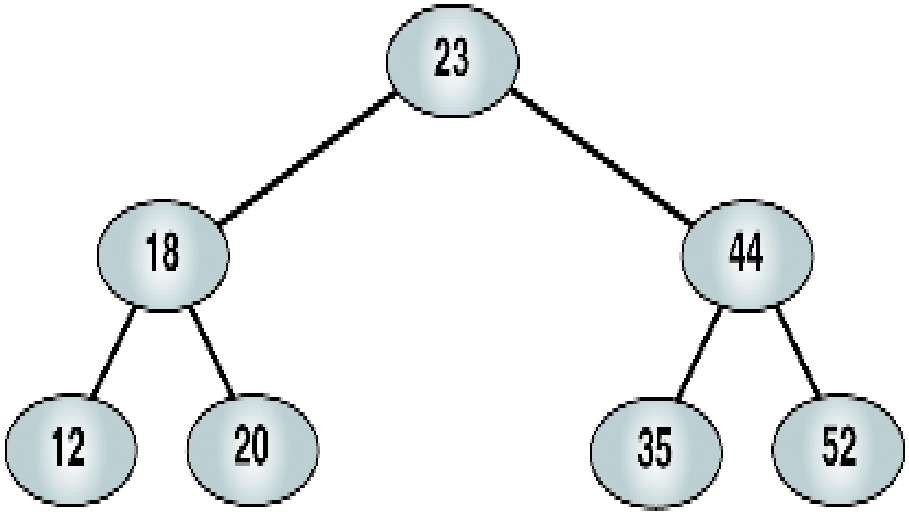
Left Right Root



12 20 18 35 52 44 23

Inorder Traversal

Left Root Right

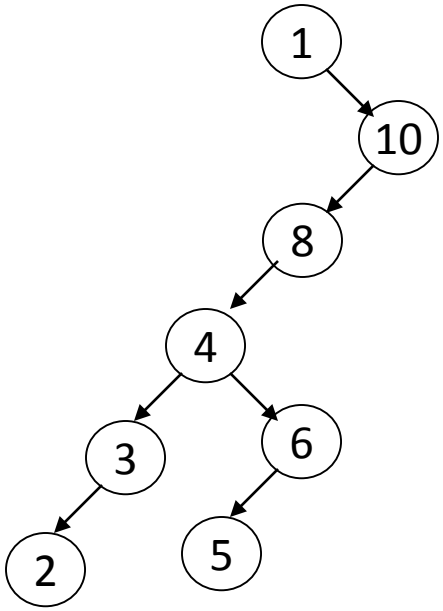


12 18 20 23 35 44 52

Produces a sequenced list

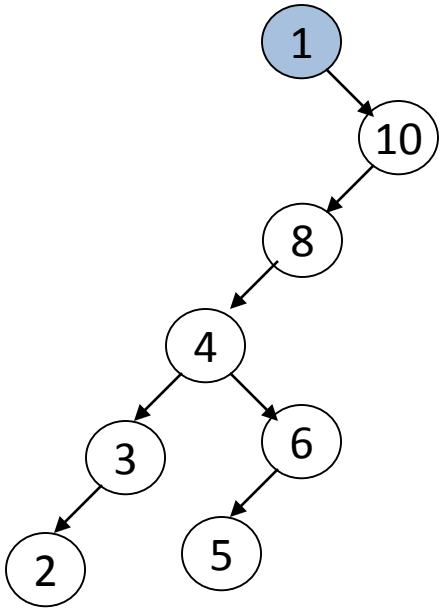
Inorder Traversal

1	10	8	4	6	3	2	5



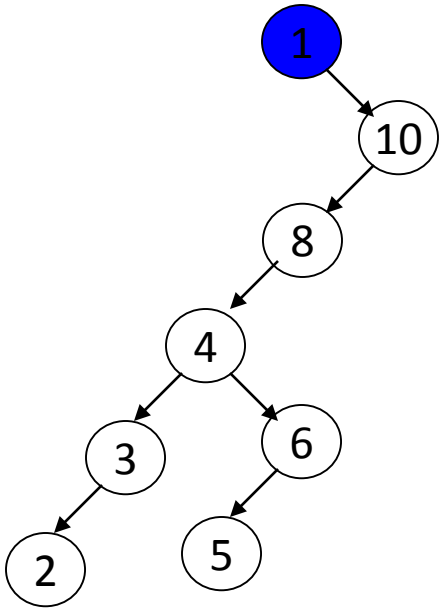
Inorder Traversal

1	10	8	4	6	3	2	5



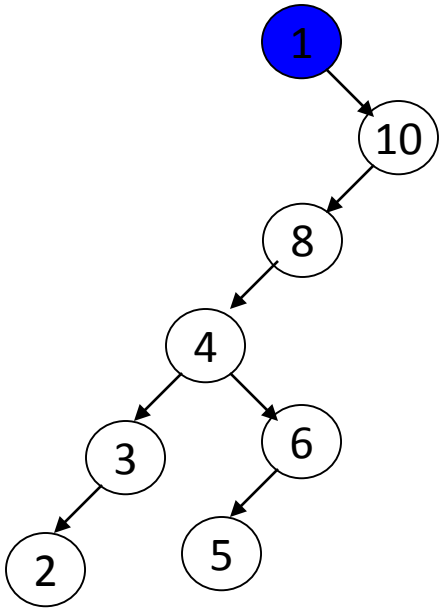
Inorder Traversal

1	10	8	4	6	3	2	5



Inorder Traversal

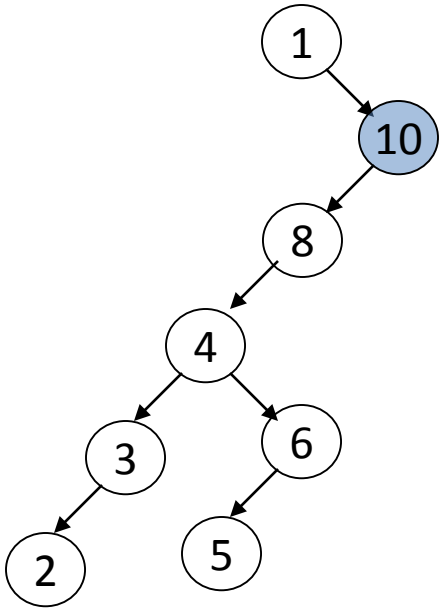
1	10	8	4	6	3	2	5



1

Inorder Traversal

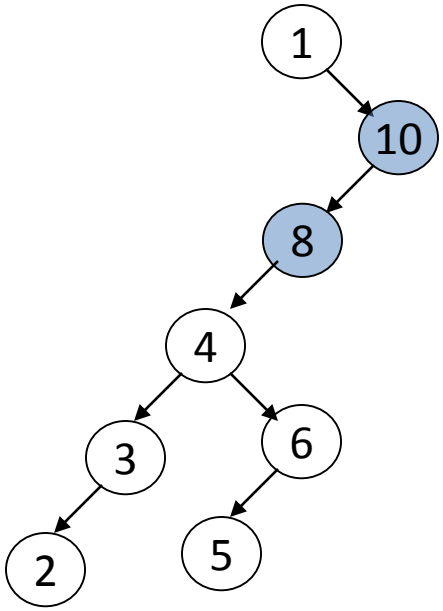
1	10	8	4	6	3	2	5



1

Inorder Traversal

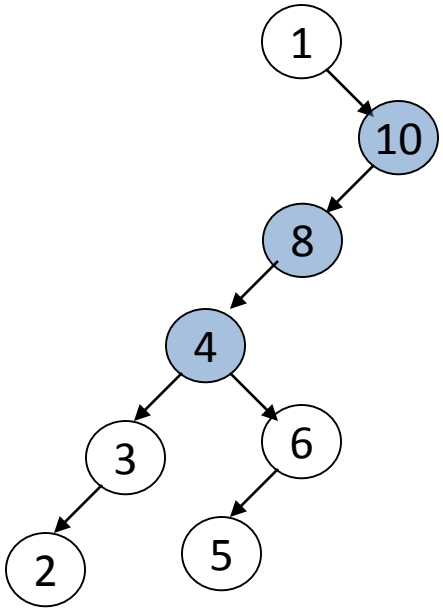
1	10	8	4	6	3	2	5



1

Inorder Traversal

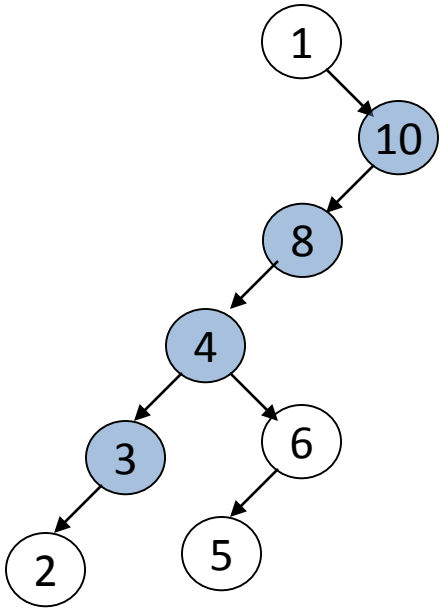
1	10	8	4	6	3	2	5



1

Inorder Traversal

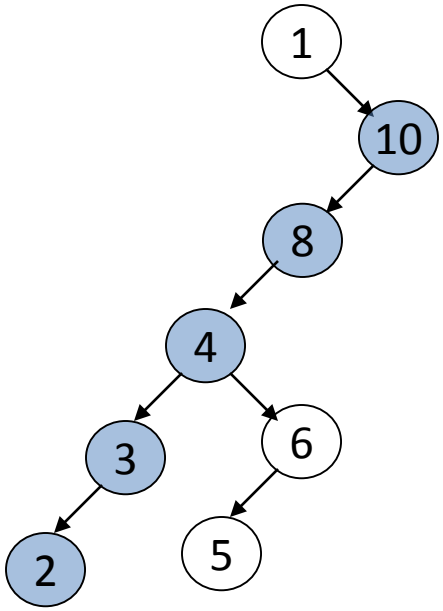
1	10	8	4	6	3	2	5



1

Inorder Traversal

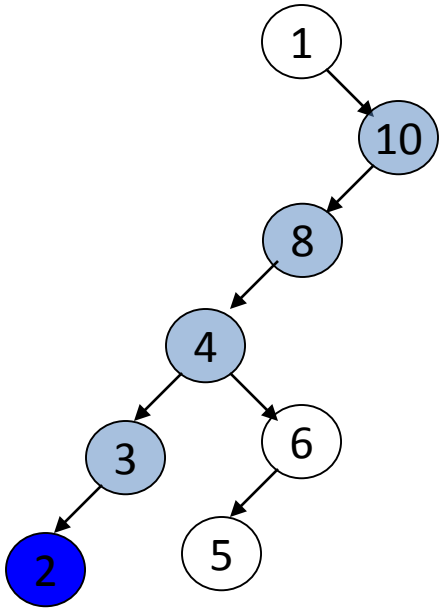
1	10	8	4	6	3	2	5



1

Inorder Traversal

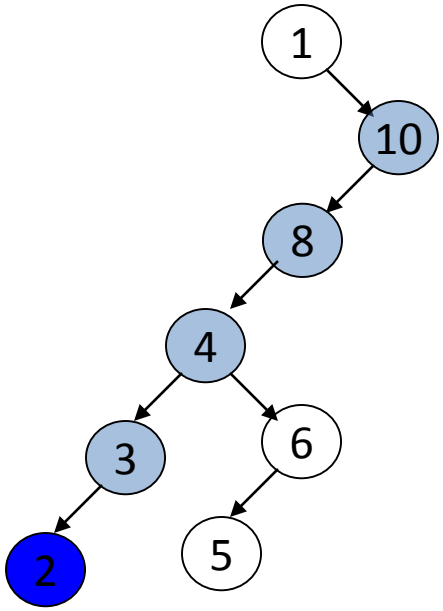
1	10	8	4	6	3	2	5



1

Inorder Traversal

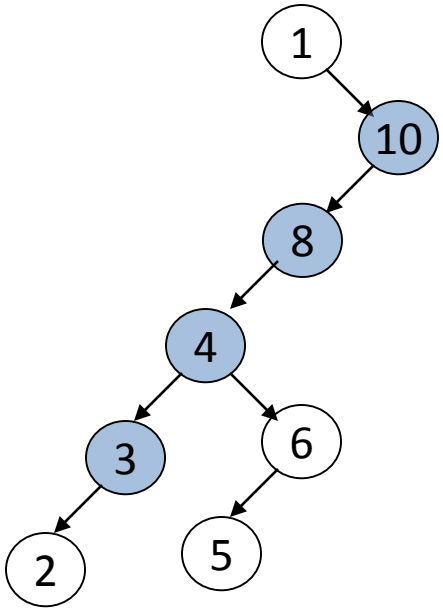
1	10	8	4	6	3	2	5



1	2

Inorder Traversal

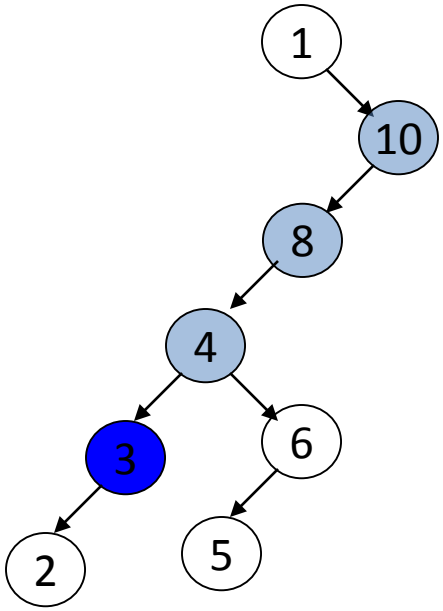
1	10	8	4	6	3	2	5



1	2

Inorder Traversal

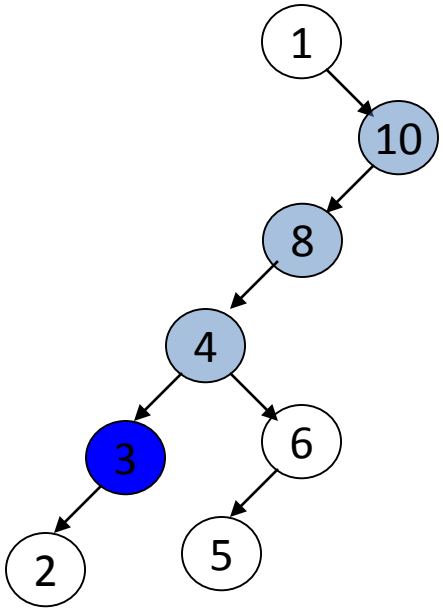
1	10	8	4	6	3	2	5



1	2

Inorder Traversal

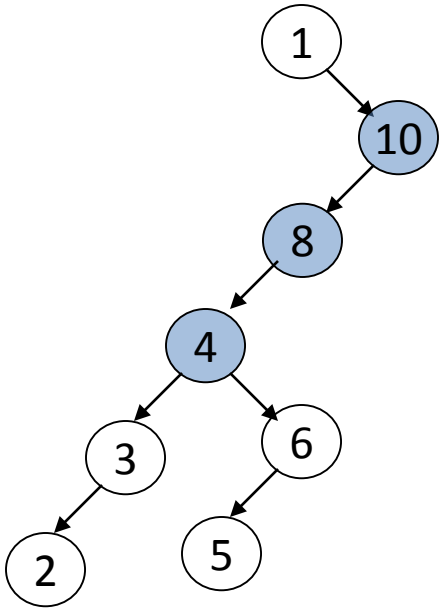
1	10	8	4	6	3	2	5



1	2	3

Inorder Traversal

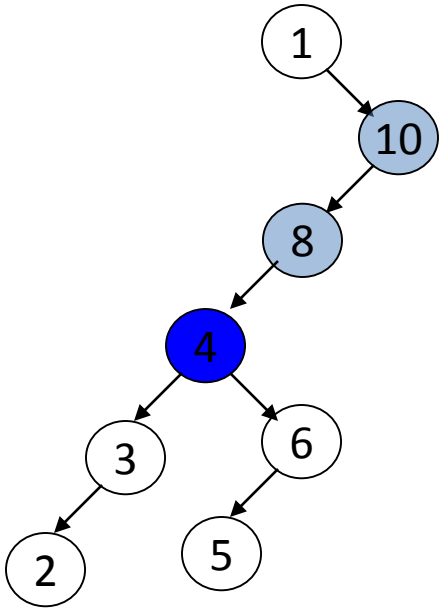
1	10	8	4	6	3	2	5



