Data Structures (15B11CI311)

Odd Semester 2020

3rd Semester , Computer Science and Engineering

Jaypee Institute Of Information Technology (JIIT), Noida

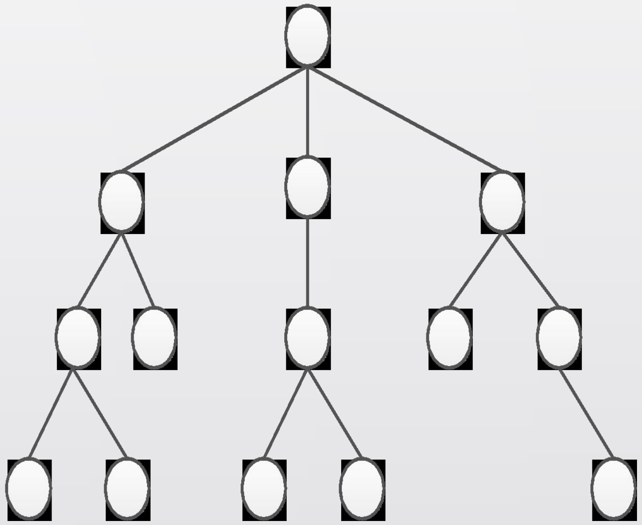
Lecture: 25

Topics to be covered:

- Binary tree and its structure
- Basic Operation of Binary Tree
- Recursive Traversal Algorithms

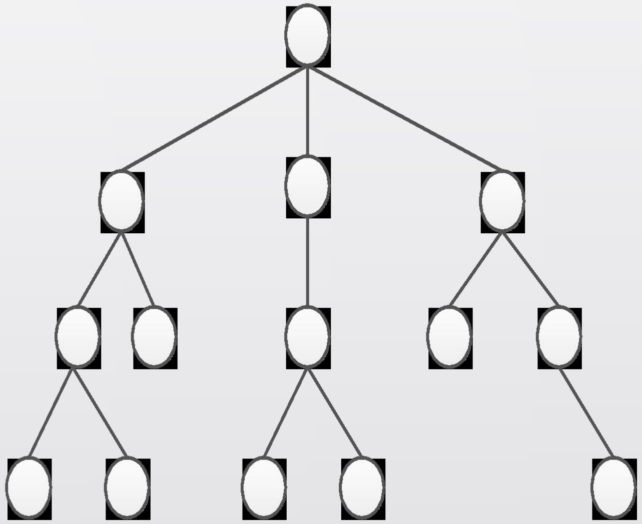
Tree

- A **tree** is a finite set of nodes, arranged in a hierarchal order. Each node is associated with some value.
- The node at the top of the hierarchy is the known as **root**.
- The nodes are connected in a **parent-child** relationship.
- The next nodes in the hierarchy after the root element are the **children** of the root, i.e. root will be the **parent** of these nodes.
- The next hierarchy nodes are the **grandchildren** of the root, and so on.



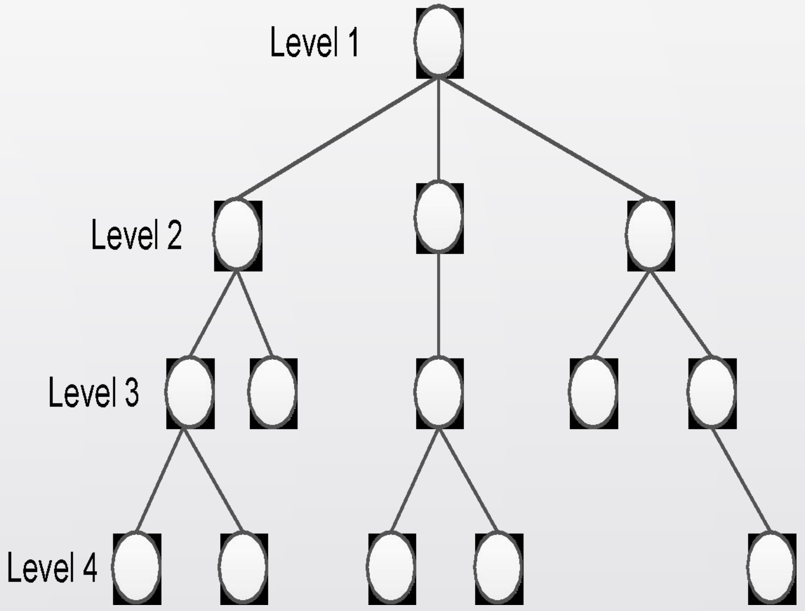
Tree

- The nodes, having same parent are known as **siblings**.
- Similarly, **ancestors** and **descendants** can be defined in a tree.
- Nodes, having no children are known as **leaves**.
- A tree can be partitioned into **subtrees**.
- Trees are useful when hierarchically ordered data is required like
  - Employees of a corporation
    - President, vice presidents, managers, and so on
  - Java’s classes
    - Object is at the top of the hierarchy
    - Subclasses of Object are next, and so on



