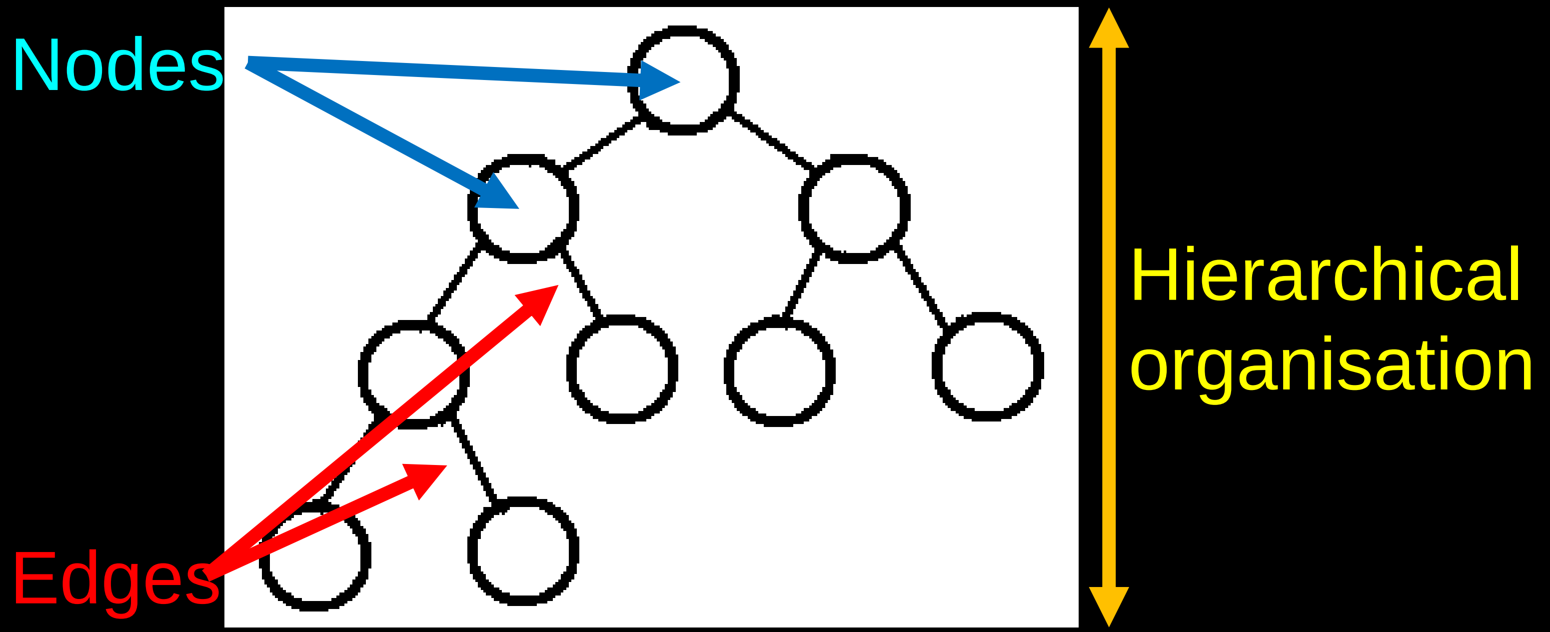
Binary Tree

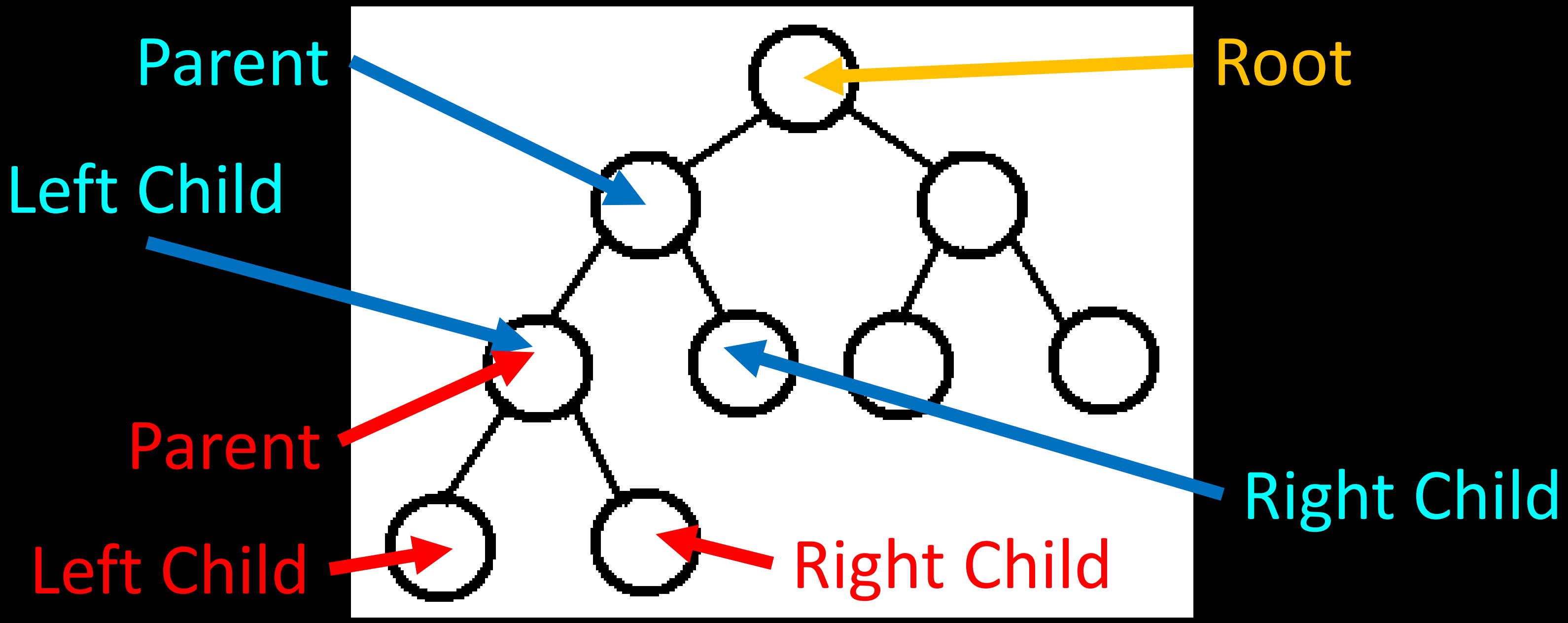
What is a Tree?

- A non-linear data structure whose entries follow a hierarchical organization.
- Contains a set of **nodes** connected by **edges**.



