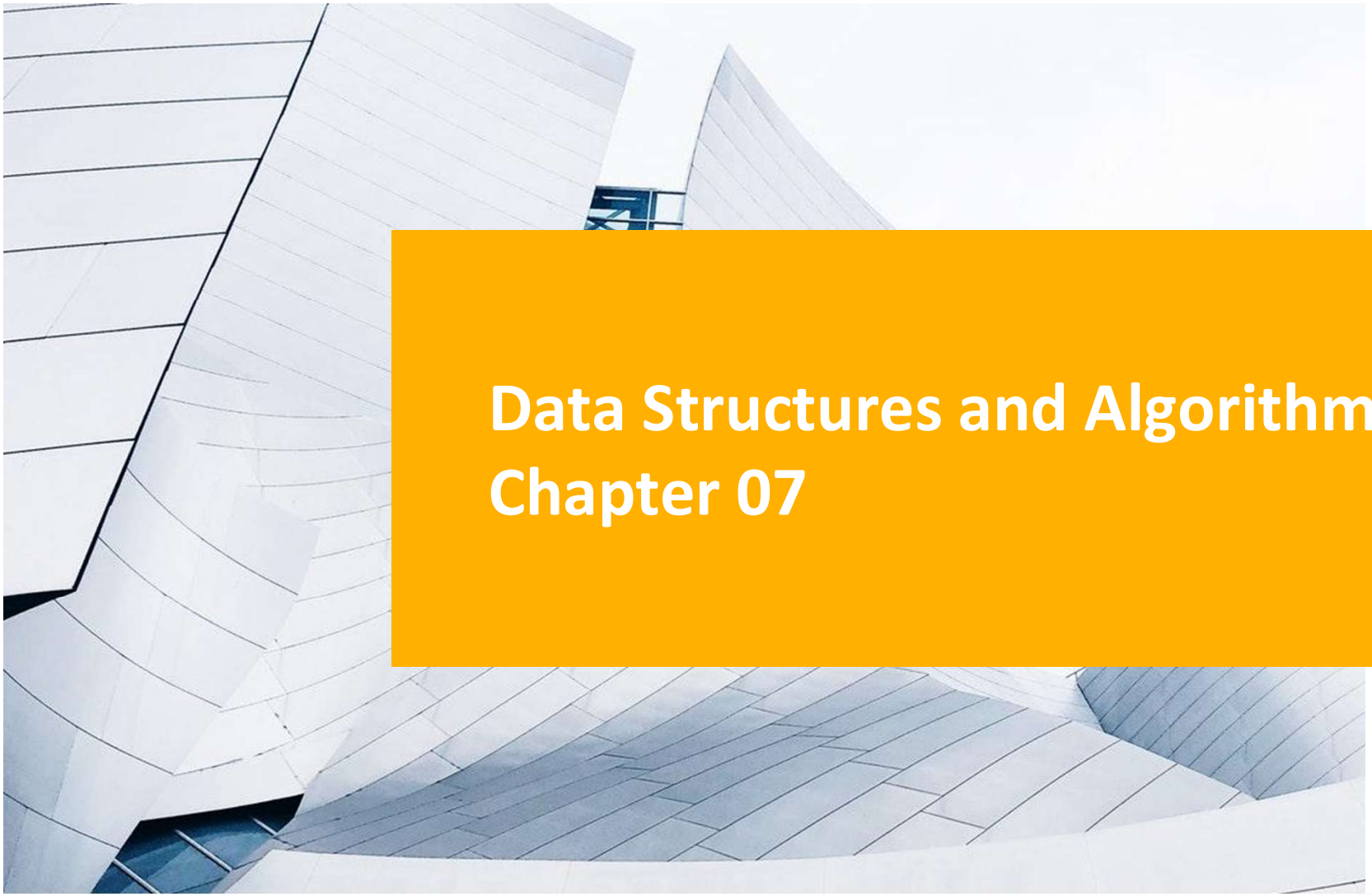


Data Structures and Algorithm
Chapter 07

Trees – Binary Tree

Data Structures and Algorithm

Trees

Siblings

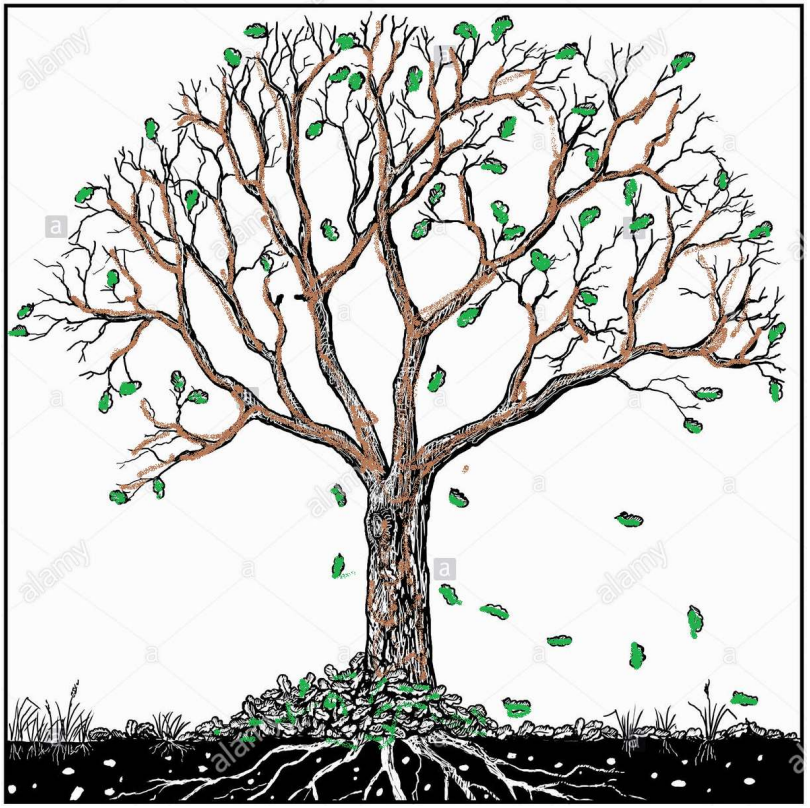
Degree of node

Successor node

Children

Parent node

Root



Depth

Internal node

Leaf node

Height

Path

Ancestor

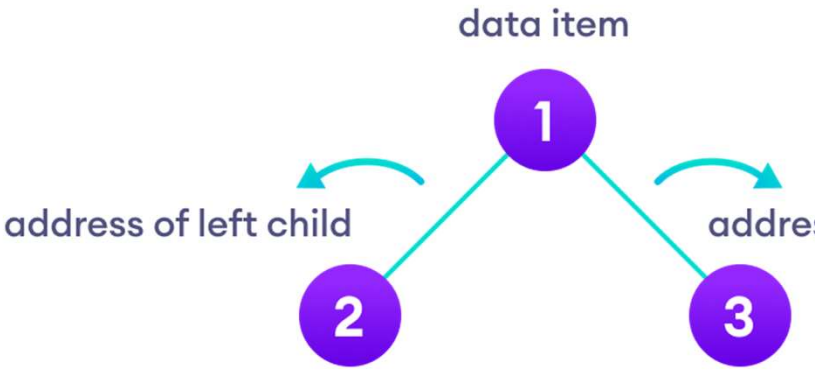
Descendant

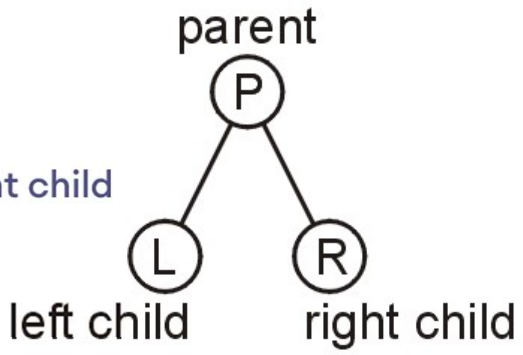
Binary Tree

- A binary tree is a tree data structure in which each node has at most two children.
- The arbitrary number of children in general trees is often unnecessary—many real-life trees are restricted to two branches
  - Expression trees using binary operators
  - An ancestral tree of an individual, parents, grandparents, etc.
  - Phylogenetic trees
  - Lossless encoding algorithms