1	2	3

Inorder Traversal

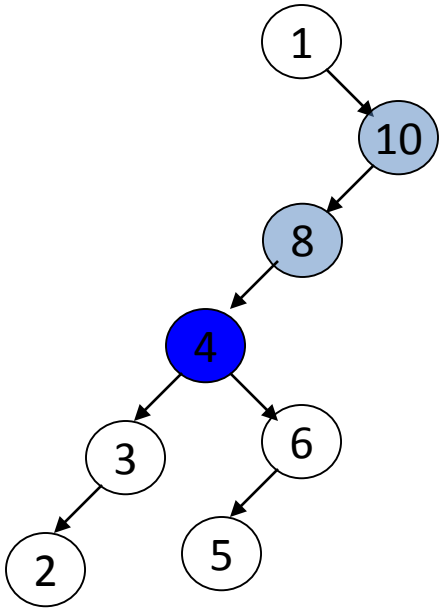
1	10	8	4	6	3	2	5



1	2	3

Inorder Traversal

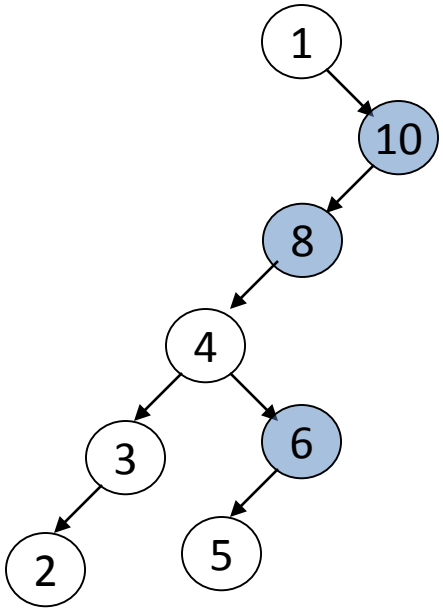
1	10	8	4	6	3	2	5



1	2	3	4

Inorder Traversal

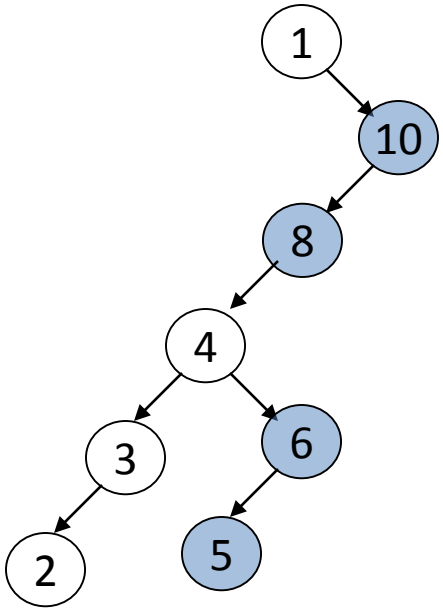
1	10	8	4	6	3	2	5



1	2	3	4

Inorder Traversal

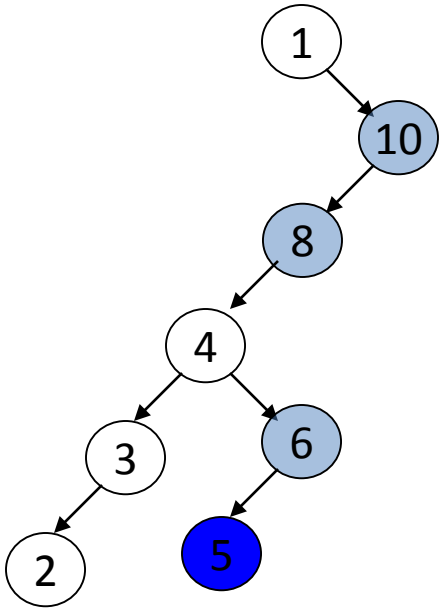
1	10	8	4	6	3	2	5



1	2	3	4

Inorder Traversal

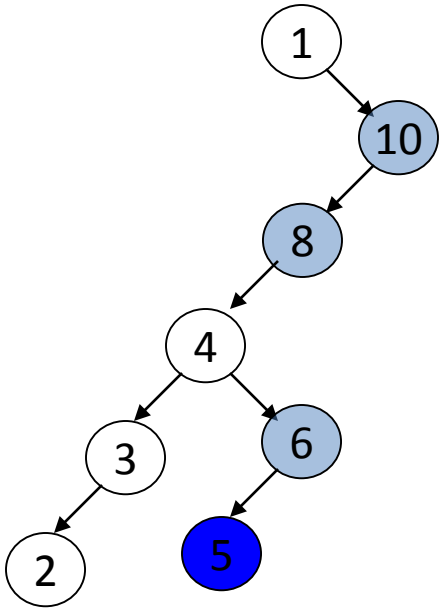
1	10	8	4	6	3	2	5



1	2	3	4

Inorder Traversal

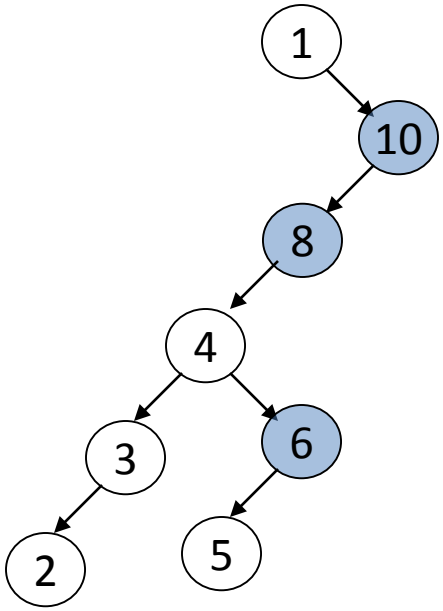
1	10	8	4	6	3	2	5



1	2	3	4	5

Inorder Traversal

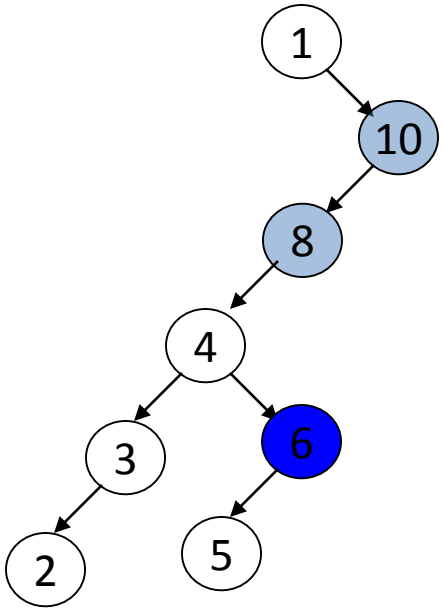
1	10	8	4	6	3	2	5



1	2	3	4	5

Inorder Traversal

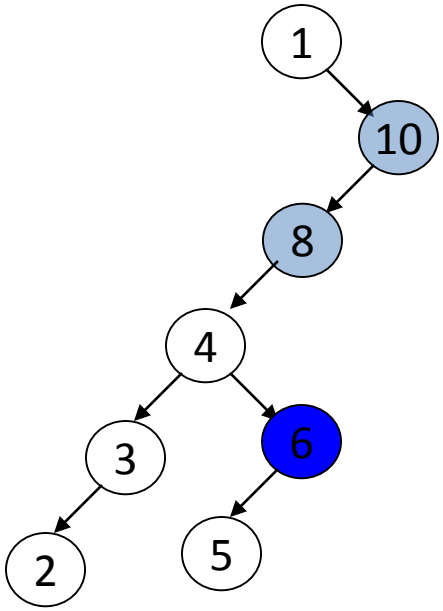
1	10	8	4	6	3	2	5



1	2	3	4	5

Inorder Traversal

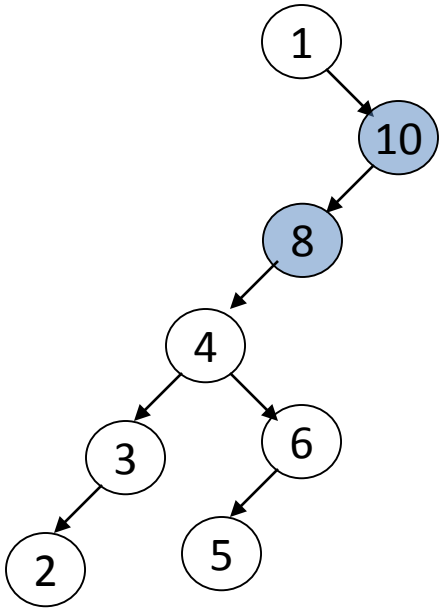
1	10	8	4	6	3	2	5



1	2	3	4	5	6

Inorder Traversal

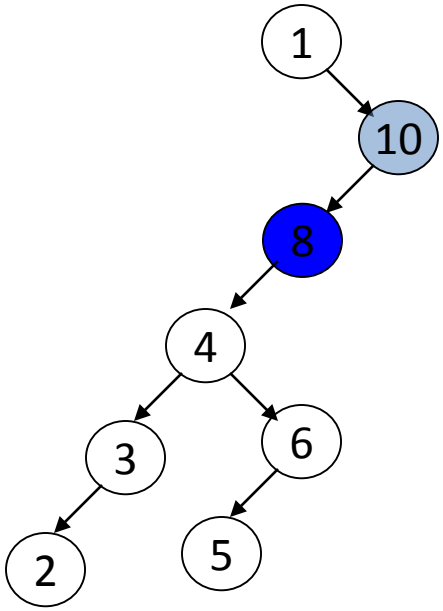
1	10	8	4	6	3	2	5



1	2	3	4	5	6

Inorder Traversal

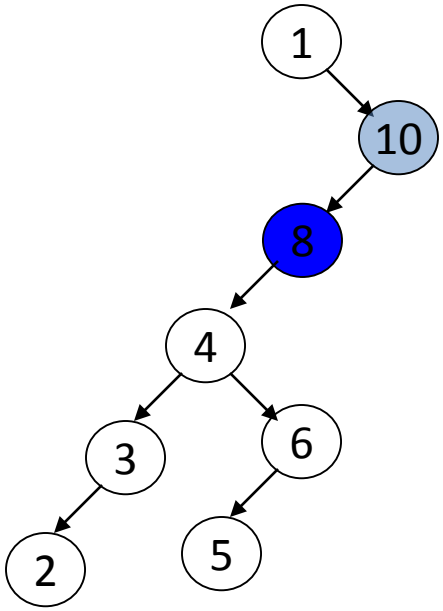
1	10	8	4	6	3	2	5



1	2	3	4	5	6

Inorder Traversal

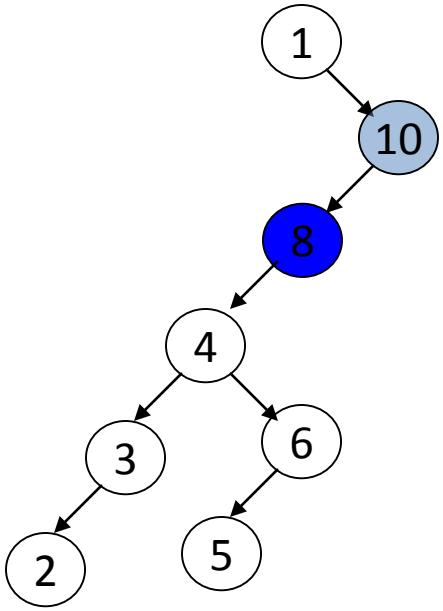
1	10	8	4	6	3	2	5



1	2	3	4	5	6

Inorder Traversal

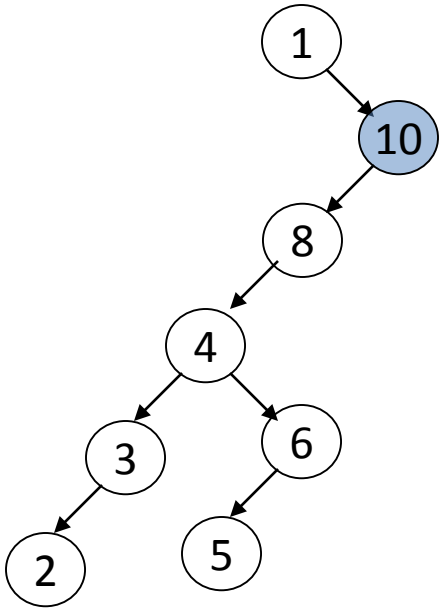
1	10	8	4	6	3	2	5



1	2	3	4	5	6	8

Inorder Traversal

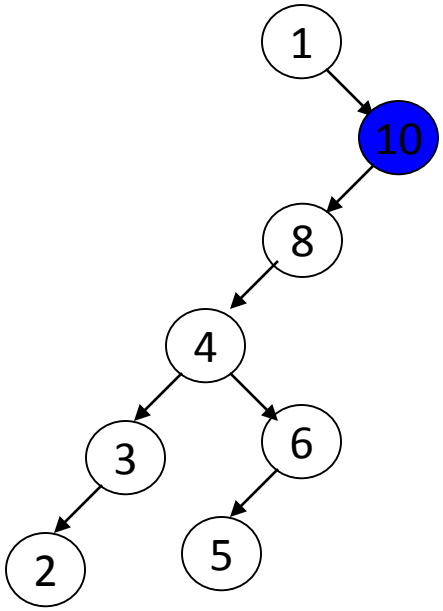
1	10	8	4	6	3	2	5



1	2	3	4	5	6	8

Inorder Traversal

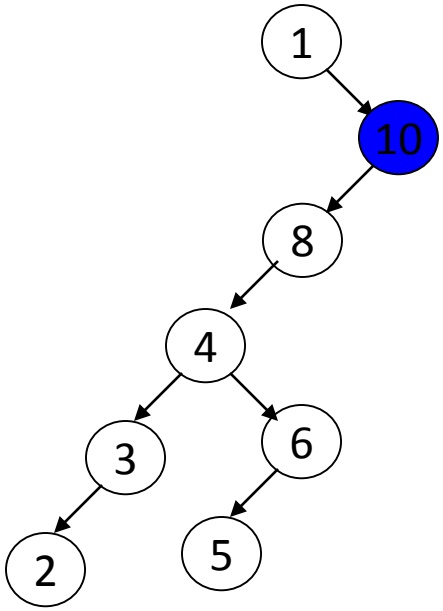
1	10	8	4	6	3	2	5



1	2	3	4	5	6	8

Inorder Traversal

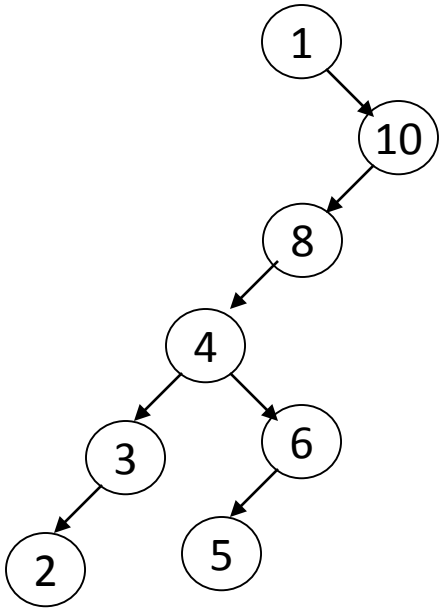