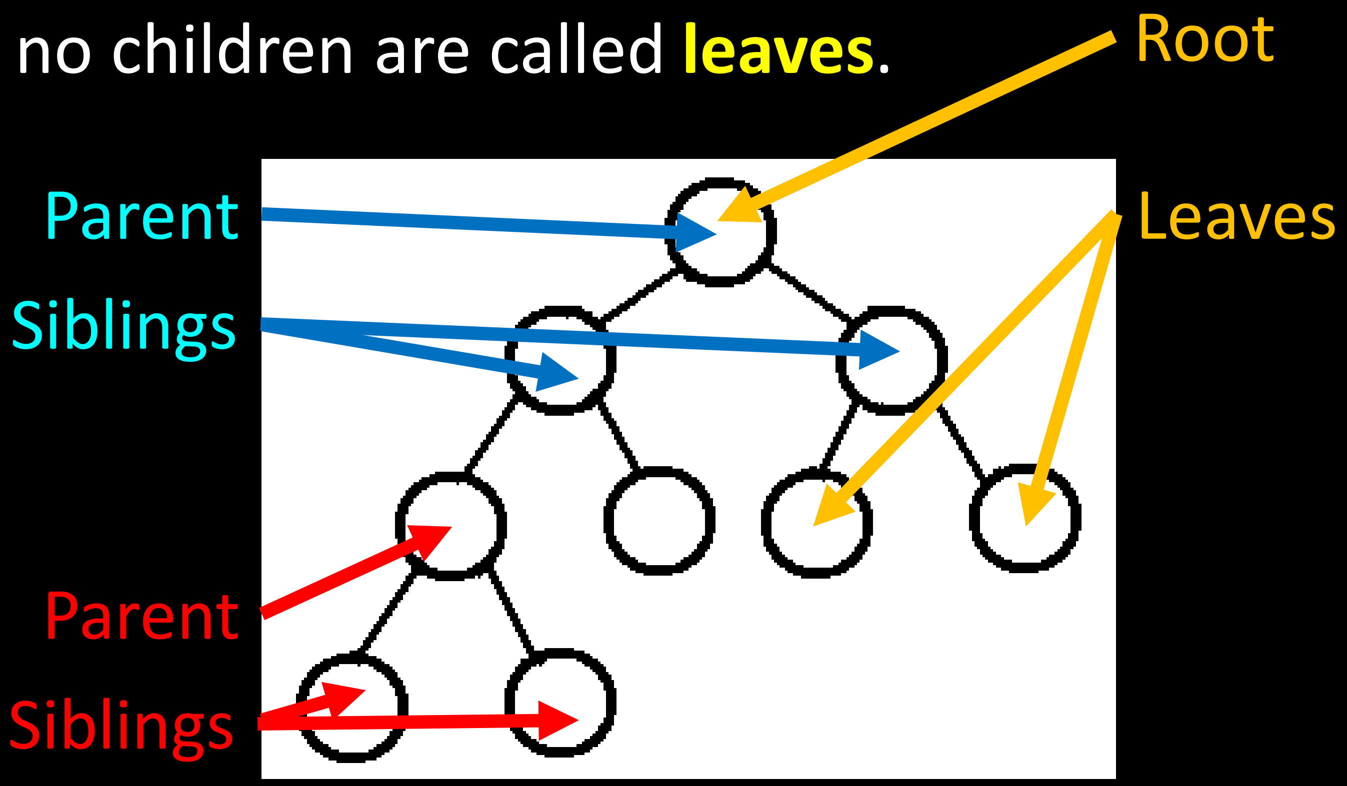
Binary Tree data structure

- Every node has one **parent**, except for the **root**.
- Every node can have zero, one or **at most two** children



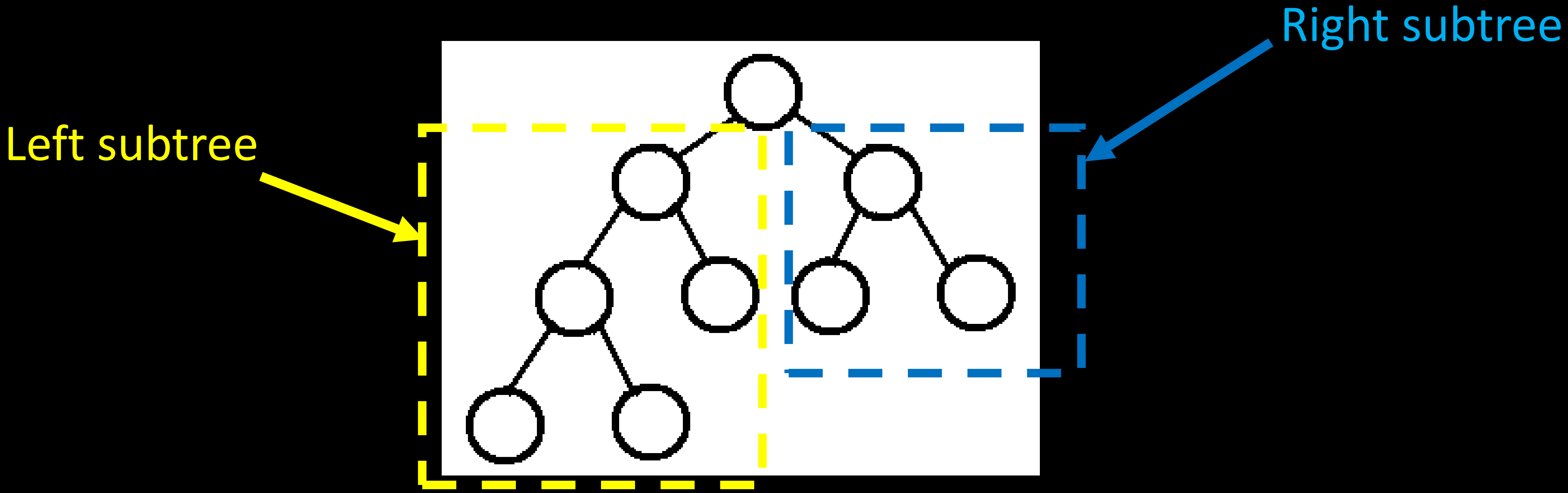
Binary Tree data structure

- Children from the same parent are called **siblings**.
- Nodes with no children are called **leaves**.