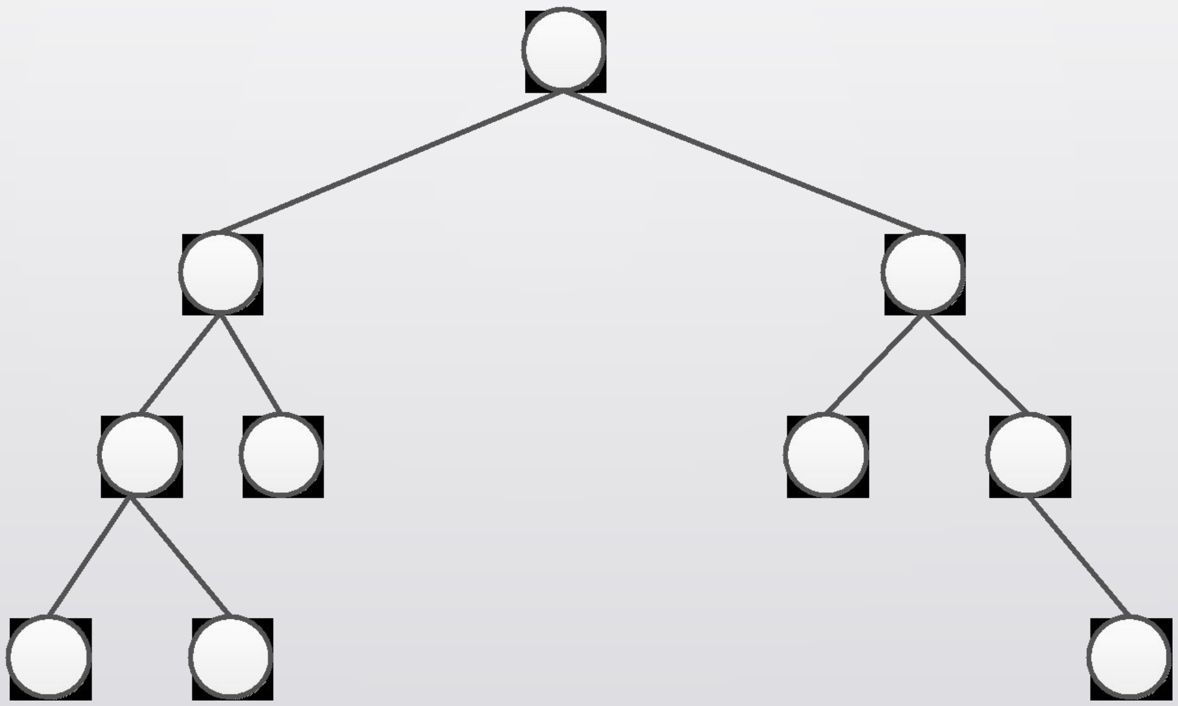
Tree

- Each hierarchical level is given a number, i.e. 1, 2, and so on
- The root is at level 1 and its children at level 2 and so on.
- In some text, level starts at 0.
- **Note**: here, we will consider the label at 1.
- Height = Depth = Number of levels
- Node Degree = Number of Children
- Tree Degree = Maximum Node Degree



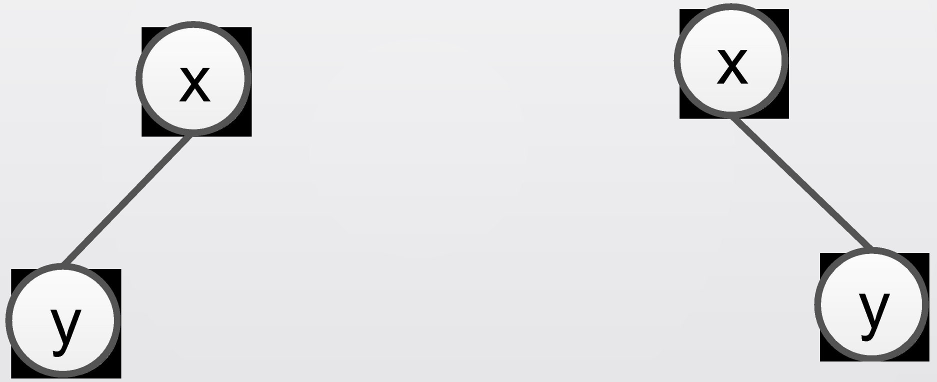
Binary Tree

- **Binary Tree** is a tree which is either empty or has at most two subtrees, each of the subtrees also being a binary tree.
- It means each node in a binary tree can have 0, 1 or 2 subtrees.



Difference between A Binary Tree and A Tree

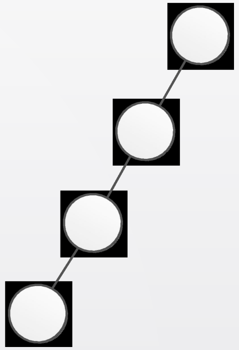
- Each node in a binary tree has a maximum degree 2, while node degree in a tree has no limit
- A binary tree may be empty; a tree cannot be empty
- The subtrees of a binary tree are ordered, while in a tree nodes are not ordered.



