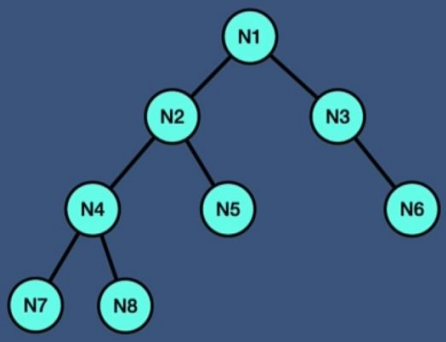
What is a Tree?

A tree is a nonlinear data structure with hierarchical relationships between its elements without having any cycle, it is basically reversed from a real life tree.



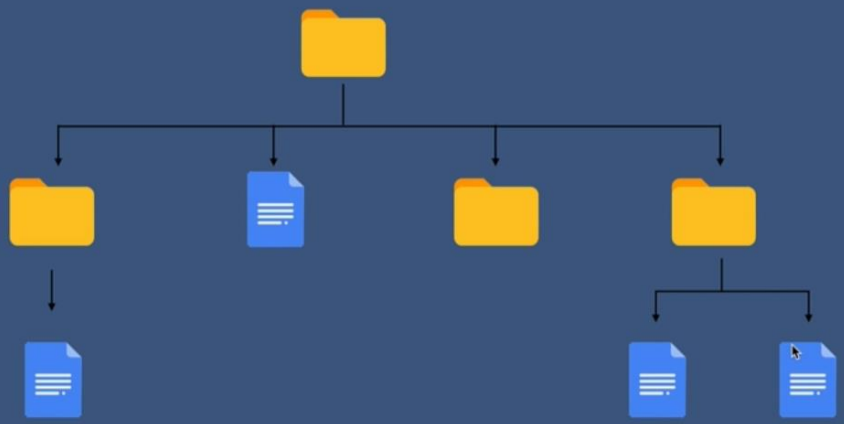
Tree Properties:

- Represent hierarchical data
- Each node has two components : data and a link to its sub category
- Base category and sub categories under it

Why Tree?

- Quicker and Easier access to the data
- Store hierarchical data, like folder structure, organization structure, XML/HTML data.

The file system on a computer

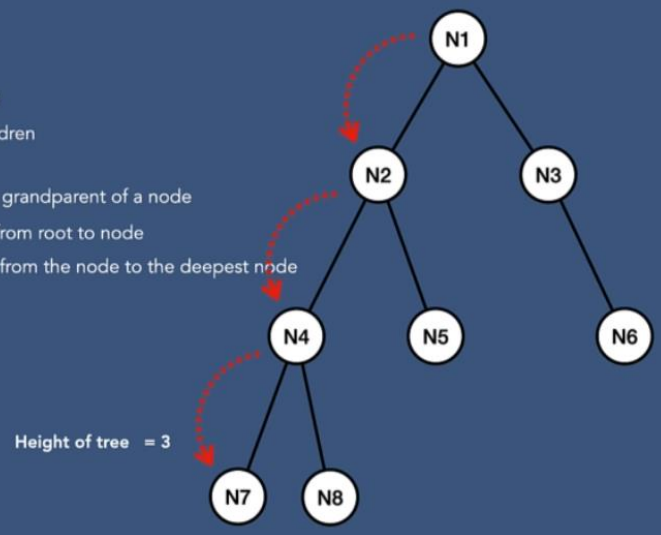


Why Tree?

- Quicker and Easier access to the data
- Store hierarchical data, like folder structure, organization structure, XML/HTML data.
- There are many different types of data structures which performs better in various situations
  - Binary Search Tree, AVL, Red Black Tree, Trie

Tree Terminology

Root : top node without parent
Edge : a link between parent and child
Leaf : a node which does not have children
Sibling : children of same parent
Ancestor : parent, grandparent, great grandparent of a node
Depth of node : a length of the path from root to node
Height of node : a length of the path from the node to the deepest node
Depth of tree : depth of root node
Height of tree : height of root node

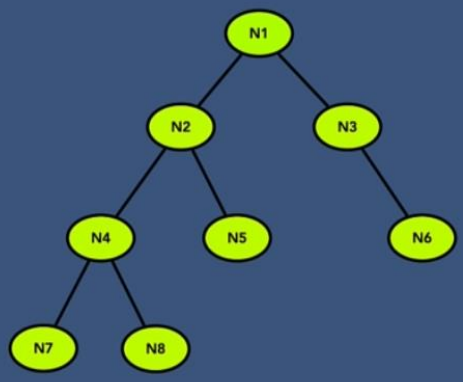


Depth is zero

Binary Tree

- Binary trees are the data structures in which each node has at most two children, often referred to as the left and right children

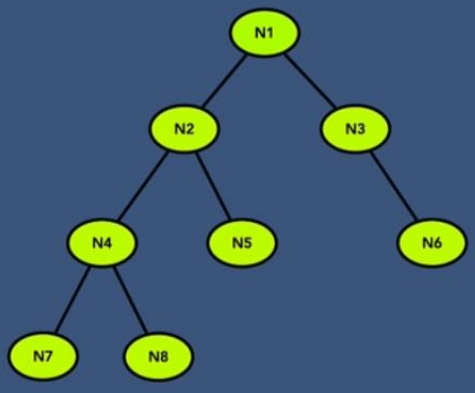
- Binary tree is a family of data structure (BST, Heap tree, AVL, red black trees, Syntax tree)