Binary Tree data structure

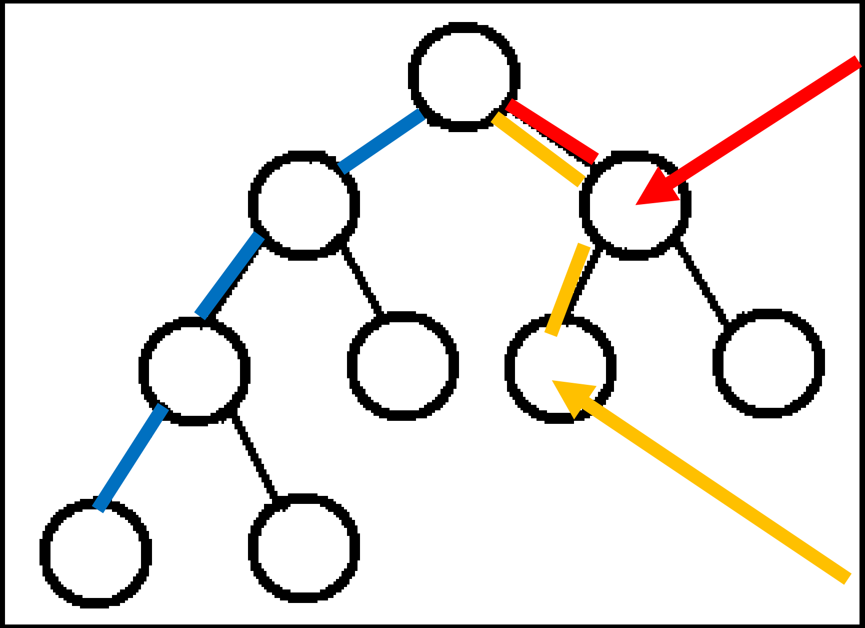
- Every node contains two subtrees which are also binary trees.



Binary Tree data structure

- The number of edges of the longest path from the root to a leaf is called the **height of the tree**.
- The number of edges of the path from the root to a node is called the **depth of the node**.

Height of tree = 3



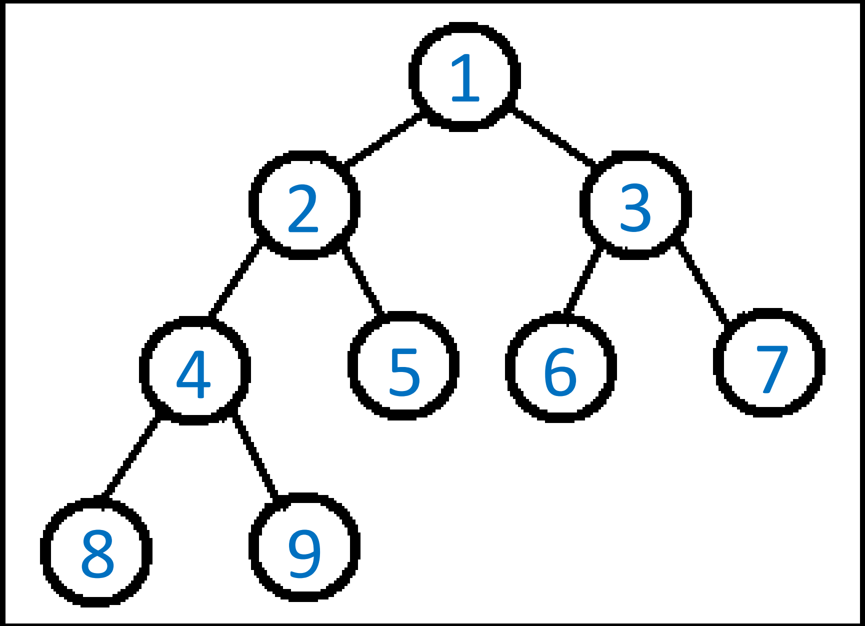
Depth of node = 1

Depth of node = 2

Binary Tree data structure

The **size** of a tree is equivalent to the number of nodes.