- A binary tree is a tree data structure in which each parent node can have at most two children. Each node of a binary tree consists of three items:
  - data item
  - address of left child
  - address of right child
- A binary tree is a hierarchical data structure in which each node has at most two children generally referred as left child and right child.
  - Pointer to left subtree
  - Pointer to right subtree
  - Data element

Binary Tree

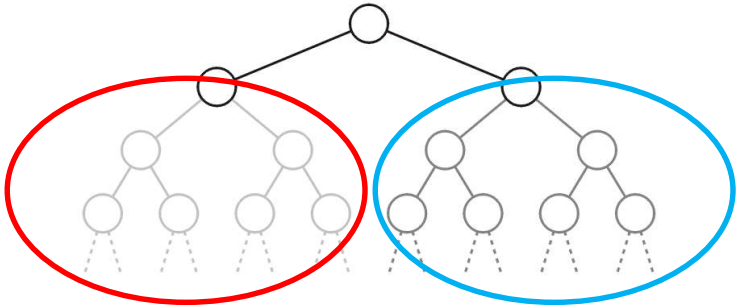
- A binary tree is a restriction where each node has exactly two children:
  - Each child is either empty or another binary tree
  - This restriction allows us to label the children as left and right subtrees

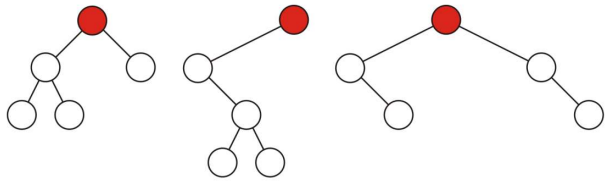




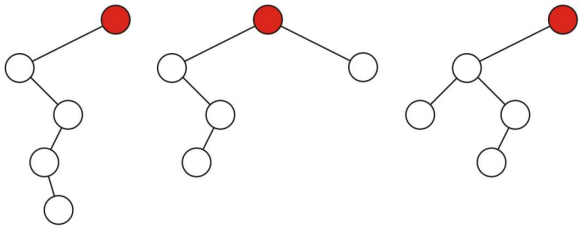
Binary Tree

- We will also refer to the two sub-trees as
  - The left-hand sub-tree,
  - The right-hand sub-tree