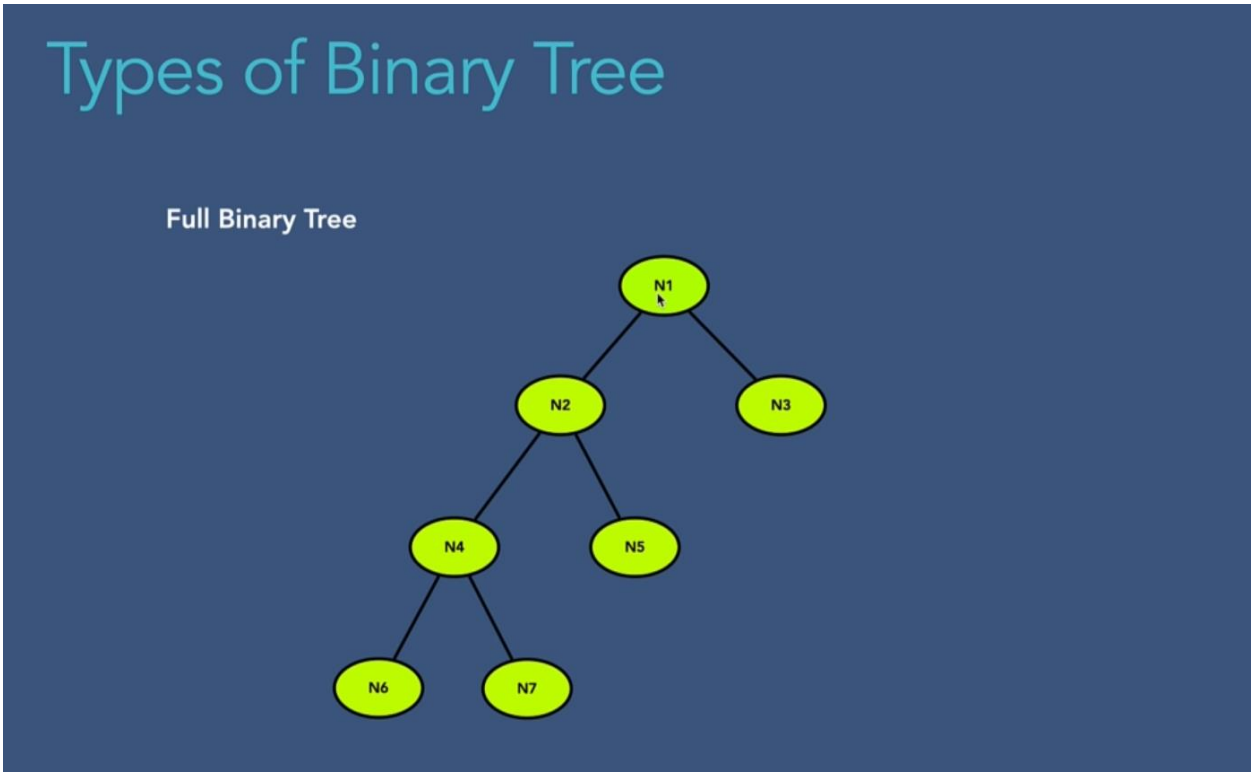


Why Binary Tree?

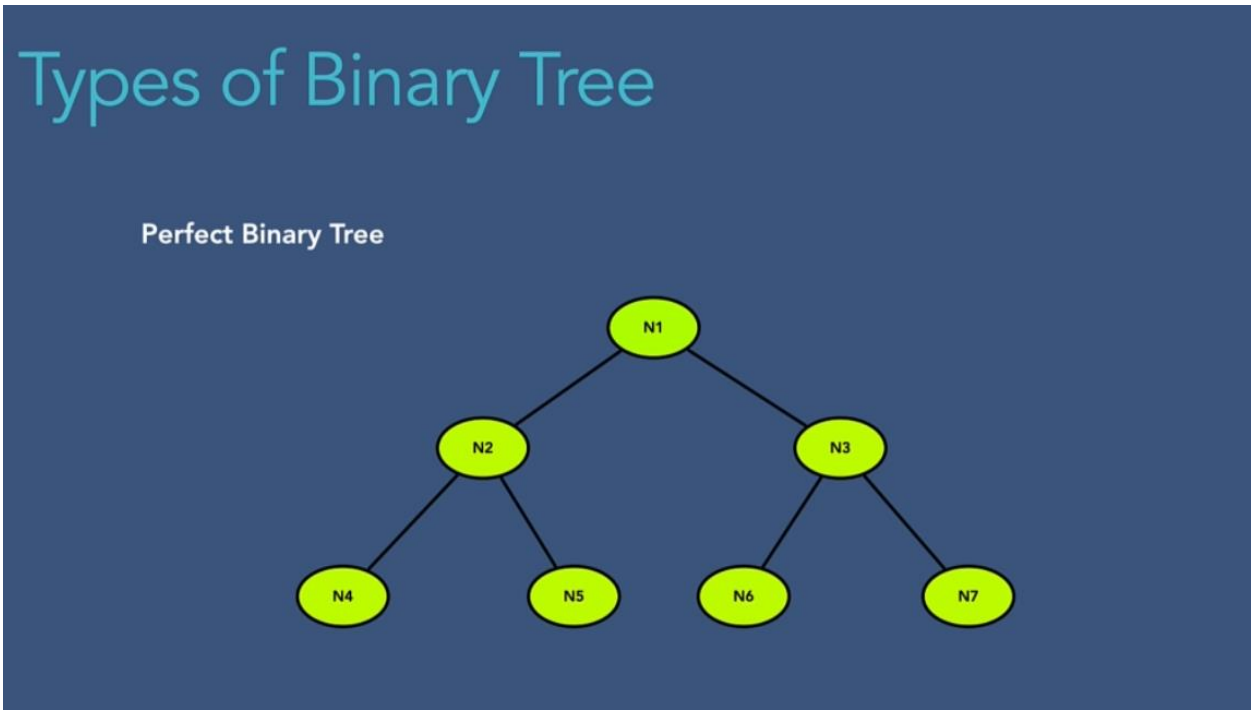
- Binary trees are a prerequisite for mode advanced trees like BST, AVL, Red Black Trees