- These two binary trees are different
- In terms of a tree, these are the same

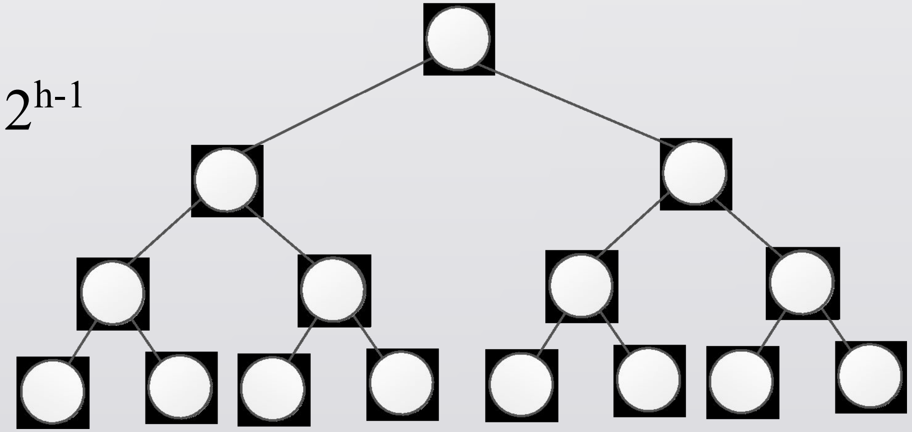
Properties in Binary Tree

- Minimum number of nodes in a binary tree with height h:
  - At least one node at each of the h levels
  - **Minimum number of nodes** = h



- Maximum number of nodes in a binary tree with height h:
  - All possible nodes at each of the h levels
  - **Maximum number of nodes** $= 1 + 2 + 4 + 8 + \ldots + 2^{h-1}$