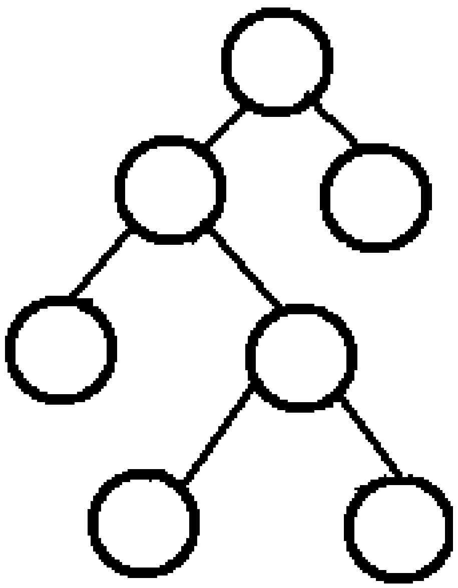
Size = 9



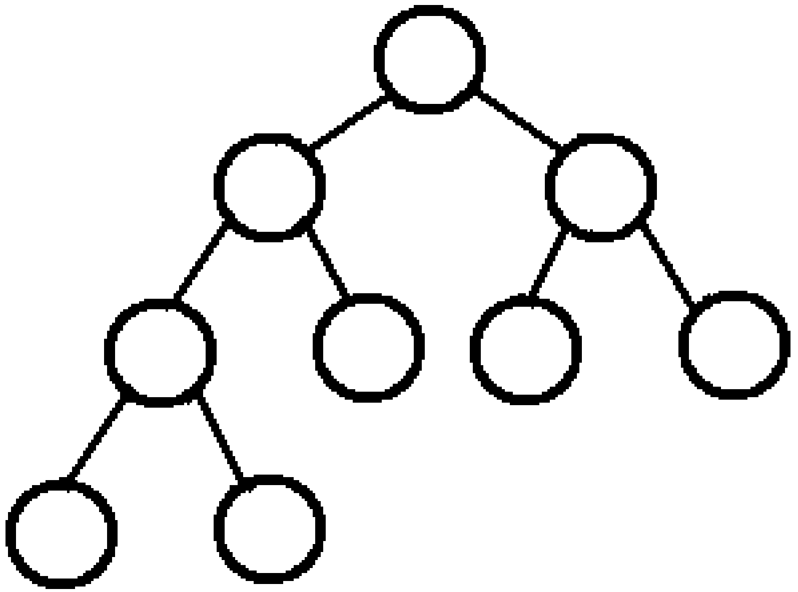
Binary Tree data structure

Full tree– A tree in which every node, except the leaves, has two children.

Complete tree– A tree in which every level, except possibly the last, is completely filled. Nodes are as far left as possible.



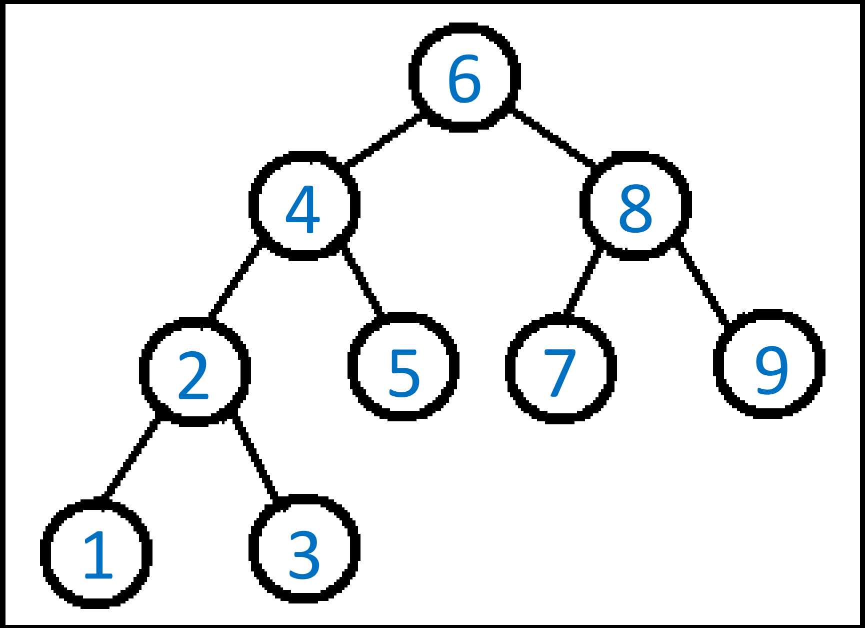
full tree



complete tree

Binary Search Tree

- Binary search tree is a binary tree with relative ordering in how the nodes are organized
- Each node stores a distinct key.
- All the nodes to the left of a node have keys less than the key of the node