- Huffman coding problem , heap priority problem and expression parsing problems can be solved efficiently using binary trees,

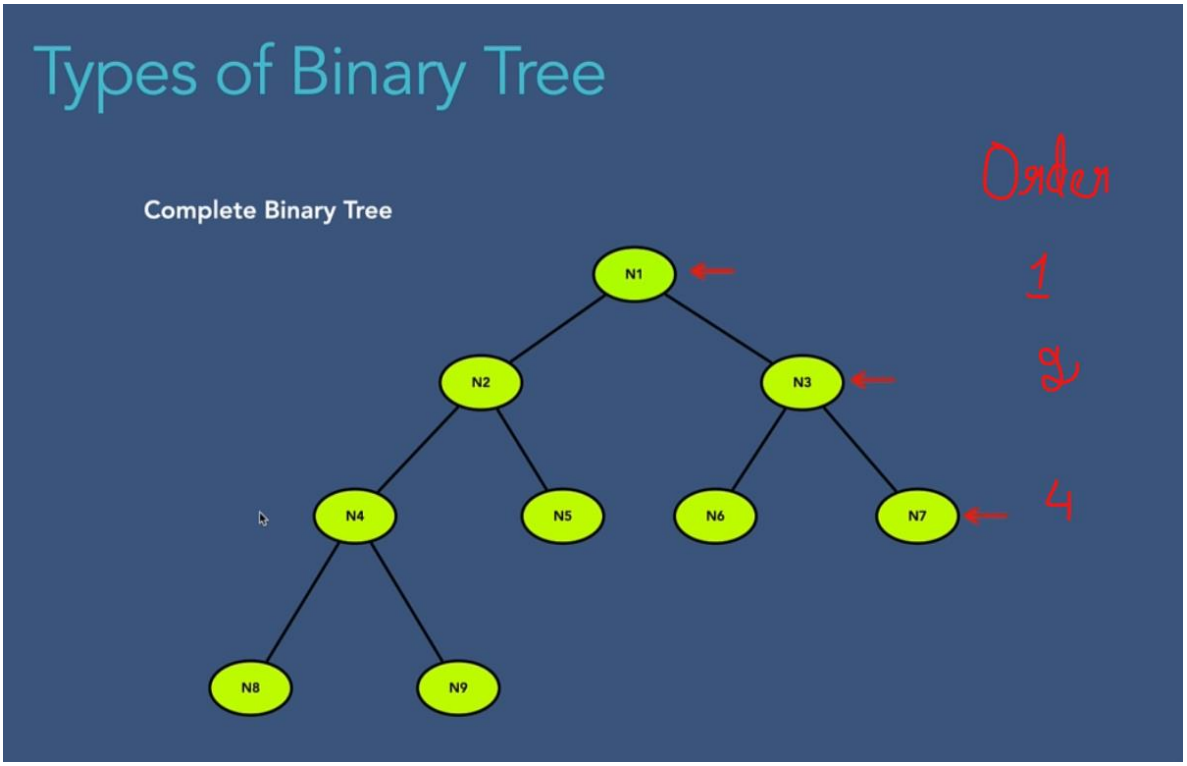




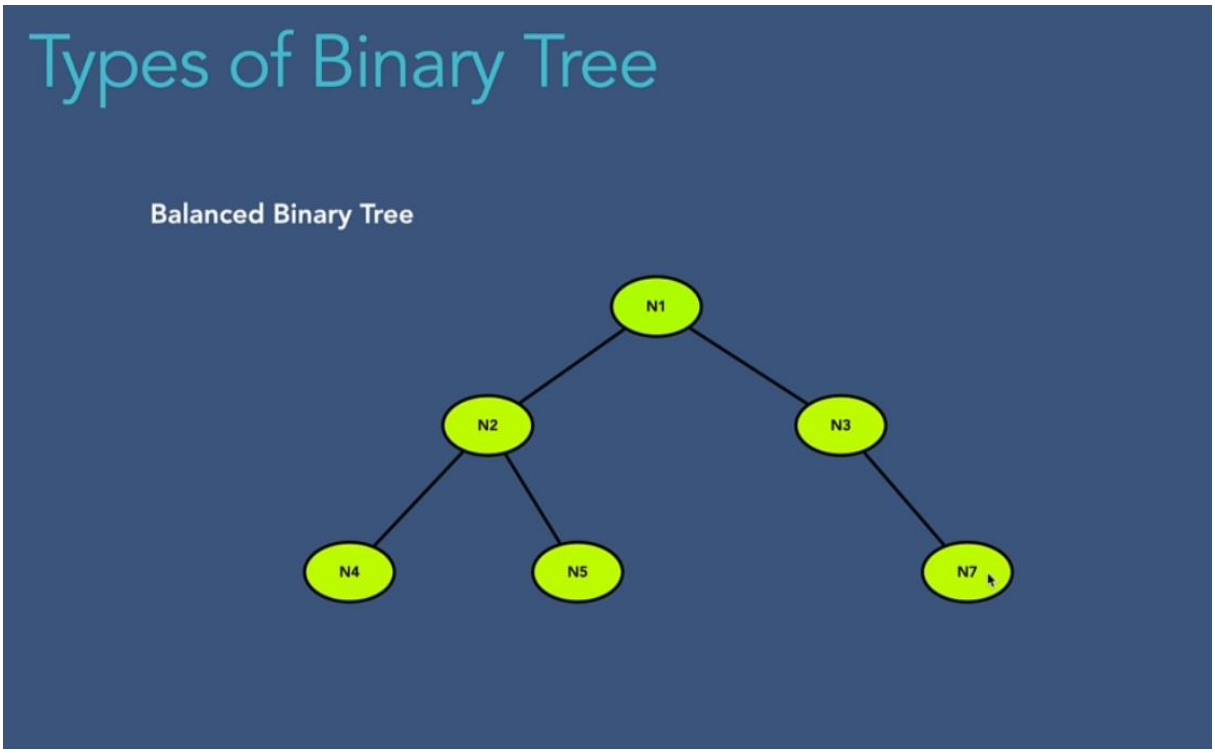
