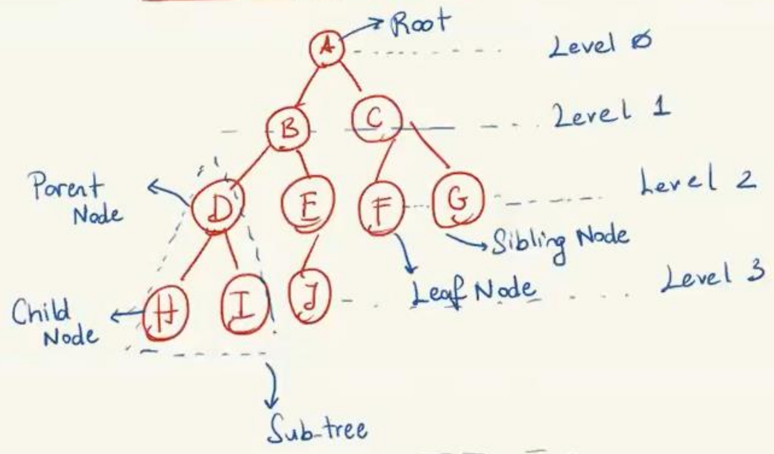
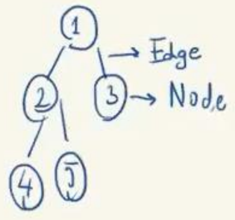
|  | Linear Data Structures | Non-Linear Data Structures |
|---|---|---|
|  | - Array<br>- Linked List<br>- Stack<br>- Queue | - Graphs<br>- Tree (Hierarchial Data Structure) |
| Basic | The data item are arranged in an orderly manner where the elements are attached adjacently | It arranges the data in a sorted order and there exists a relationship between the data elements |
| Ease of implementation | Simpler | Complex |
| Levels involved | Single Level | Multiple Level |
| Memory Utilization | Ineffective | Effective |

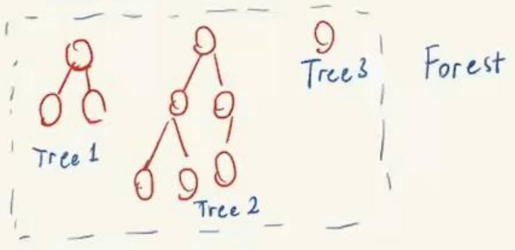
## Tree Data Structure

Terminology

- Path
- Root
- Parent
- Child
- Leaf
- Subtree
- Visiting
- Traversing
- Level
- Forest





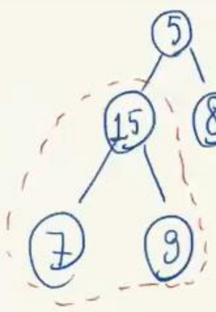


Tree Application in Real-Life

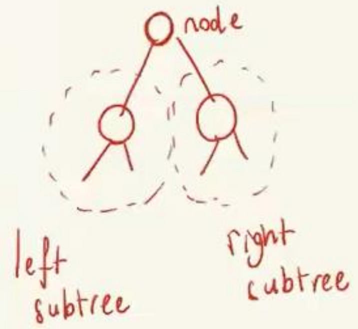
- File Management (File Structure)
- Compression Algorithms (Huffmann Algorithm)
- Databases
- Compilers (Syntax Tree)
- Priority Queue
- AI (Decision Tree, Random Tree)
- Indexing multi-dimensional information such as geographical coordinates, rectangles or polygons

Types of Tree

Binary Tree
Binary Search Tree
AVL Tree
B-Tree
R-Tree
Interval Tree
D-ary Heap
Heap Tree



5 → 15 → 8 → 7 → 9 from top left to right
7 → 9 → 15 → 8 → 5 from bottom left to right



Tree Traversal

|  |  |
|---|---|
| Inorder Traversal | left subtree - node - rightsubtree |
| Preorder Traversal | node - leftsubtree - rightsubtree |
| Postorder Traversal | leftsubtree - rightsubtree - node |