1	10	8	4	6	3	2	5



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

1

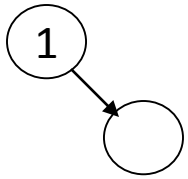
Binary Tree Insertion

1	10	8	4	6	3	2	5

1

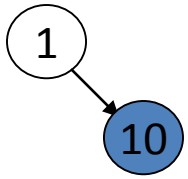
Binary Tree Insertion

1	10	8	4	6	3	2	5



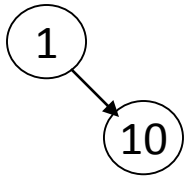
Binary Tree Insertion

1	10	8	4	6	3	2	5



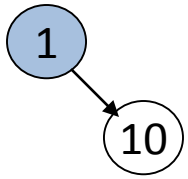
Binary Tree Insertion

1	10	8	4	6	3	2	5



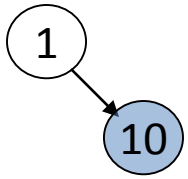
Binary Tree Insertion

1	10	8	4	6	3	2	5



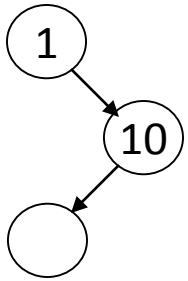
Binary Tree Insertion

1	10	8	4	6	3	2	5



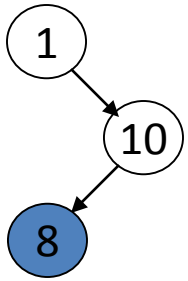
Binary Tree Insertion

1	10	8	4	6	3	2	5



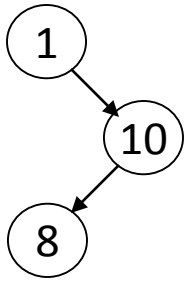
Binary Tree Insertion

1	10	8	4	6	3	2	5



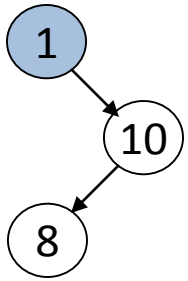
Binary Tree Insertion

1	10	8	4	6	3	2	5



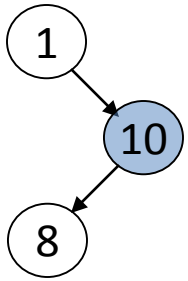
Binary Tree Insertion

1	10	8	4	6	3	2	5



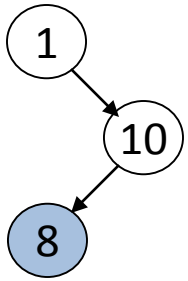
Binary Tree Insertion

1	10	8	4	6	3	2	5



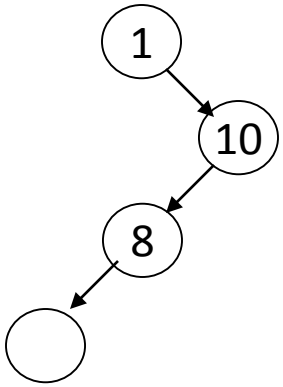
Binary Tree Insertion

1	10	8	4	6	3	2	5



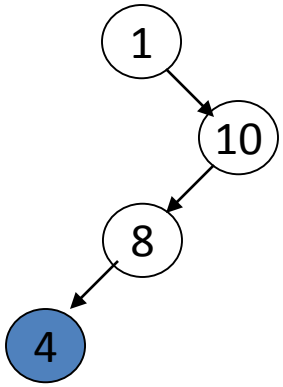
Binary Tree Insertion

1	10	8	4	6	3	2	5



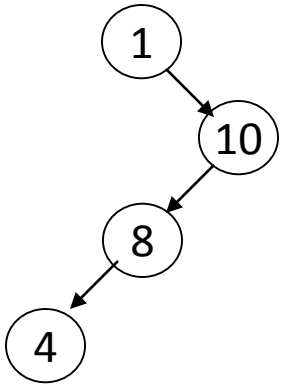
Binary Tree Insertion

1	10	8	4	6	3	2	5



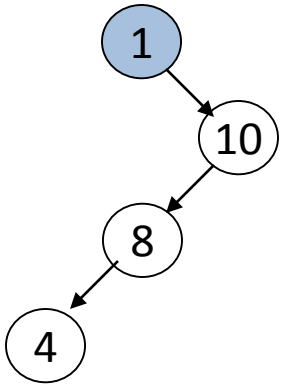
Binary Tree Insertion

1	10	8	4	6	3	2	5



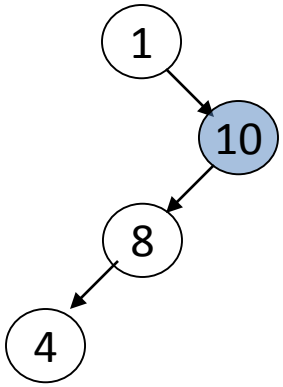
Binary Tree Insertion

1	10	8	4	6	3	2	5



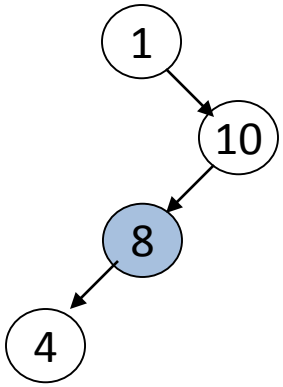