Full Binary Tree → can have zero or two children and cannot have 1 child.



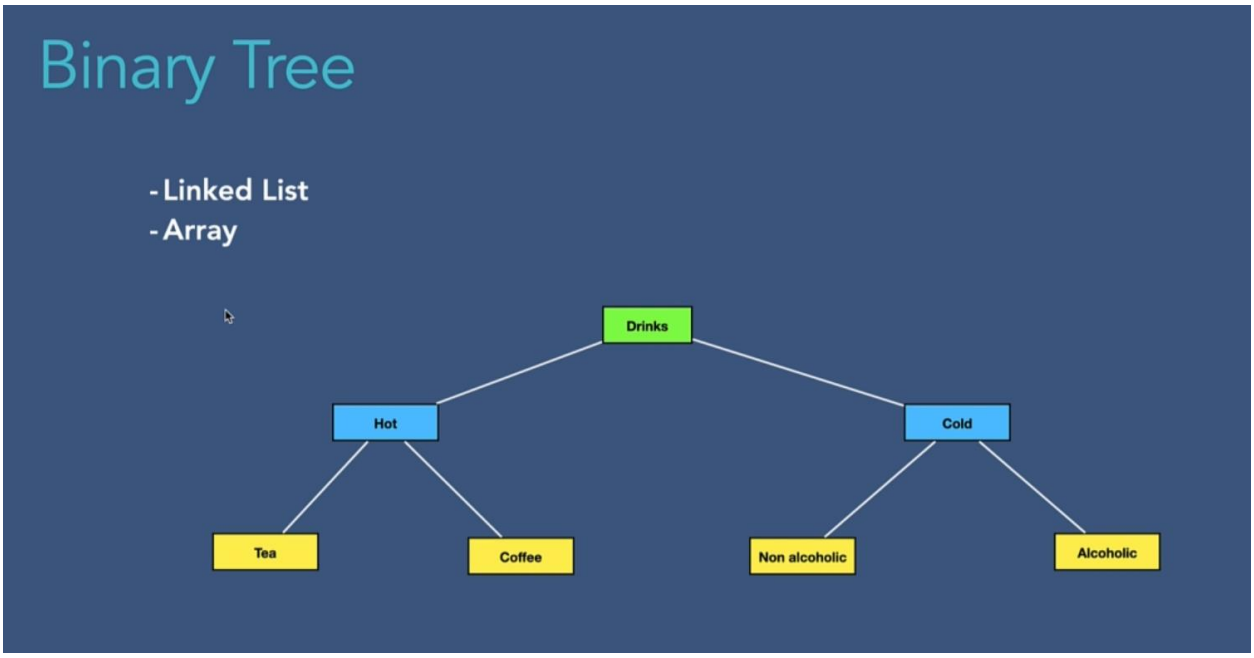
Perfect Binary Tree → all non-leaf nodes have two children and are at the same level