Inorder Traversal

7 → 15 → 9 → 5 → 8

Preorder Traversal

5 → 15 → 7 → 9 → 8

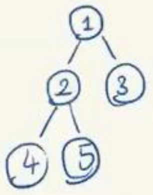
Postorder Traversal

7 → 9 → 15 → 8 → 5

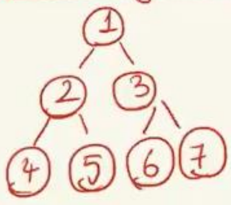
```
struct node {
    int data ;
    struct node * leftc;
    struct node * rightc;
};

void inorderTraversal (struct node *root) {
    if (root == NULL) return;
    inorder Traversal (root->left);
    printf ("%d", root->data);
    inorder Traversal (root->right);
}

void preorder Traversal ( struct node *root ) {
    if (root == NULL) return;
    printf ("%d" , root->data);
    preorder Traversal (root->left);
    preorder Traversal (root->right);
}

void postOrder Traversal (struct node *root){
    if (root == NULL) return;
    postorder Traversal (root->left);
    postorder Traversal (root->right);
    printf ("%d", root->data);
}
```

Tree

Binary Tree

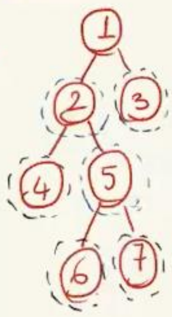


Perfect Binary Tree

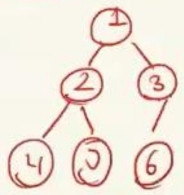


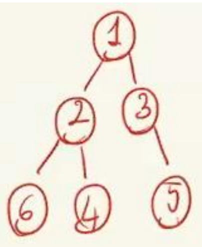
Leaf nodes should be at the same level

Full Binary Tree

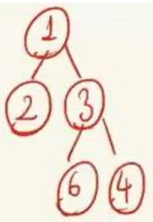


Complete Binary Tree

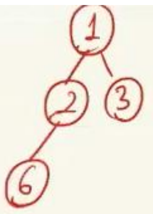




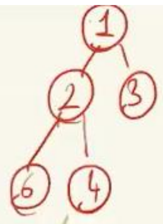
✗ Full Binary Tree
✗ Complete Binary Tree



✓ FBT
✗ CBT

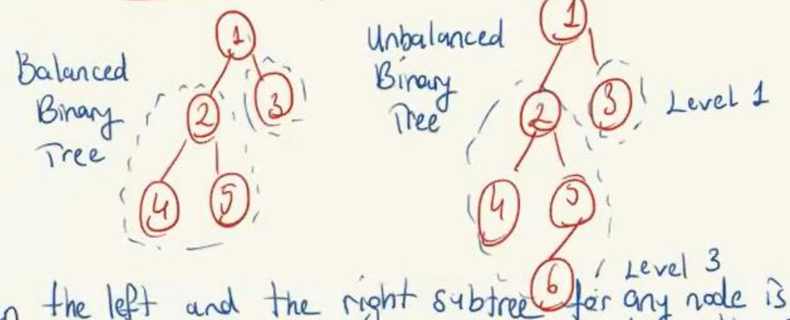


✗ FBT
✓ CBT



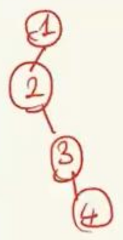
✓ FBT
✓ CBT

## Balanced Binary Tree

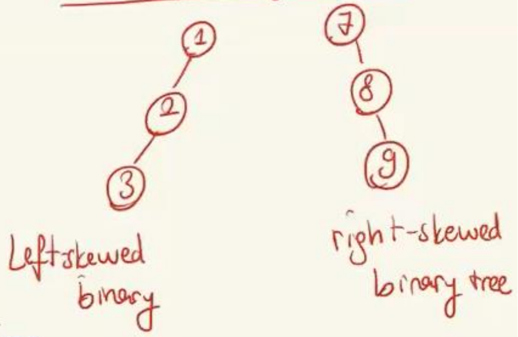


Difference between the left and the right subtree for any node is not more than one