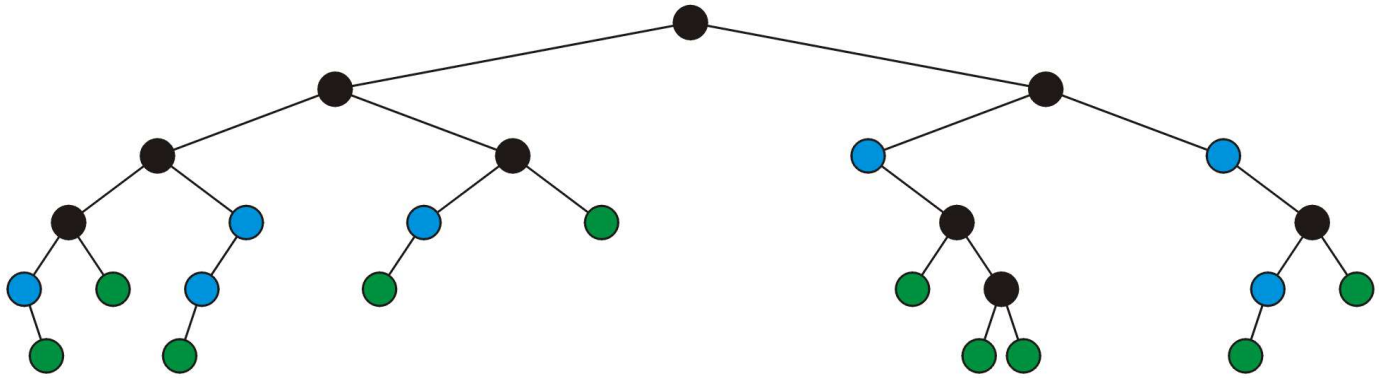


- Sample variations on binary trees with five nodes:



Full node

- A full node is a node where both the left and right sub-trees are non-empty trees



Legend:

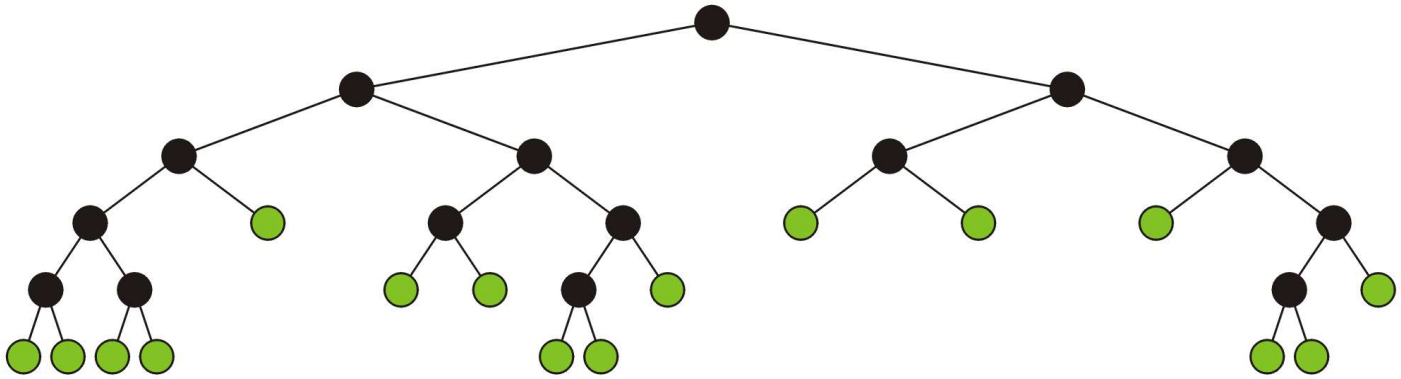
Full nodes

Neither

Leaf nodes

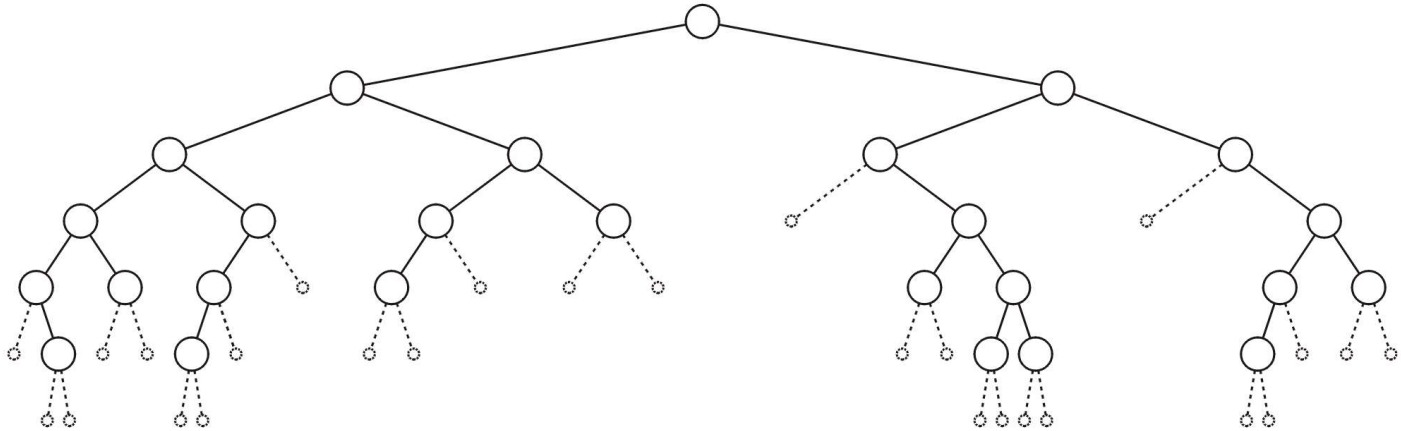
Full node

- A full binary tree is where each node is:
  - A full node, or
  - A leaf node
- These have applications in
  - Expression trees
  - Huffman encoding



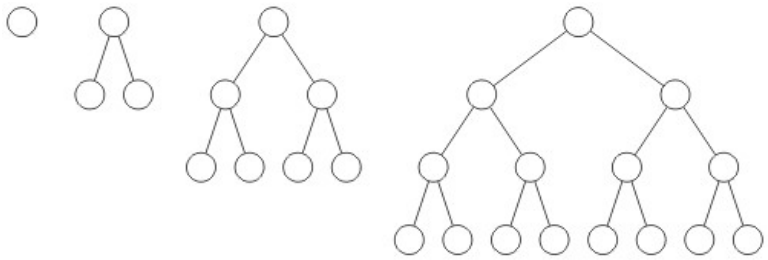
Empty node

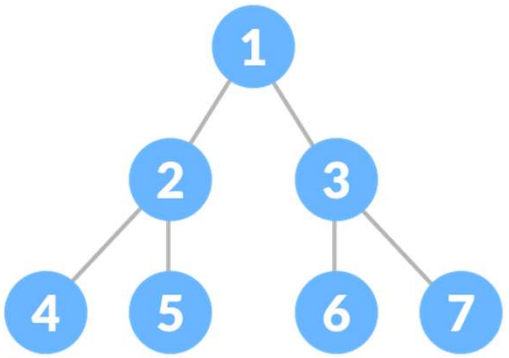