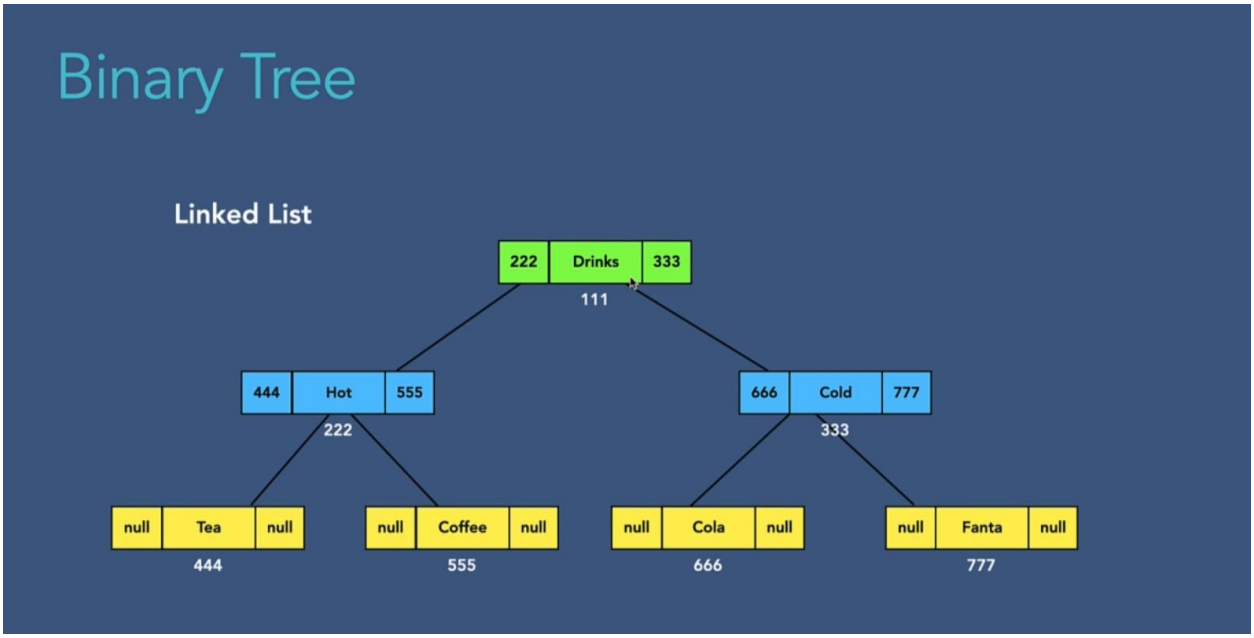
Complete Binary Tree → all the levels are completely filled except the last level and the last level nodes should be as left as possible



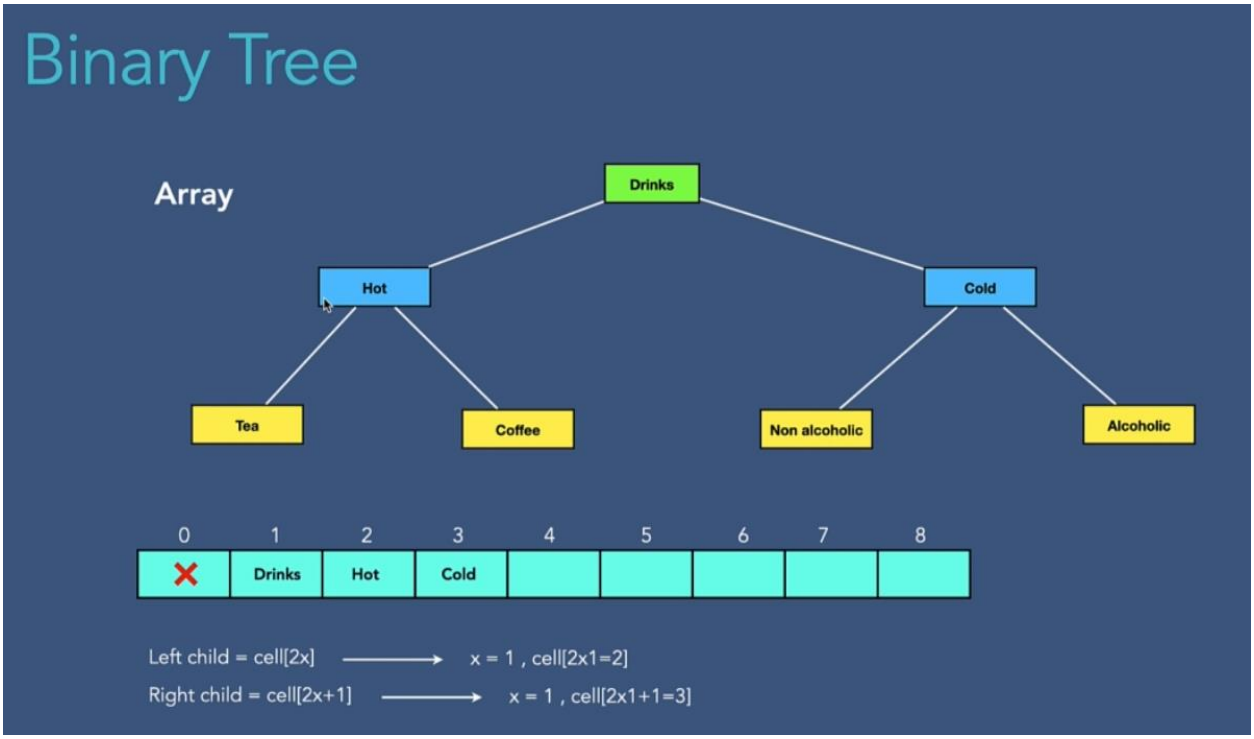
Balanced Binary Tree → All the leaf nodes are at same distant from each other