    $= 2^h - 1$

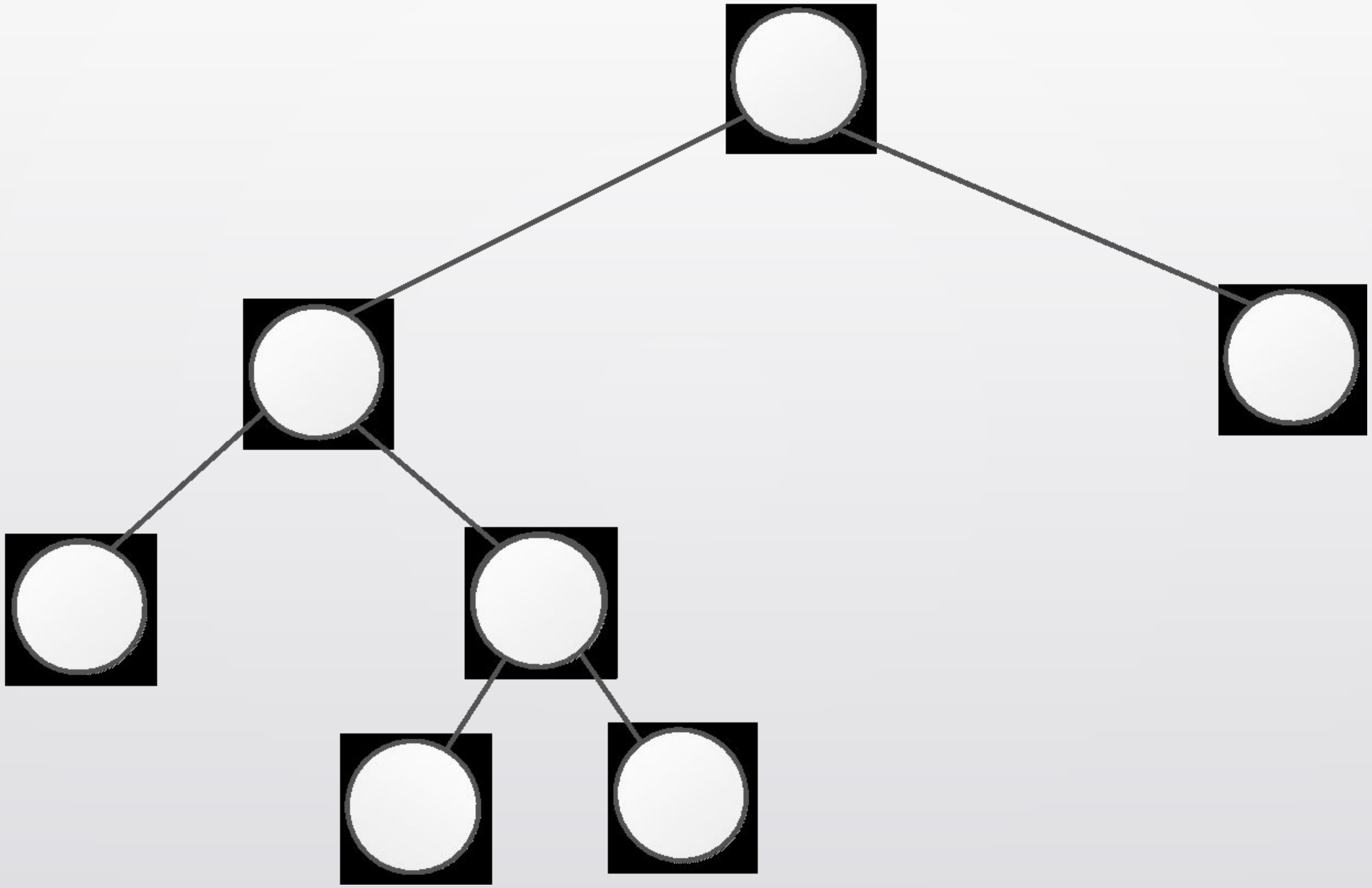


Properties in Binary Tree

- Let n be the number of nodes in a binary tree with height h:
  - $h <= n <= 2^h - 1$
  - $\log_2(n+1) <= h <= n$

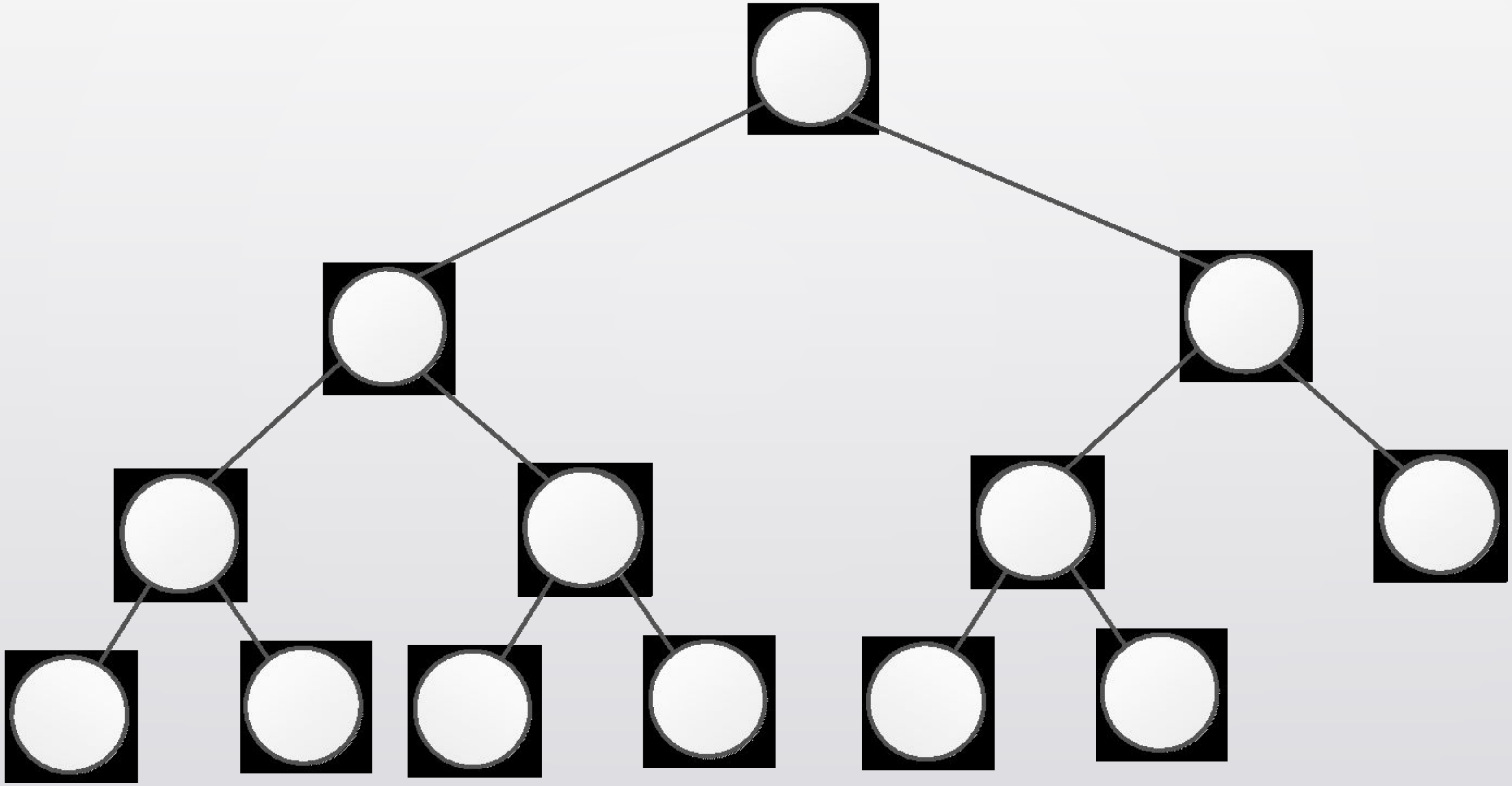
Terminology of Binary Tree

- **Full binary tree:** A binary tree in which each node has exactly zero or two children.