- An empty node or a null sub-tree is any location where a new leaf node could be appended

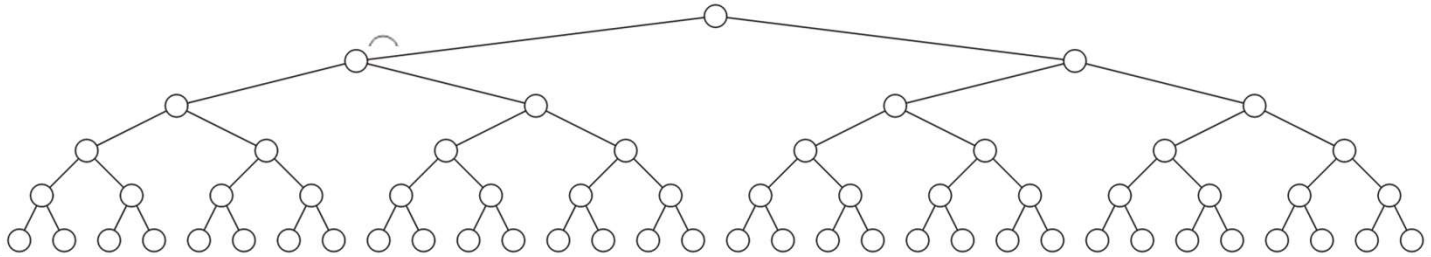


Perfect Binary Tree

- A perfect binary tree is a type of binary tree in which every internal node has exactly two child nodes and all the leaf nodes are at the same level.
- A perfect binary tree of height h is a binary tree where
  - All leaf nodes have the same depth h
  - All other nodes are full