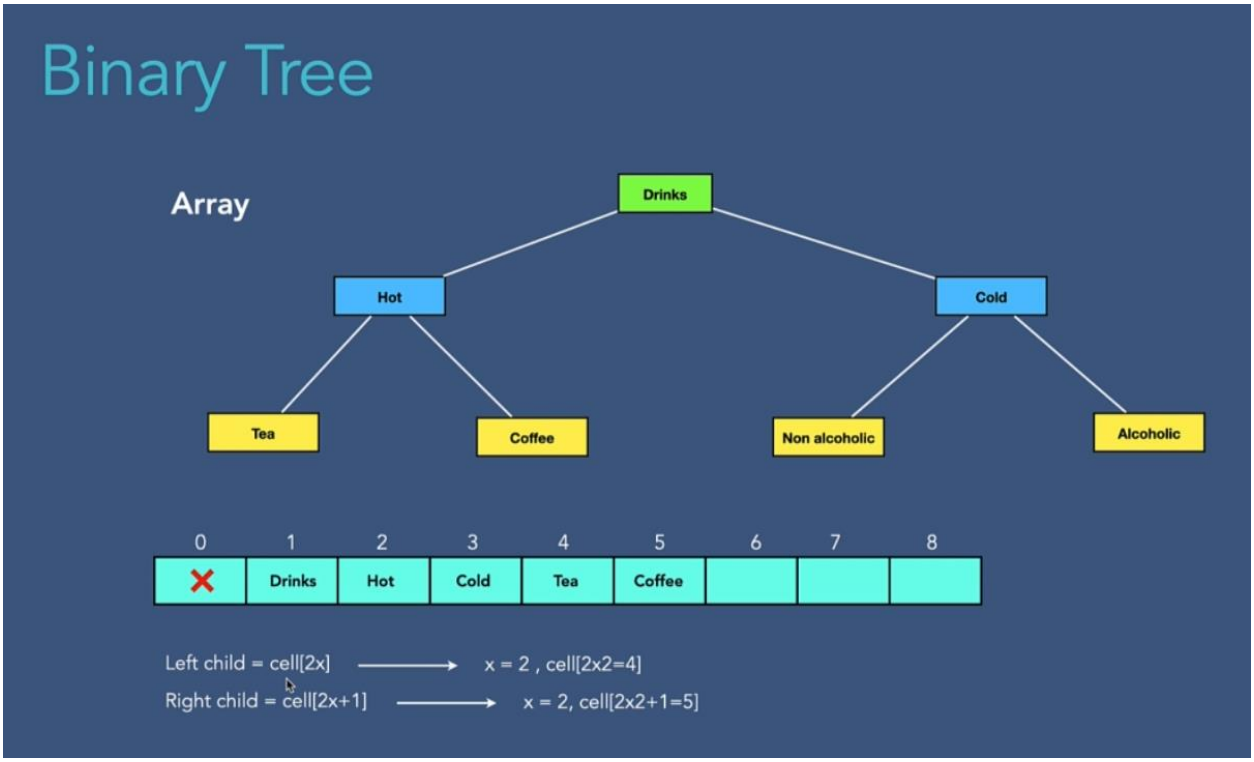


Binary tree implementation using Linked List:



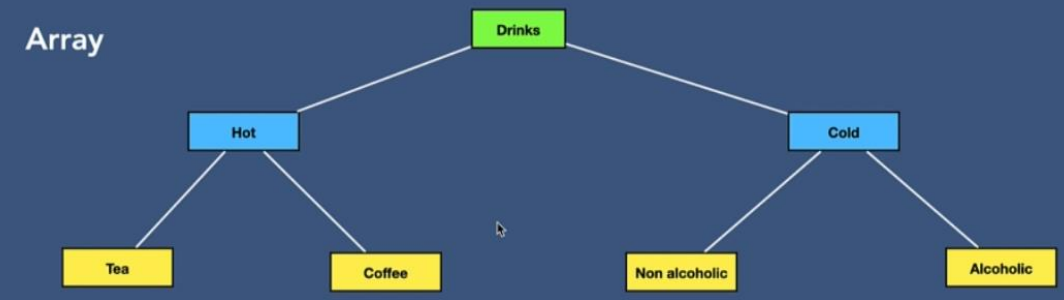
Binary Tree Implementation using Arrays:





Binary Tree

Array



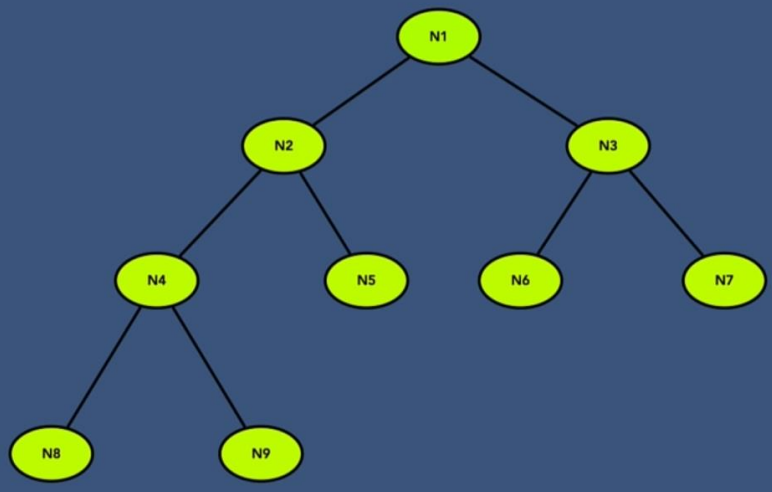
0	1	2	3	4	5	6	7	8
✗	Drinks	Hot	Cold	Tea	Coffee	Non alcoholic	Alcoholic	