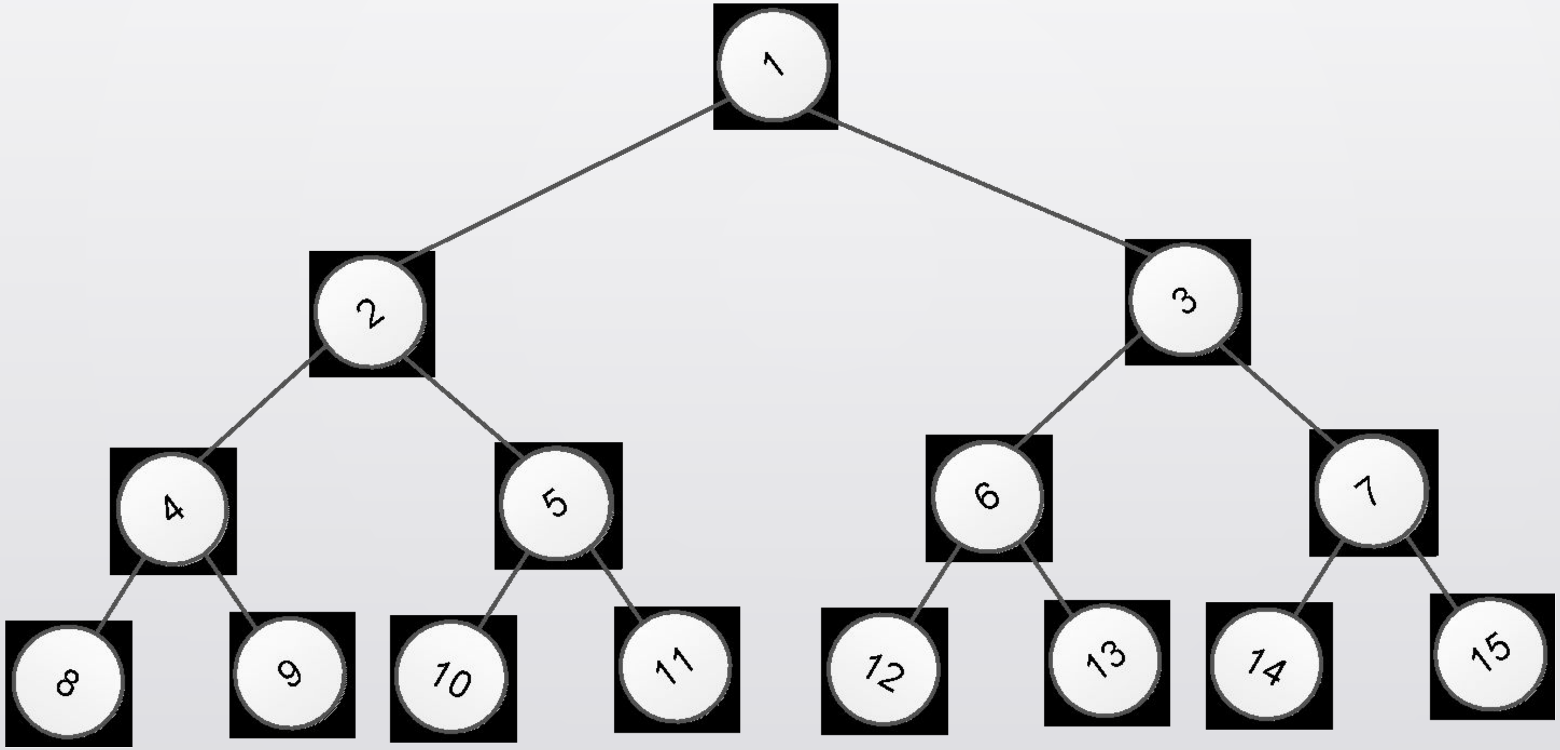
Terminology of Binary Tree

- **Complete binary tree:** A binary tree, which is completely filled, with the possible exception of the bottom level, which is filled from left to right.



Terminology of Binary Tree

- **Complete binary tree:** In a complete binary tree, nodes can be numbered from 1 to $2^h - 1$.
- Numbers are labelled from top to bottom and left to right.