- All the nodes to the right of a node have keys greater than the key of the node.



Binary Tree traversals

- The process of checking/visiting each node once.
- Applications include:
  - Printing all values in a binary tree
  - Determining if there is one or more nodes with some property
  - Making a copy

- The order of traversal depends on the algorithm used.
- Three such algorithms are:
  - Pre-order
  - In-order
  - Post-order

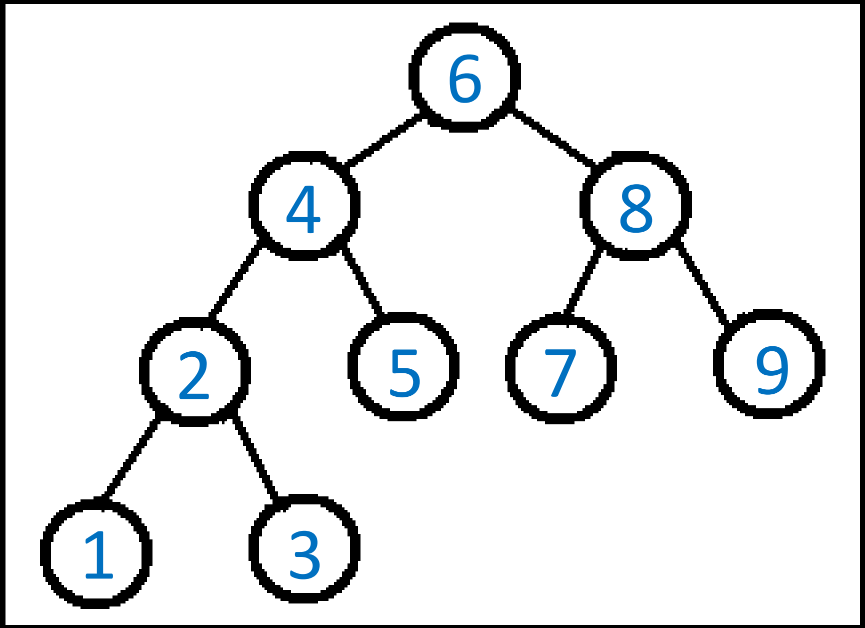
  https://www.educative.io/collection/page/10370001/160001/220001