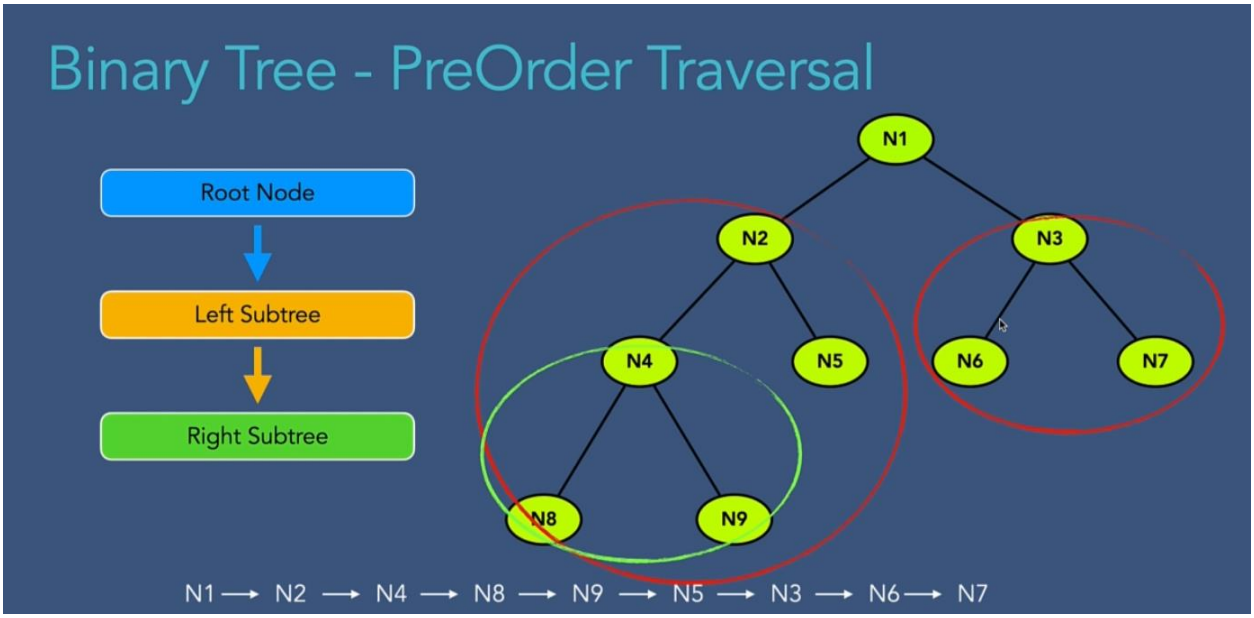
Left child = cell[2x] ⟶ x = 3 , cell[2x3=6]

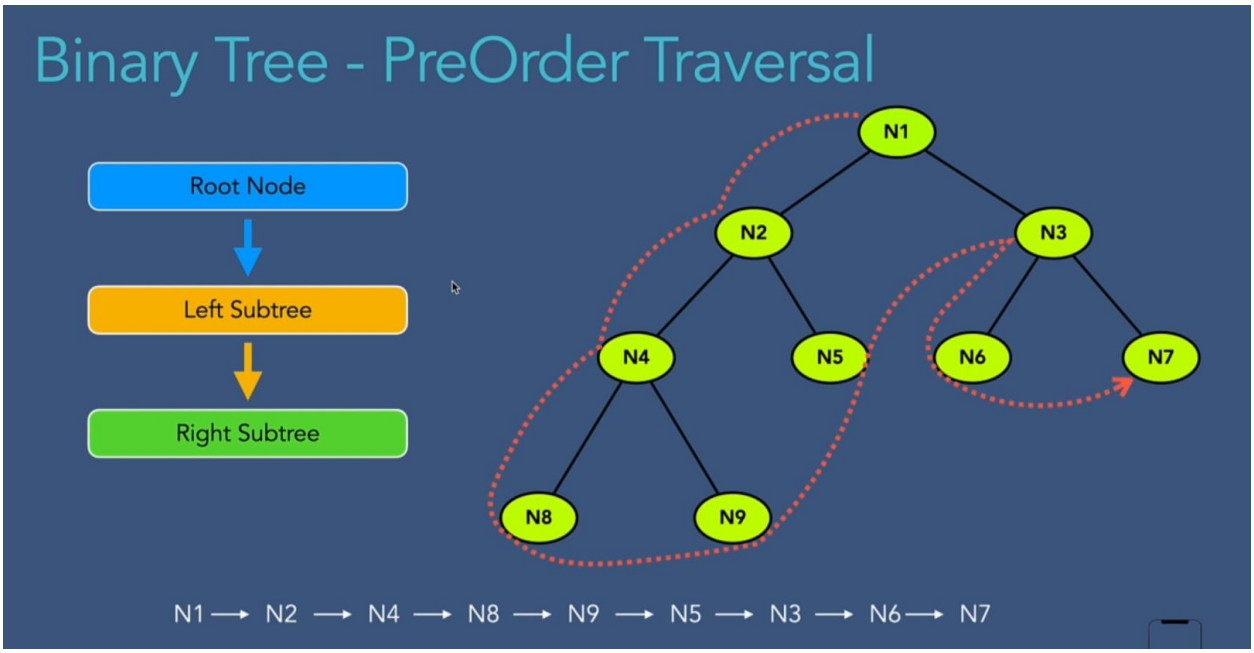
Right child = cell[2x+1] ⟶ x = 3, cell[2x3+1=7]