Binary Tree Insertion

1	10	8	4	6	3	2	5



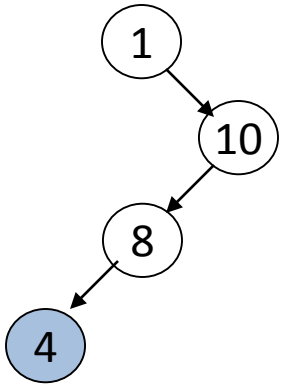
Binary Tree Insertion

1	10	8	4	6	3	2	5



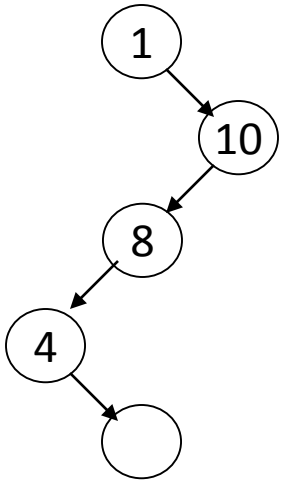
Binary Tree Insertion

1	10	8	4	6	3	2	5



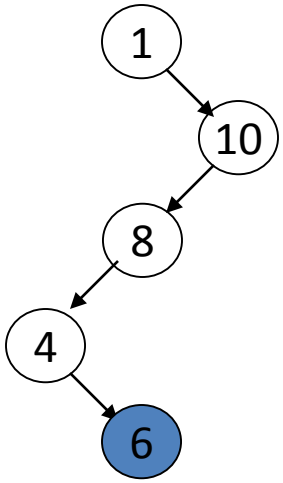
Binary Tree Insertion

1	10	8	4	6	3	2	5



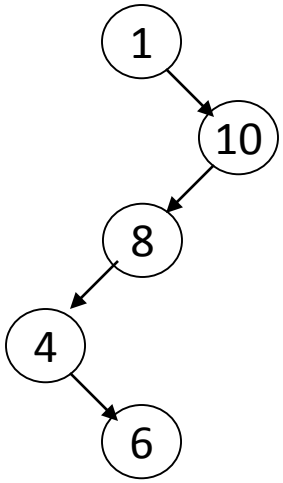
Binary Tree Insertion

1	10	8	4	6	3	2	5



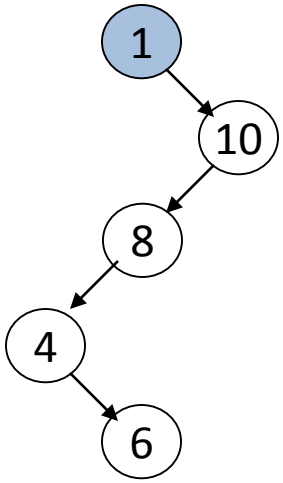
Binary Tree Insertion

1	10	8	4	6	3	2	5



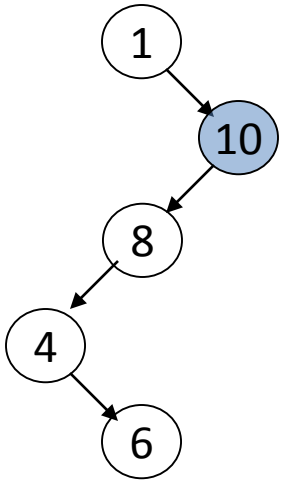
Binary Tree Insertion

1	10	8	4	6	3	2	5



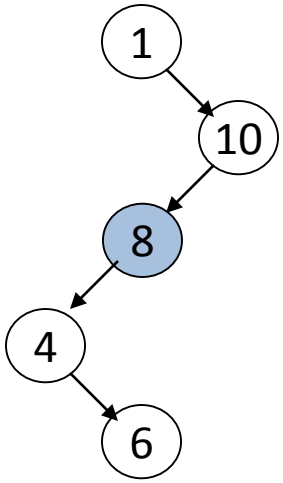
Binary Tree Insertion

1	10	8	4	6	3	2	5



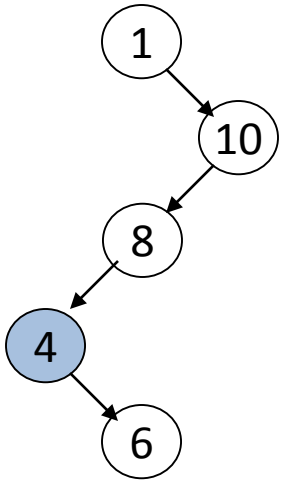
Binary Tree Insertion

1	10	8	4	6	3	2	5



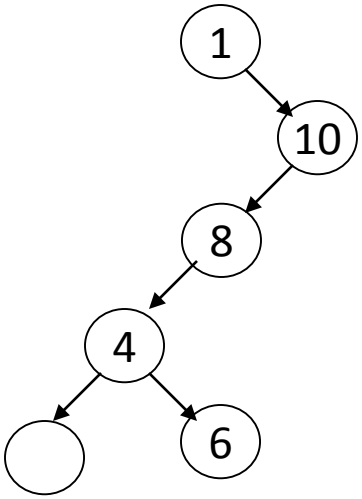
Binary Tree Insertion

1	10	8	4	6	3	2	5



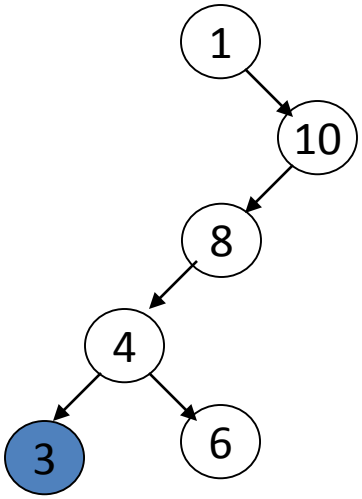
Binary Tree Insertion

1	10	8	4	6	3	2	5



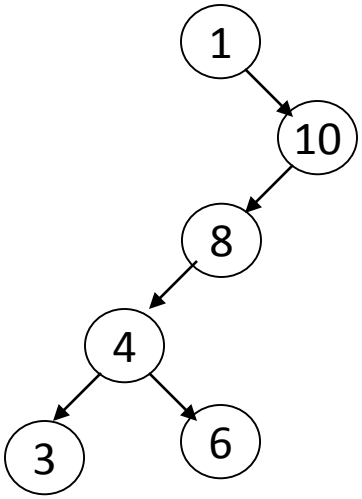
Binary Tree Insertion

1	10	8	4	6	3	2	5



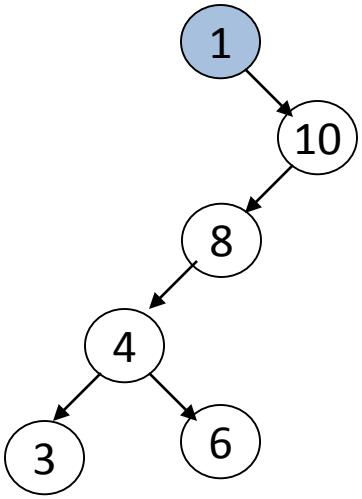
Binary Tree Insertion

1	10	8	4	6	3	2	5



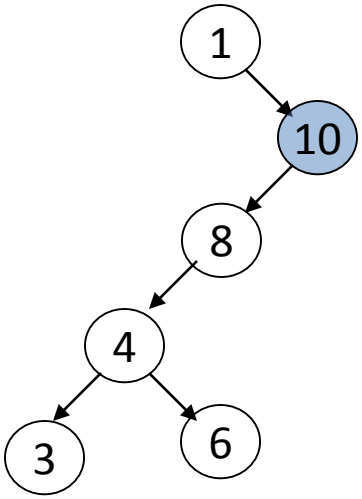
Binary Tree Insertion

1	10	8	4	6	3	2	5



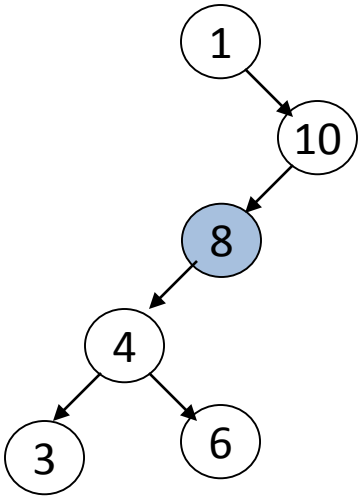
Binary Tree Insertion

1	10	8	4	6	3	2	5



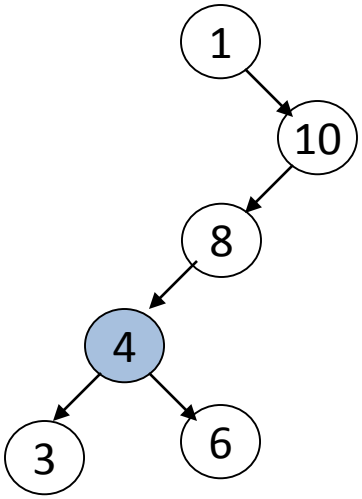
Binary Tree Insertion

1	10	8	4	6	3	2	5



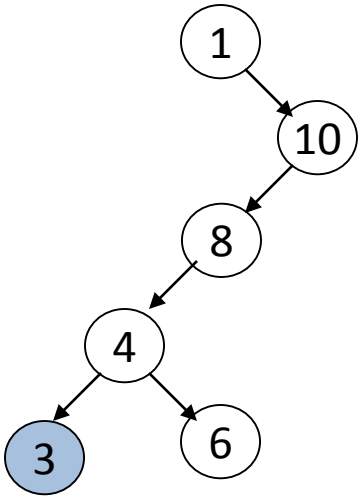
Binary Tree Insertion

1	10	8	4	6	3	2	5



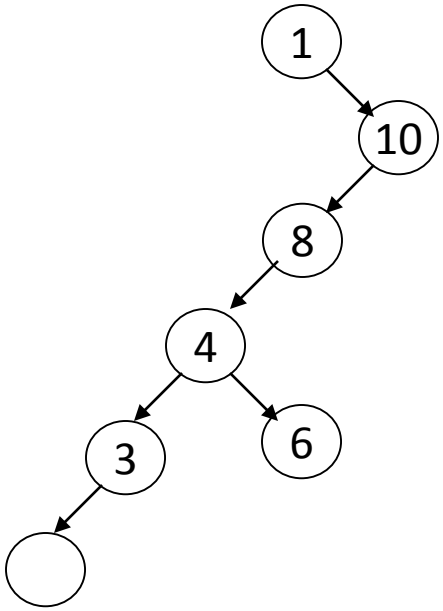