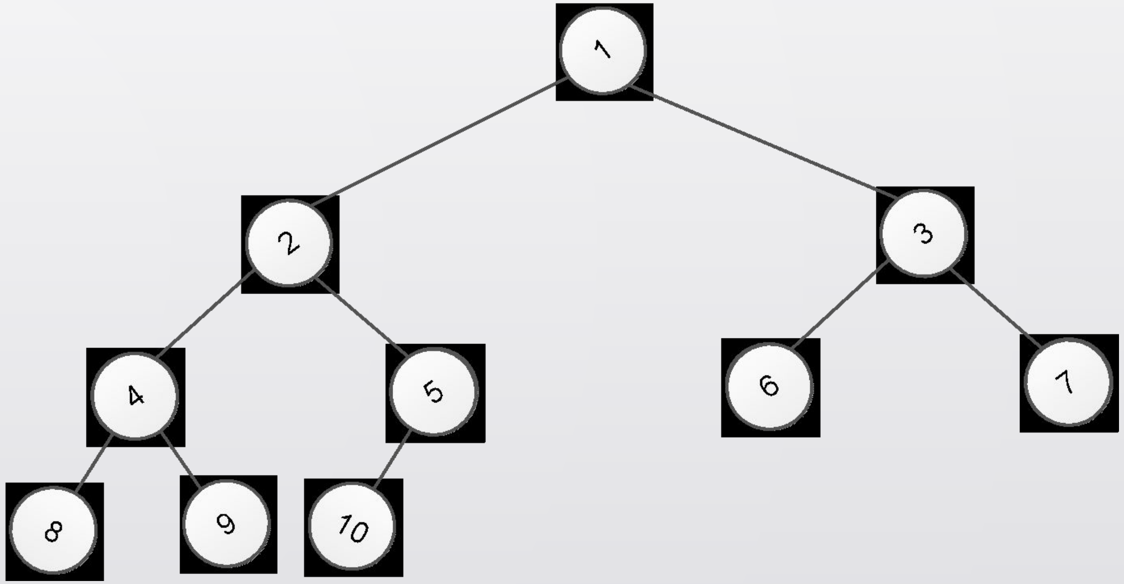
Representation of Binary Tree

- There are two ways for representation of binary tree.
  - Array representation
  - Linked List representation

Array Representation of Binary Tree

- A single dimensional array can be used to represent a binary tree.
- BT =

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

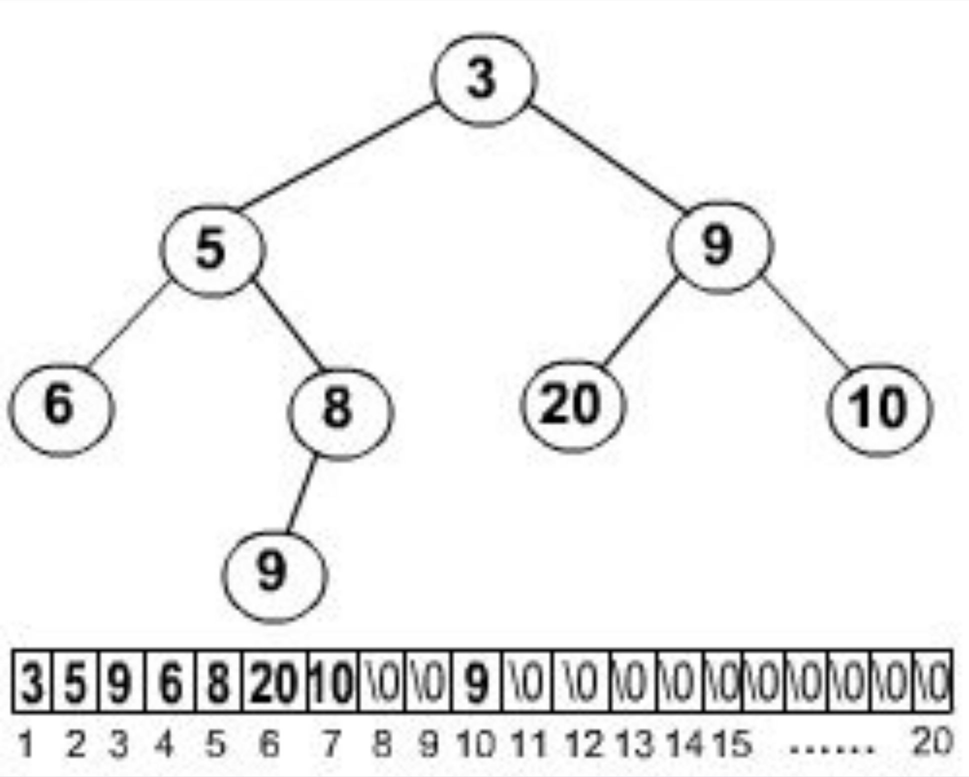


- Parent of node i is node i / 2 , unless node 1
- Node 1 is the root and has no parent.
- Left child of node i is node 2i, unless n, where n is the number of nodes.
- If 2i > n, node i has no left child.
- Right child of node i is node 2i+1, unless 2i+1 > n, where n is the number of nodes.
- If 2i + 1 > n, node i has no right child.

Linked Representation of Binary Tree

- Each node of a binary tree has one data field and two pointers, one for the right child node and other for the left child node.
- struct node{

```
struct node{
    int data;
    node *lchild;
    node *rchild;
};
```



Source: https://nptel.ac.in/content/storage2/courses/106103069/Module_9/array_rep_btree.htm

- If any node has its left or right child empty then it's respective link will have a null value.
- In a leaf node, both links has a null value.