Pre-order traversal

Starting at the root,

We traverse in the following order:

1. First visit the node itself.
2. Then visit the left subtree of the node.
3. Then visit the right subtree of the node.

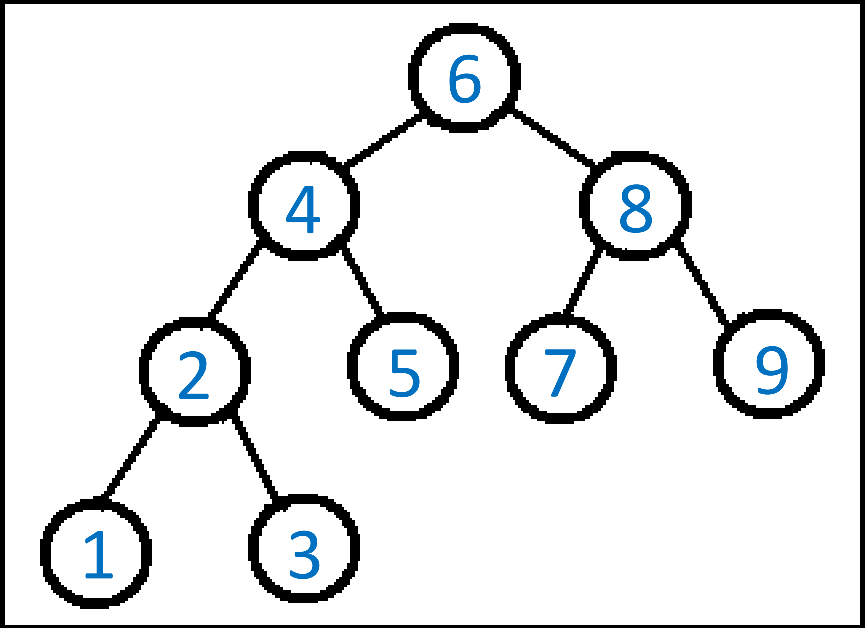


In-order traversal

Starting at the root,

We traverse in the following order:

1. First visit the left subtree of the node.
2. Then visit the node itself.
3. Then visit the right subtree of the node.

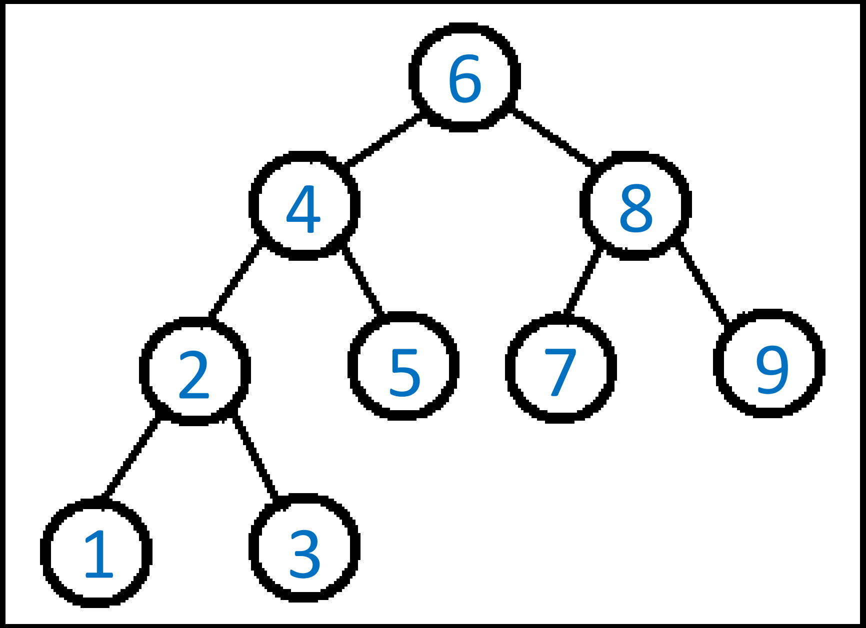


Post-order traversal

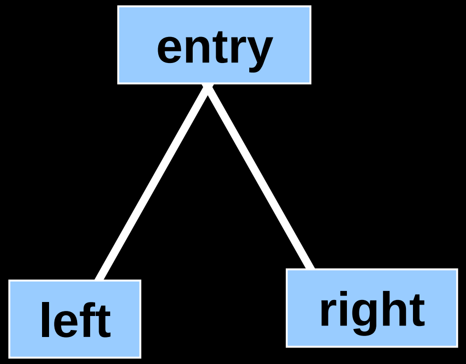
Starting at the root,

We traverse in the following order:

1. First visit the left subtree of the node.
2. Then visit the right subtree of the node.
3. Then visit the node itself.



Binary Search Tree ADT



- Constructors:

```
def make_tree(entry, left, right):
    return [entry, left, right]
```

- Accessors:

```
def entry(tree):
    return tree[0]

def left_branch(tree):
    return tree[1]

def right_branch(tree):
    return tree[2]
```

Binary Search Tree ADT

- Predicates:

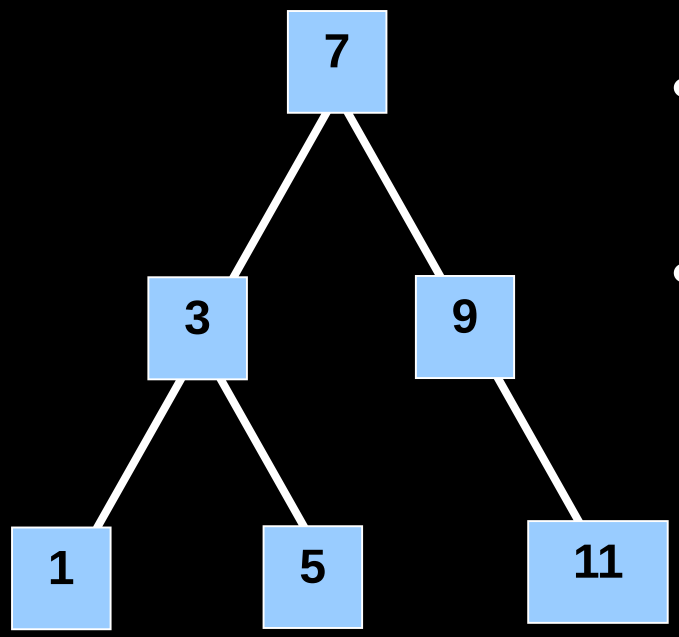
```
def is_empty_tree(tree):
    return tree==[]

def contains(x, tree):
    (left as an exercise in
  tutorial)
```

- Modifiers:

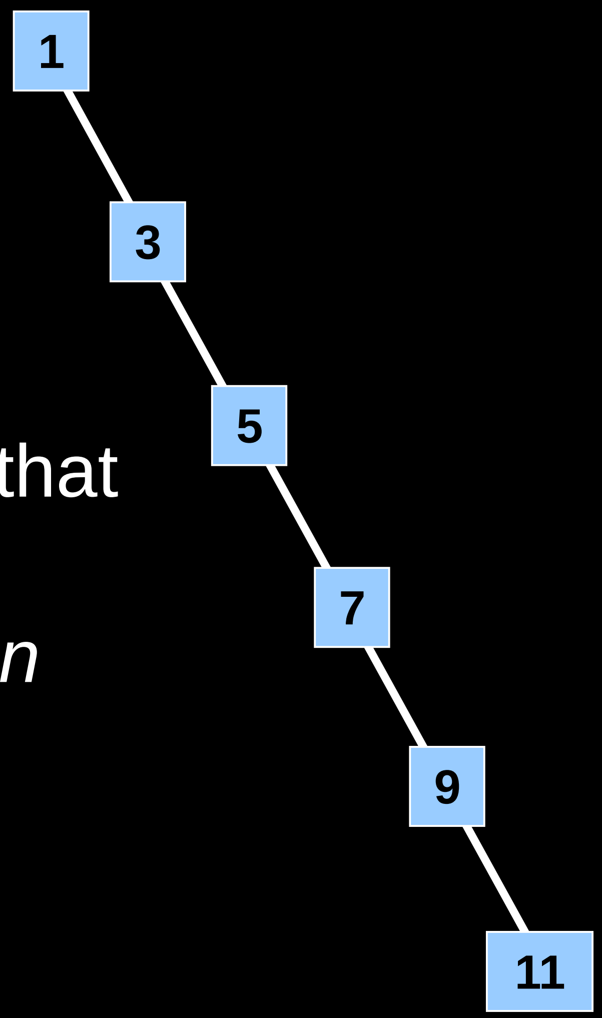
```
def insert(x, tree):
    (left as an exercise in
  tutorial)

def remove(x, tree):
    (left as an extra
  practice)
```