Binary Tree Insertion

1	10	8	4	6	3	2	5



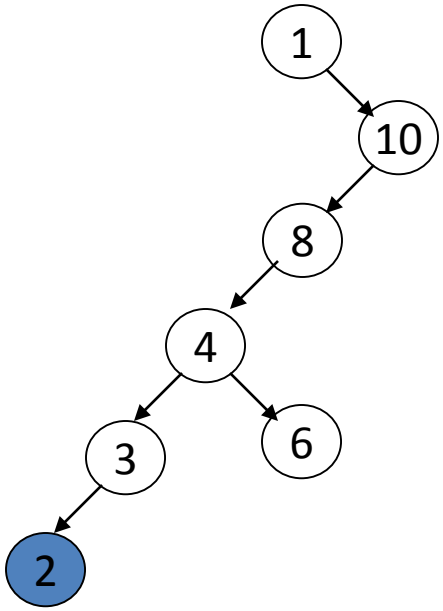
Binary Tree Insertion

1	10	8	4	6	3	2	5



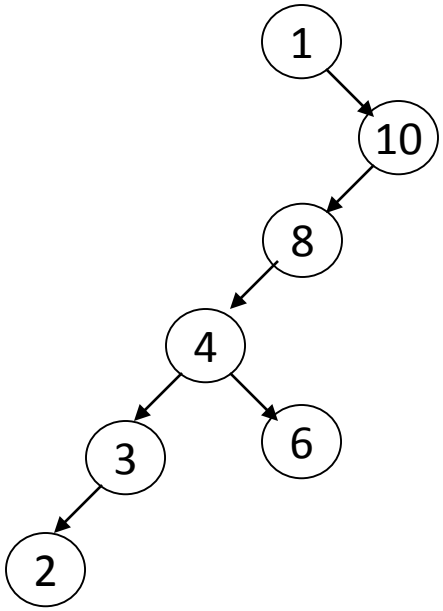
Binary Tree Insertion

1	10	8	4	6	3	2	5



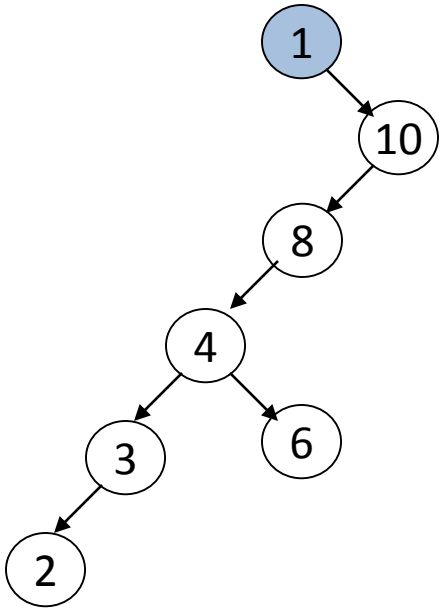
Binary Tree Insertion

1	10	8	4	6	3	2	5



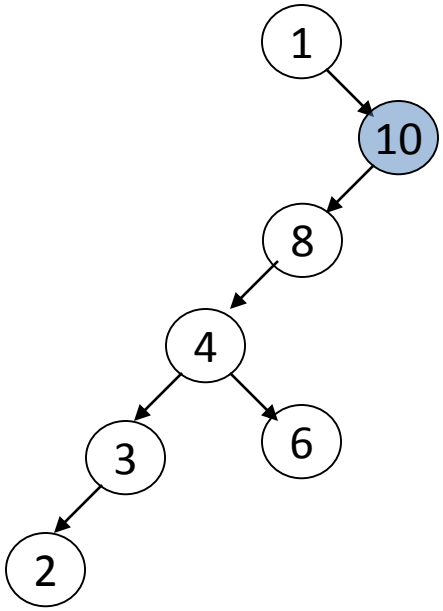
Binary Tree Insertion

1	10	8	4	6	3	2	5



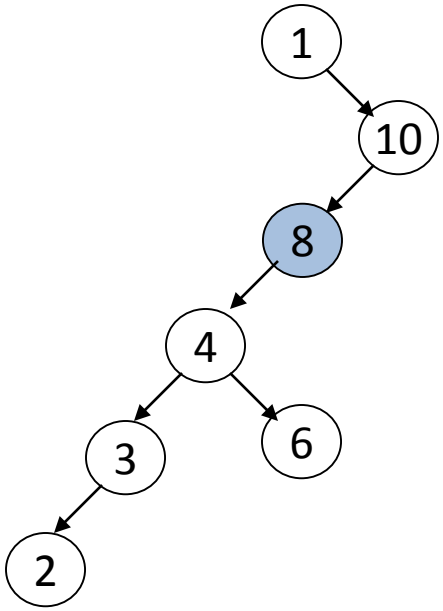
Binary Tree Insertion

1	10	8	4	6	3	2	5



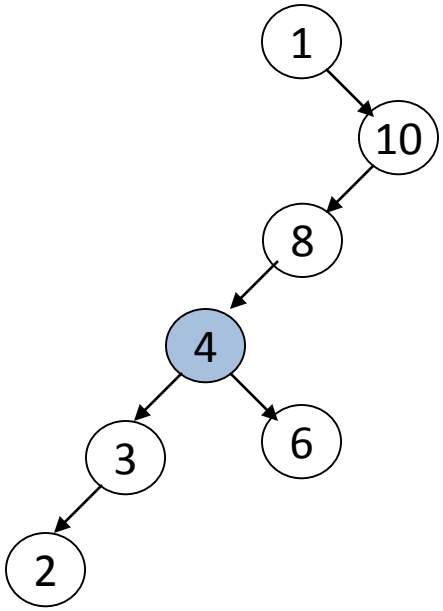
Binary Tree Insertion

1	10	8	4	6	3	2	5



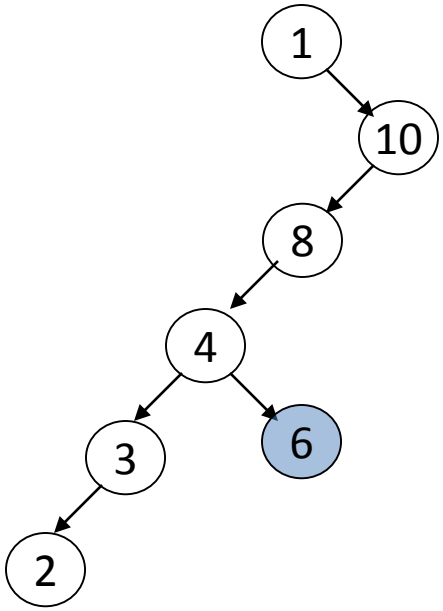
Binary Tree Insertion

1	10	8	4	6	3	2	5



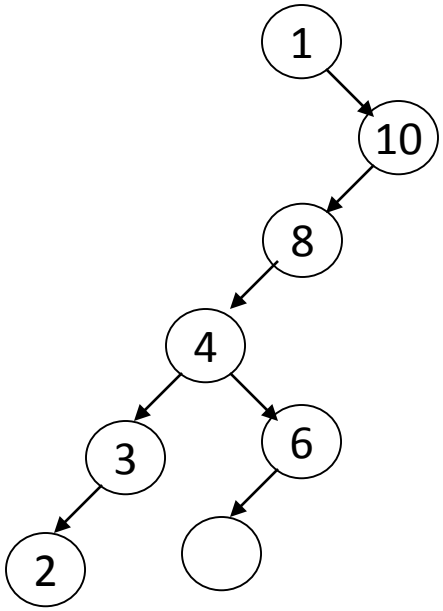
Binary Tree Insertion

1	10	8	4	6	3	2	5



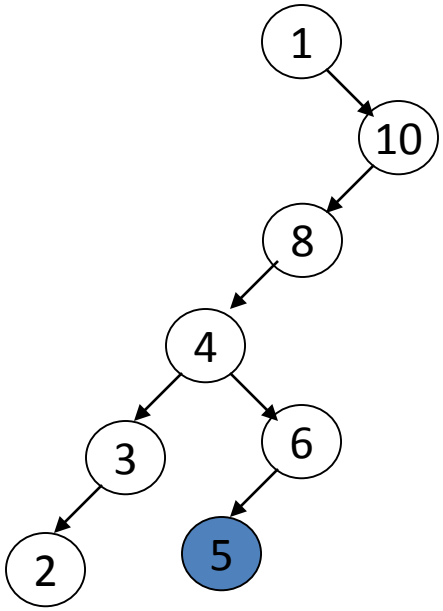
Binary Tree Insertion

1	10	8	4	6	3	2	5



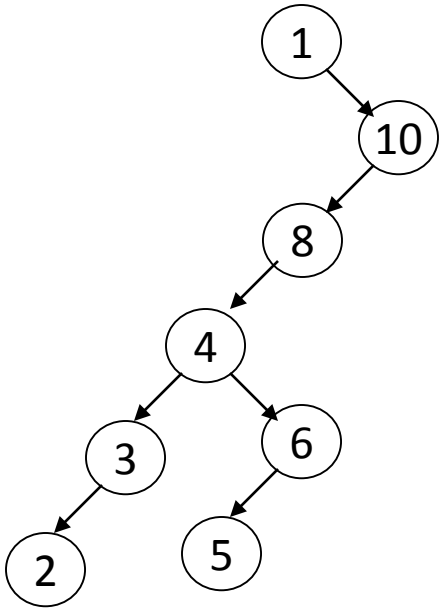
Binary Tree Insertion

1	10	8	4	6	3	2	5



Binary Tree Insertion

1	10	8	4	6	3	2	5



Insertion in BST

- Check whether root node is present or not(tree available or not). If root is NULL, create root node.

- If the element to be inserted is less than the element present in the root node, traverse the left sub-tree recursively until we reach T->left/ T->right is NULL and place the new node at T->left(key in new node < key in T)/T->right (key in new node > key in T).

- If the element to be inserted is greater than the element present in root node, traverse the right sub-tree recursively until we reach T->left/T->right is NULL and place the new node at T->left/T->right.