Building Binary Tree with Array Representation


```
#include <iostream>
 using namespace std;
 struct node
{
  int data;
  node *lchild;
  node *rchild;
};
struct node *buildtree (int);   /* builds the tree */
int a[] = { 3, 5, 9, 6, 8, 20, 10, 0, 0, 9,0,0,0,0,0};
```

```
node *buildtree (int n) {
  node *temp = NULL;
  if(n>sizeof(a)/sizeof(int))
        return temp;
  else if (a[n])  {
      temp = new node;
      temp->data = a[n];
      temp->lchild = buildtree (2*n + 1);
      temp->rchild = buildtree (2*n + 2);
    }
    else
      return temp;
    }
```

Building Binary Tree with Array Representation


```
void inorder (node *root)
{
    if (root != NULL)
        {
          inorder (root->lchild);
          cout<<root->data;
          inorder (root->rchild);
        }
}
```

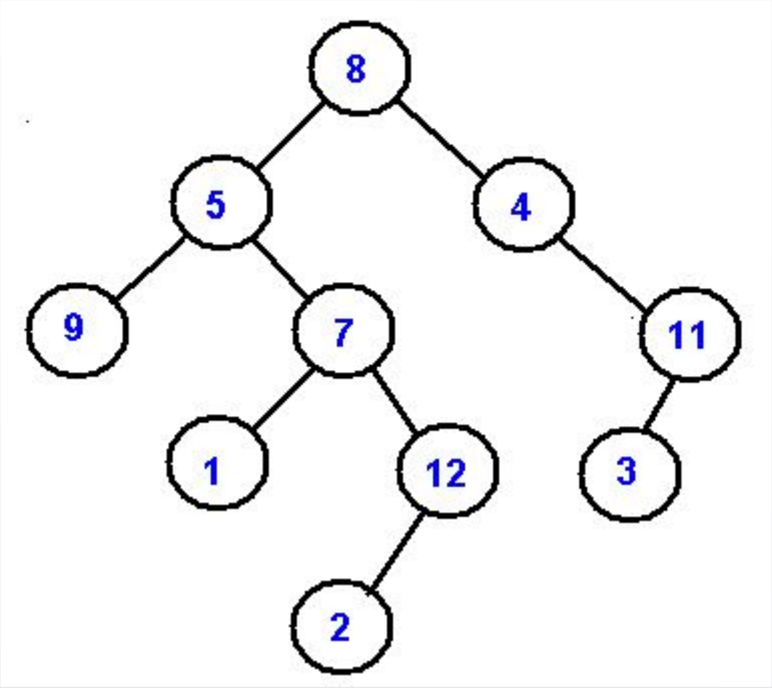
```
int main()
{
  node *root;
  cout<<sizeof(a)<<endl;
  root = buildtree (0);
  cout <<"\n Inorder Traversal\n";
  inorder(root);
  return 0;
}
```

Binary Tree Traversal

- A traversal is a process that visits all the nodes in the tree exactly once.
  - depth-first traversal
    - Preorder traversal
    - Inorder traversal
    - Postorder traversal
  - breadth-first traversal
    - Level order traversal
- Applications:
  - Determine height of the tree
  - Finding the number of nodes

Preorder Traversal

- Traverse the nodes in the following order recursively
  - Visit the root
  - Traverse the left subtree
  - Traverse the right subtree



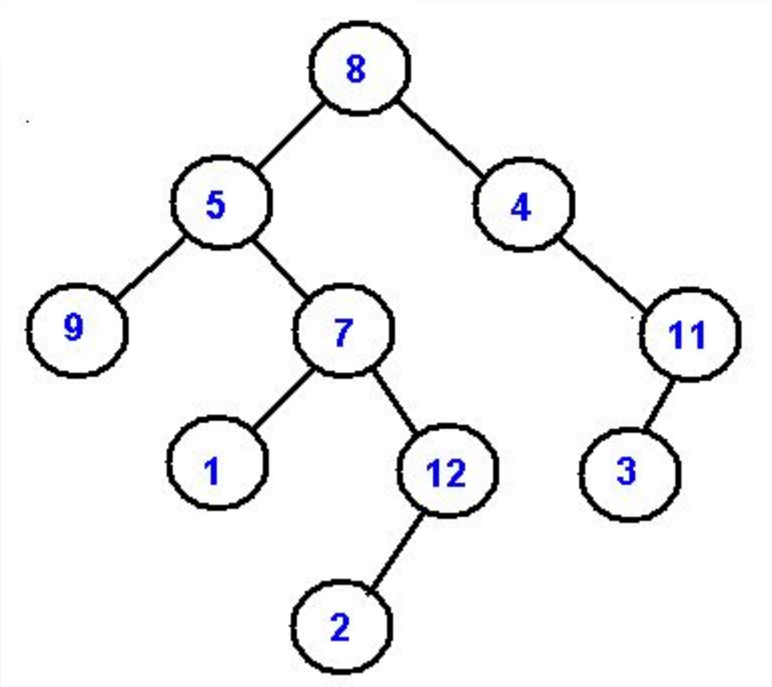
Preorder - 8, 5, 9, 7, 1, 12, 2, 4, 11, 3

Preorder Traversal: Recursive Implementation

```
void preorder(node * root)
    {
        if(root != NULL)
        {
            cout<<root->data;
            preorder(root-> lchild);
            preorder(root->rchild);
        }
    }
```

Inorder Traversal

- Traverse the nodes in the following order recursively
  - Traverse the left subtree
  - Visit the root
  - Traverse the right subtree