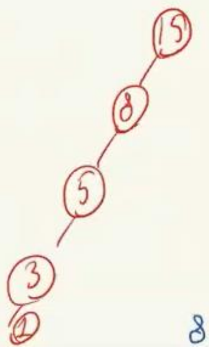
## Degenerate or Pathological Tree

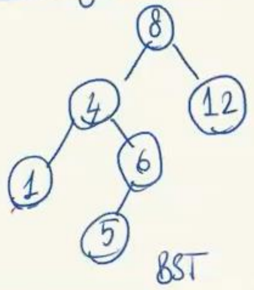


## Skewed Binary Tree



## Binary Search Tree



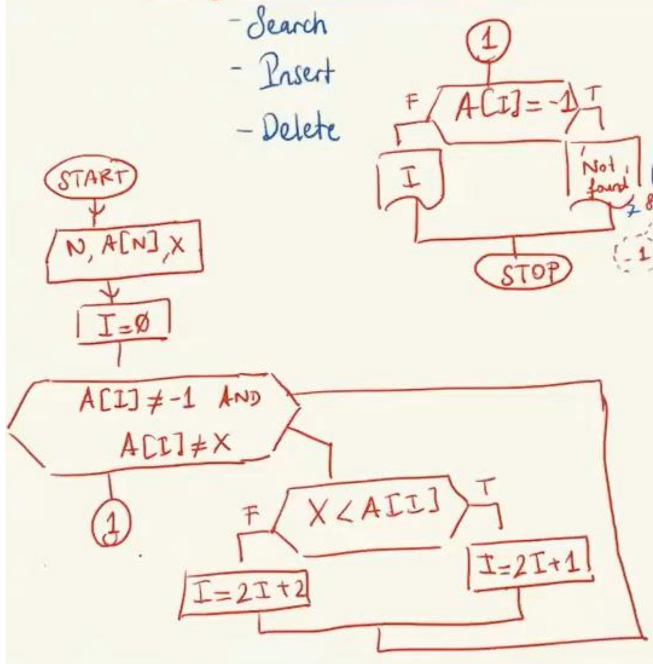


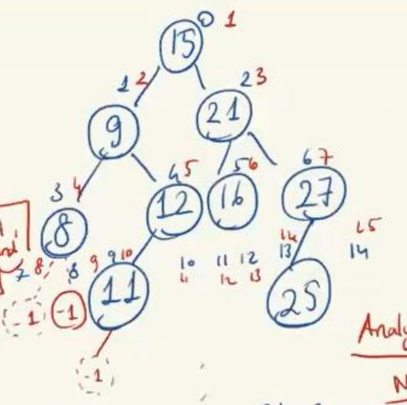
- All the nodes of the left subtree are less than the root node
- All the nodes of the right subtree are greater than the root node.

8 4 12 1 6 5

## Binary Search Tree Operations

- Search
- Insert
- Delete





| | |
|---|---|
| k=∅ | k=∅ |
| k=1 | k=2 |
| k=3 | k=6 |
| k=7 | k=14 |

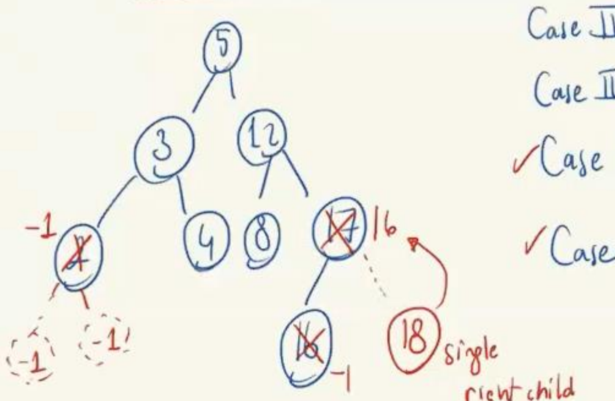
| | |
|---|---|
| 2k+1 | 2k+2 |
| 2k | 2k+1 |

Analysis

| N | X | I | A[I] |
|---|---|---|---|
| | 8 | ∅ | 15 |
| | | 1 | 9 |
| | | 3 | 8 |
| | 7 | 0 | 15 |
| | | 1 | 9 |
| | | 3 | 8 |
| | | 7 | -1 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 3 | 5 | 7 | 8 | 12 | 19 |

## Delete operation on a BST

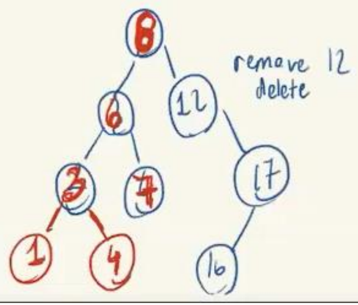


Case IV not found

Case III internal node (right, left child)

✓Case II internal node with a single child (leafnode)

✓Case I Leaf node



## Binary Search Tree Complexities

| Operation | Best Case | Worst Case |
|---|---|---|
| Search | O(log n) | O(n) |
| Insert | O(log n) | O(n) |
| Delete | O(log n) | O(n) |

Space Complexity O(n)