Algorithm for BST Insert

```
TreeNode insert(TreeNode T, int data) {
        if T is NULL {
                T = (TreeNode *)malloc(sizeof (Struct TreeNode));
                (Allocate Memory of new node and load the data into it)
                T->data = data;
                T->left   = NULL;
                T->right = NULL;
}  else if T is less than T->left {
                T->left = insert(data, T->left);
                (Then node needs to be inserted in left sub-tree.So,
                 recursively traverse left sub-tree to find the place
                 where the new node needs to be inserted)
} else if T is greater than T->right {
                T->right = insert(data, T->right);
                (Then node needs to be inserted in right sub-tree
                 So, recursively traverse right sub-tree to find the
                 place where the new node needs to be inserted.)
}
return T;  }
```

Binary Tree Deletion

BST Delete Operation

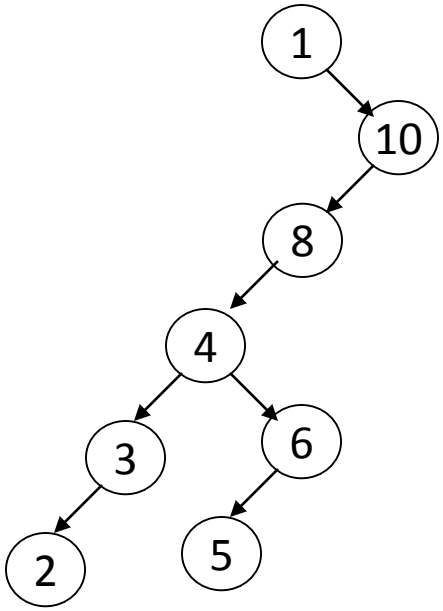
Case 1. Node to be deleted is leaf: Simply remove from the tree

Case 2. Node to be deleted has only one child: Copy the child to the node and delete the child

Case 3. Node to be deleted has two children: Find inorder successor of the node. Copy contents of the inorder successor to the node and delete the inorder successor. Note that inorder predecessor can also be used.