- Time complexity: *O(log n)*

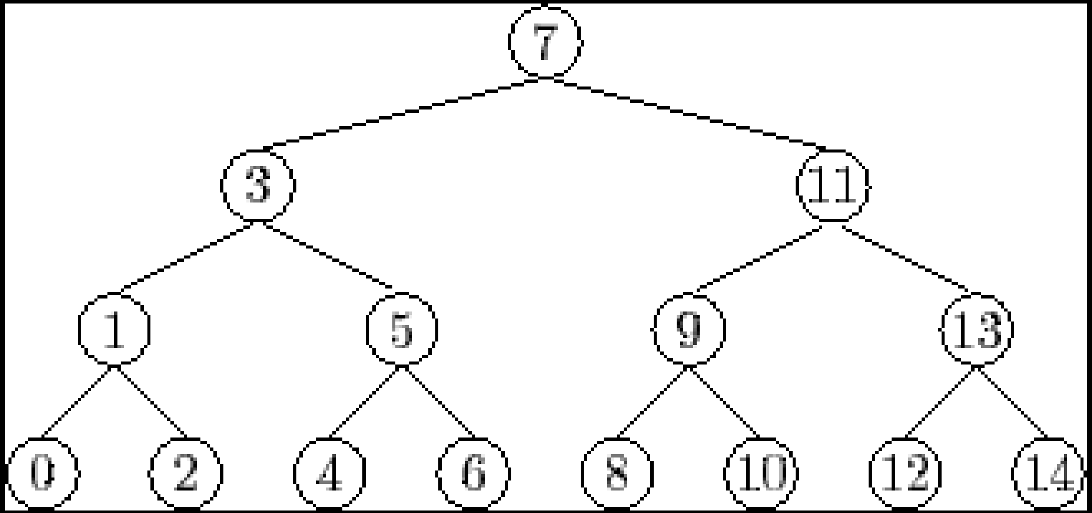
Balanced trees



- Operation is *O(log n)* assuming that tree is balanced.
- Unbalanced trees break the log *n* complexity.



Balanced trees



- For a complete tree with height h, where all levels are filled up, the total number of nodes,

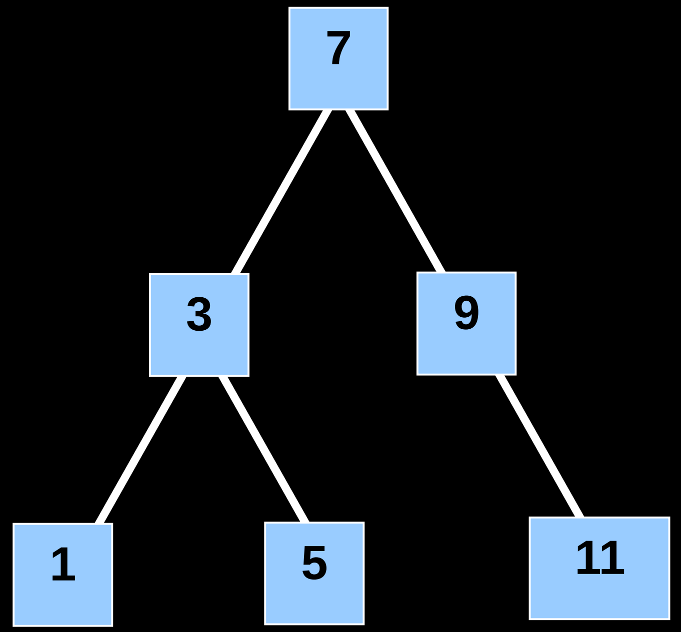
$n = 1 + 2 + 4 + \ldots. + 2^{h-1} + 2^{h}$

$= (2^{h+1} - 1)$ [sum of a GP]

- Since the time needed for search/insertion depends directly on h, making h the subject, we have:
- $h = \log_2(n-1)-1$
- Using big-O,
- Time complexity: O(log n)

Binary Search Tree vs Binary Search

- Searching using a binary search tree works the same way as the binary search in a linear sorted array.
- The algorithm for binary search also has time complexity of O(log n) since the search scope is always reduced by half for every iteration, making it one of the most time efficient algorithm.



{1,3,5,7,9,11}