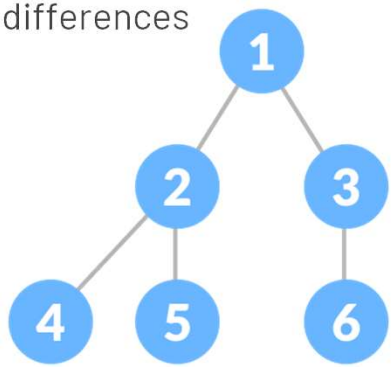




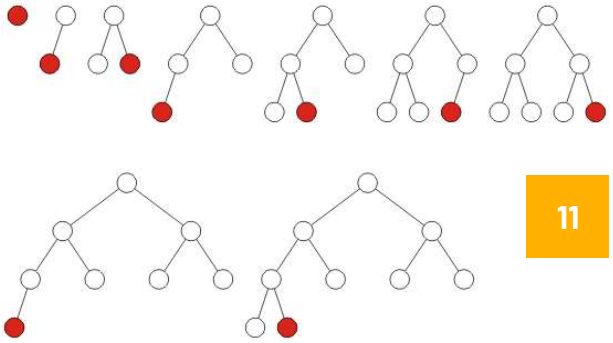
Complete Binary Tree

A complete binary tree is just like a full binary tree, but with two major differences

- Every level must be completely filled
- All the leaf elements must lean towards the left.
- The last leaf element might not have a right sibling i.e.
  a complete binary tree doesn't have to be a full binary tree.

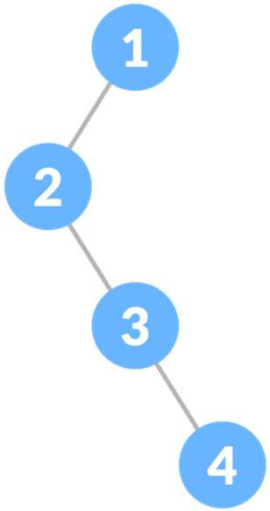


A complete binary tree filled at each depth from left to right:



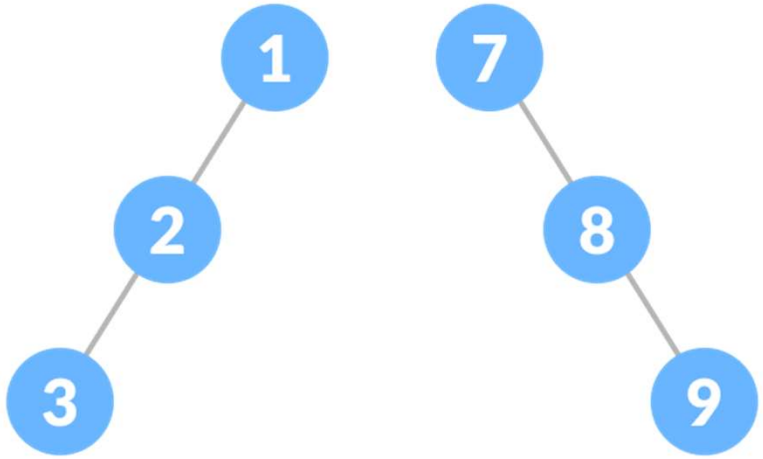
Degenerate or Pathological Tree

- A degenerate or pathological tree is the tree having a single child either left or right.