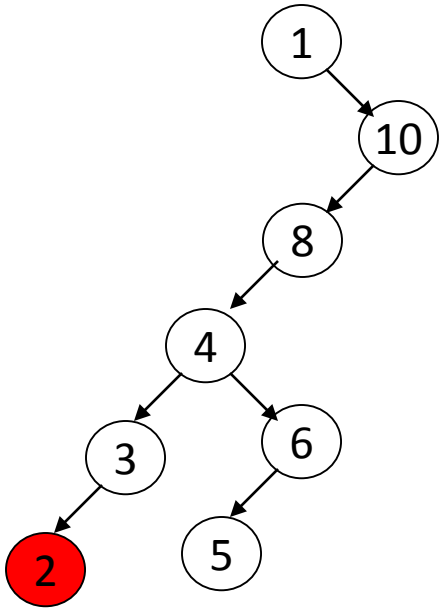
Binary Tree Deletion

1	10	8	4	6	3	2	5



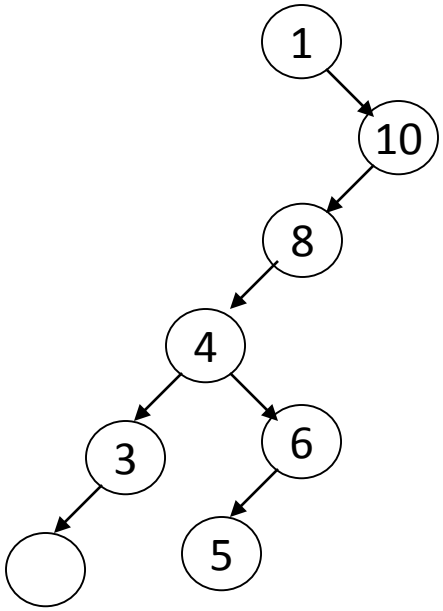
Binary Tree Deletion

1	10	8	4	6	3	2	5



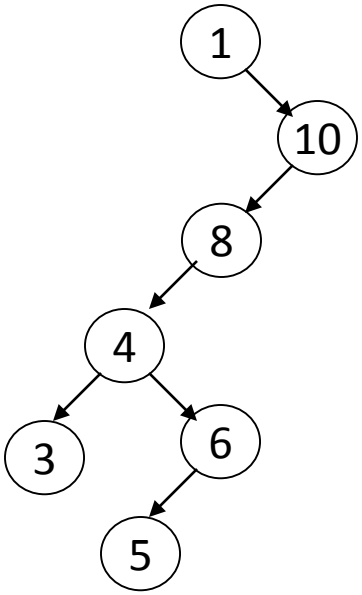
Binary Tree Deletion

1	10	8	4	6	3	2	5



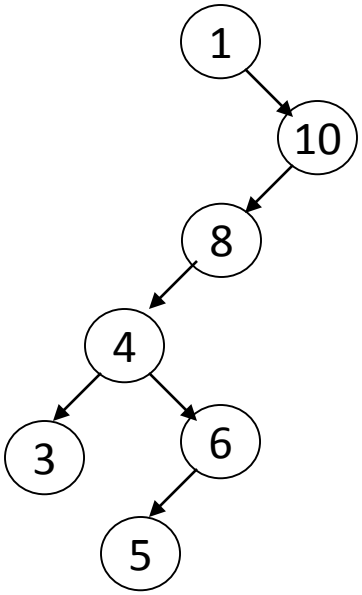
Binary Tree Deletion

1	10	8	4	6	3	2	5



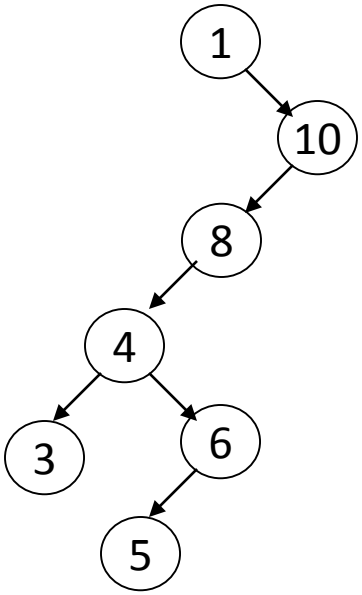
Binary Tree Deletion

1	10	8	4	6	3	5



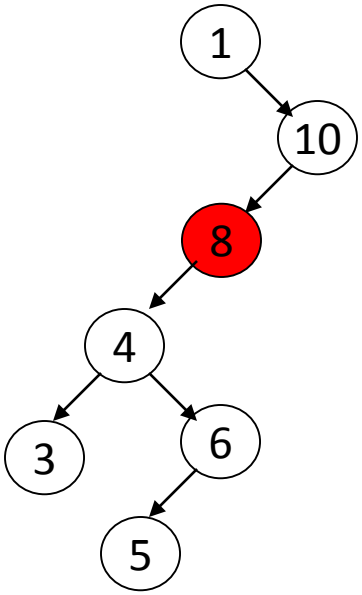
Binary Tree Deletion

1	10	8	4	6	3	5



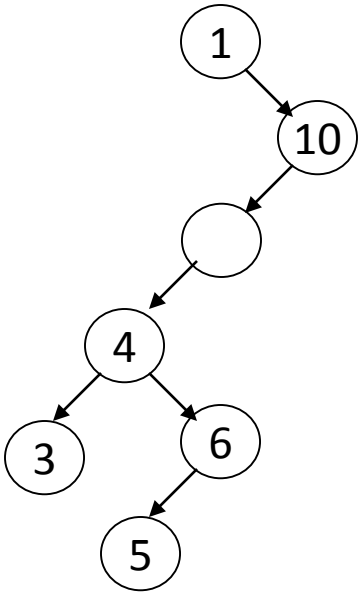
Binary Tree Deletion

1	10	8	4	6	3	5



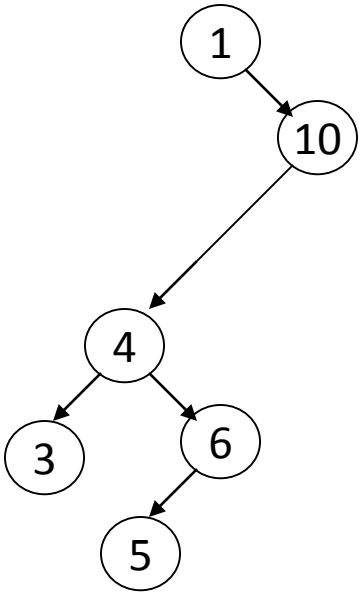
Binary Tree Deletion

1	10	8	4	6	3	5



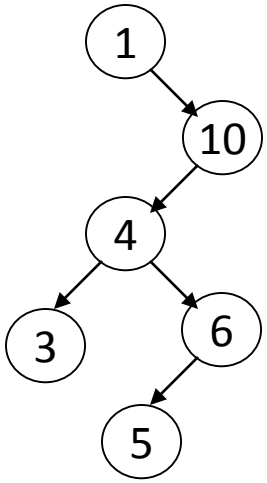
Binary Tree Deletion

1	10	8	4	6	3	5



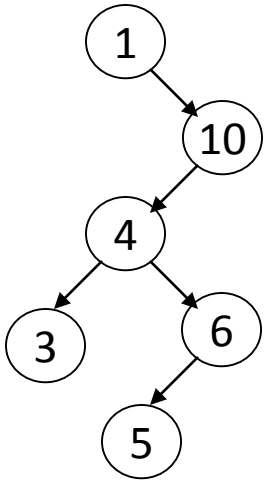
Binary Tree Deletion

1	10	8	4	6	3	5



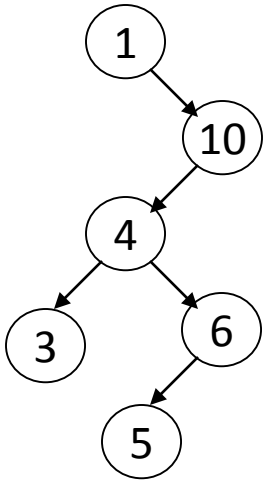
Binary Tree Deletion

1	10	4	6	3	5



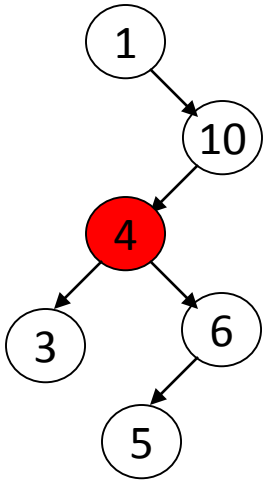
Binary Tree Deletion

1	10	4	6	3	5



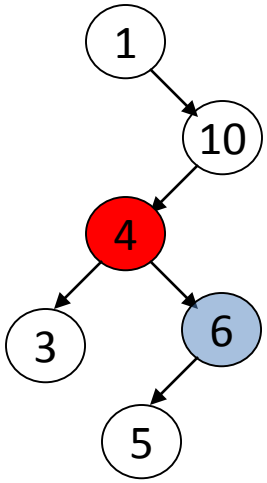
Binary Tree Deletion

1	10	4	6	3	5



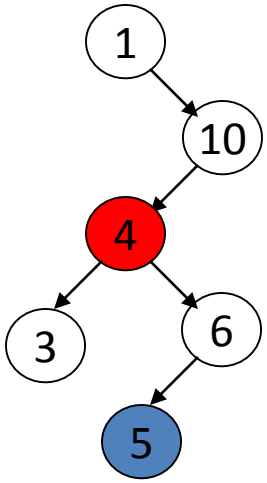
Binary Tree Deletion

1	10	4	6	3	5



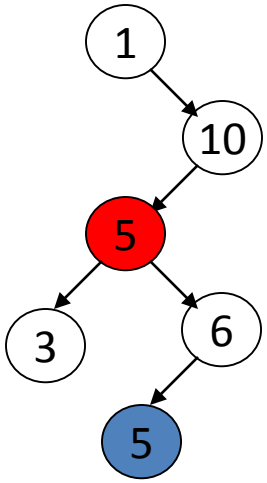
Binary Tree Deletion

1	10	4	6	3	5



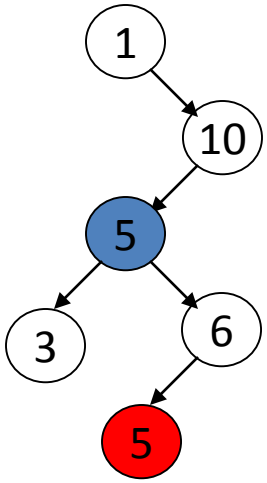
Binary Tree Deletion

1	10	4	6	3	5



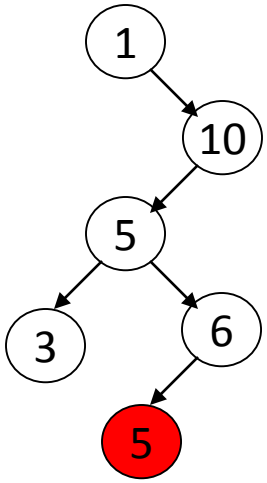
Binary Tree Deletion

1	10	4	6	3	5



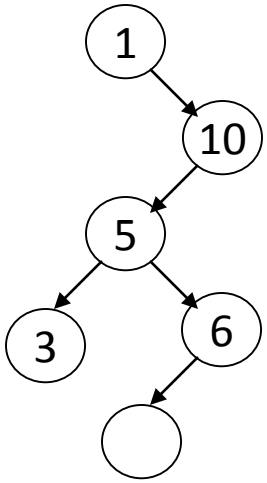
Binary Tree Deletion

1	10	4	6	3	5



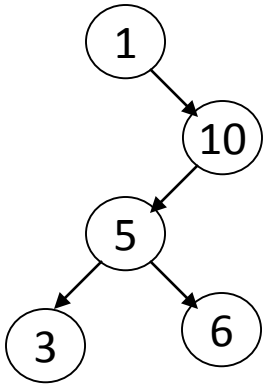
Binary Tree Deletion

1	10	4	6	3	5



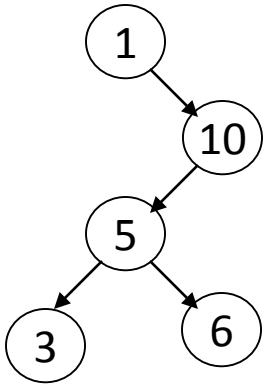
Binary Tree Deletion

1	10	4	6	3	5



Binary Tree Deletion

1	10	6	3	5



Theory Behind the Experiment

The following operations are performed on a Binary Search Tree:

Insertion:

- To insert data all we need to do is follow the branches to an empty subtree and then insert the new node. In other words, all inserts take place at a leaf or at a leaf like node – a node that has only one null subtree.

Deletion: There are the following possible cases when we delete a node:

- The node to be deleted has no children. In this case, all we need to do is delete the node.
- The node to be deleted has only a right subtree. We delete the node and attach the right subtree to the deleted node's parent.
- The node to be deleted has only a left subtree. We delete the node and attach the left subtree to the deleted node's parent.