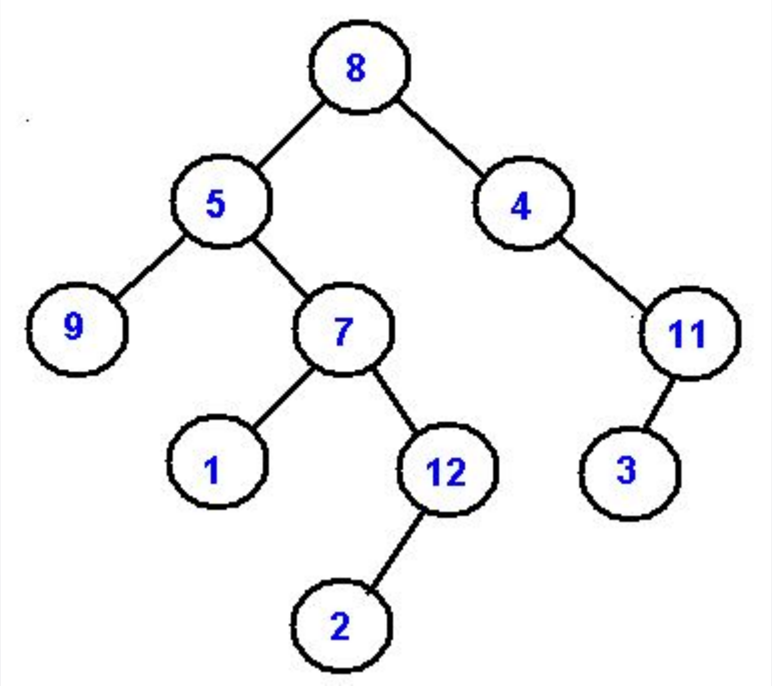
Inorder - 9, 5, 1, 7, 2, 12, 8, 4, 3, 11

Inorder Traversal: Recursive Implementation

```
void Inorder(node * root)
    {
        if(root != NULL)
        {
          Inorder(root-> lchild);
          cout<<root->data;
          Inorder(root->rchild);
        }
    }
```

Postorder Traversal

- Traverse the nodes in the following order recursively
  - Traverse the left subtree
  - Traverse the right subtree
  - Visit the root



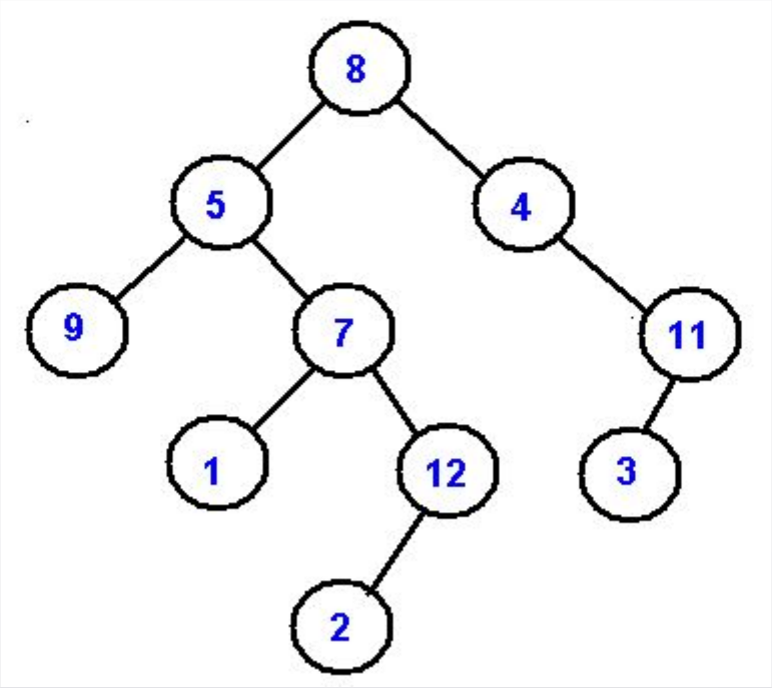
Postorder - 9, 1, 2, 12, 7, 5, 3, 11, 4, 8

Postorder Traversal: Recursive Implementation

```
void Postorder(node * root)
    {
        if(root != NULL)
        {
          Postorder(root-> lchild);
          Postorder(root->rchild);
          cout<<root->data;
        }
    }
```

Levelorder Traversal

- Traverse the nodes in the following order recursively
  - Visit the root
  - Insert its children into a Queue (FIFO)
  - Remove a node from queue and call it root



Levelorder - 8, 5, 4, 9, 7, 11, 1, 12, 3, 2

Levelorder Traversal: Recursive Implementation

```
void Levelorder (node *root) {
  if (root != NULL)     {
      cout<< root->data;
      if (root->lchild)
            Q.push (root->lchild);
      if (root->lchild)
            Q.push (root->rchild);
       node *temp = Q.front();
       Q.pop();
       Levelorder (temp);
    }
}
```

References

- *Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009) [1990]*
- *Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill. ISBN 0-262-03384-4. 1320 pp.*
- Adam Drozdek, Data Structures and Algorithms in C++ (2nd Edition), 2001