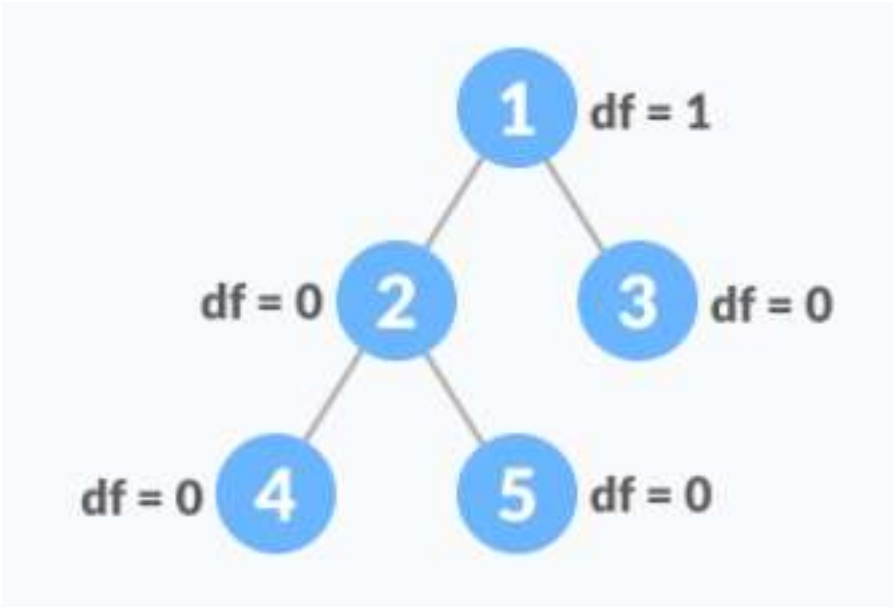
Skewed Binary Tree

- A skewed binary tree is a pathological/degenerate tree in which the tree is either dominated by the left nodes or the right nodes. Thus, there are two types of skewed binary tree: left-skewed binary tree and right-skewed binary tree.



Balanced Binary Tree

- It is a type of binary tree in which the difference between the height of the left and the right subtree for each node is either 0 or 1.



Run Times

- Recall that with linked lists and arrays, some operations would run in **O(n)** time
- The run times of operations on binary trees, we will see, depends on the height of the tree

We will see that:

- The worst is clearly **O(n)**
- Under average conditions, the height is $\Theta\left(\sqrt{n}\right)$
- The best case is **O(ln(n))**

Run Time

- If we can achieve and maintain a height **O(lg(n)),** we will see that many operations can run in Q(lg(n)) we
- Logarithmic time is not significantly worse than constant time:

$$\lg(1000) \approx 10 \quad \text{kB}$$
$$\lg(1\,000\,000) \approx 20 \quad \text{MB}$$
$$\lg(1\,000\,000\,000) \approx 30 \quad \text{GB}$$
$$\lg(1\,000\,000\,000\,000) \approx 40 \quad \text{TB}$$
$$\lg(1000n) \approx 10\,n$$

THERE'S BEEN A LOT OF CONFUSION OVER 1024 vs 1000, KBYTE vs KBIT, AND THE CAPITALIZATION FOR EACH.

HERE, AT LAST, IS A SINGLE, DEFINITIVE STANDARD:

SYMBOL	NAME	SIZE	NOTES
kB	KILOBYTE	1024 BYTES OR 1000 BYTES	1000 BYTES DURING LEAP YEARS, 1024 OTHERWISE
KB	KELLY-BOOTLE STANDARD UNIT	1012 BYTES	COMPROMISE BETWEEN 1000 AND 1024 BYTES
KiB	IMAGINARY KILOBYTE	1024 $\sqrt{\pi}$ BYTES	USED IN QUANTUM COMPUTING
kb	INTEL KILOBYTE	1023.937528 BYTES	CALCULATED ON PENTIUM F.P.U.
Kb	DRIVEMAKER'S KILOBYTE	CURRENTLY 908 BYTES	SHRINKS BY 4 BYTES EACH YEAR FOR MARKETING REASONS
KBa	BAKER'S KILOBYTE	1152 BYTES	9 BITS TO THE BYTE SINCE YOU'RE SUCH A GOOD CUSTOMER