- The node to be deleted has two subtrees. It is possible to delete a node from the middle of a tree, but the result tends to create very unbalanced trees.

The other important operations are traversal and search

Experimental Procedure

- Analyse the problem statement
- Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
- Implement the algorithm in C language
- Compile the C program
- Design test cases and test the implemented program
- Document the Results
- Analyse and discuss the outcomes of your experiment

Exercise

- Design the Binary Tree ADT and write the program and demonstrate its operations such as insert, delete and traversal operations on integer data. Produce output for binary tree traversals. Tabulate the output for various inputs and verify against expected values. Analyse the efficiency of the algorithm. Describe your learning along with the limitations of overall approach if any. Suggest how these can be overcome.

Key factors and discussion

- The tree ADT should allow all the four operations:
  - Insert
  - Delete
  - Traverse
  - Search
- Test the BST Operations on integer data
  - The input integers should include all cases discussed
- Design the test cases to test the developed program
  - The testing ensures that the program meets the expected results
  - Design the test cases for all the cases considered

Results and Presentations

- Calculations/Computations/Algorithms

  The calculations/computations/algorithms involved in each program has to be presented

- Presentation of Results

  The results for all the valid and invalid cases have to be presented

- Analysis and Discussions

  how the data is manipulated or transformed, what are the key operations involved. Errors encounters and how they are resolved.

- Conclusions

  Summary

Comments

- Limitations of Experiments

  Outline the loopholes in the program, data structures or solution approach.

- Limitations of Results

  Present the test cases; justify if the program is tested correctly considering all the outcomes. Mention what is not tested, if any.

- Learning happened

  What is the overall learning happened

- Conclusions

  Summary

References

- Gilberg, R. F., and Forouzan, B. A. (2007): A Pseudocode Approach With C, 2nd edn. Cengage Learning