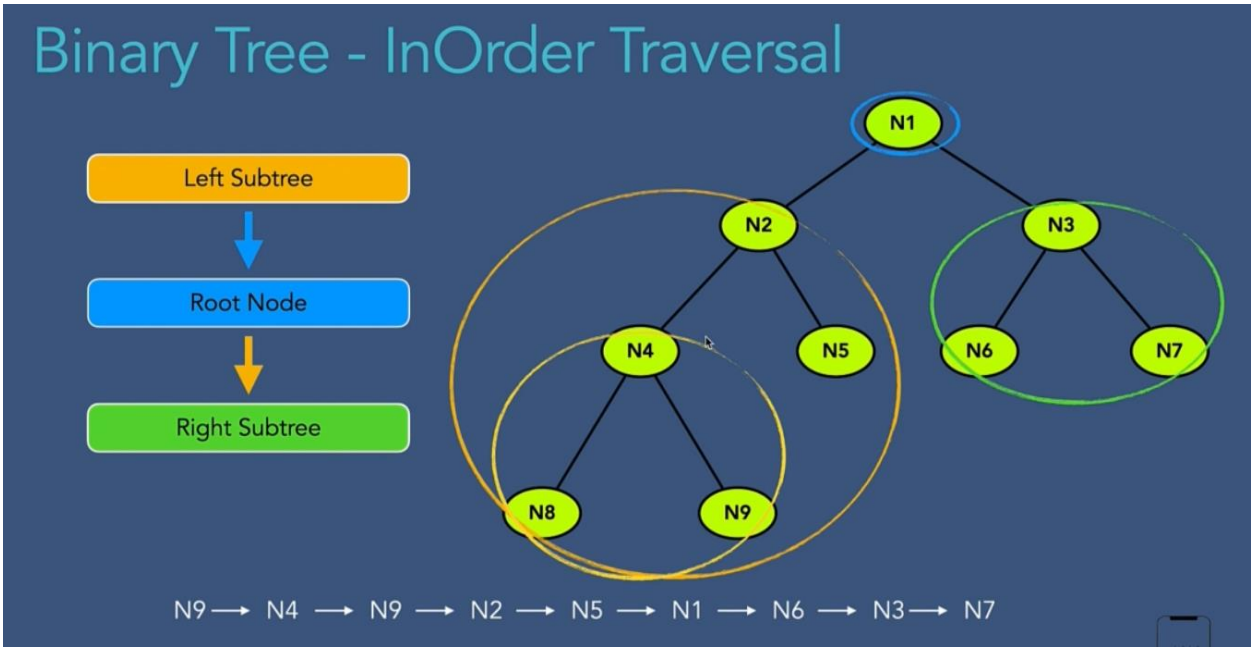
Binary Tree using Linked List

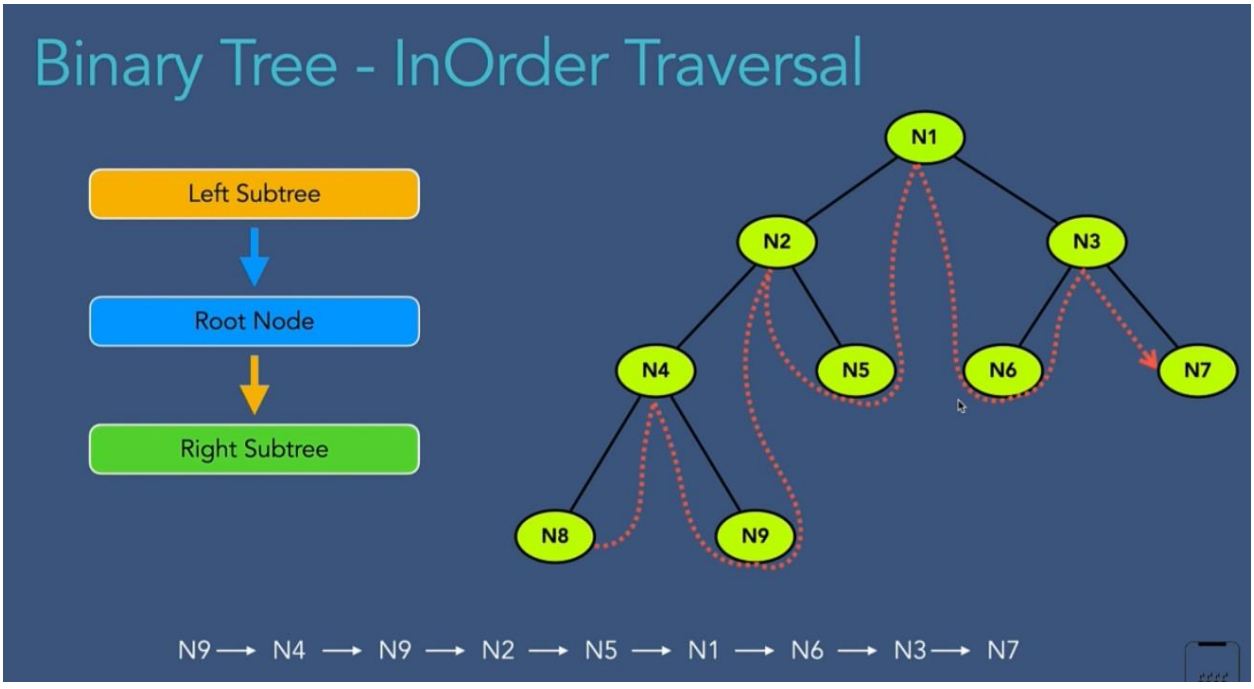
- Creation of Tree
- Insertion of a node
- Deletion of a node
- Search for a value
- Traverse all nodes
- Deletion of tree

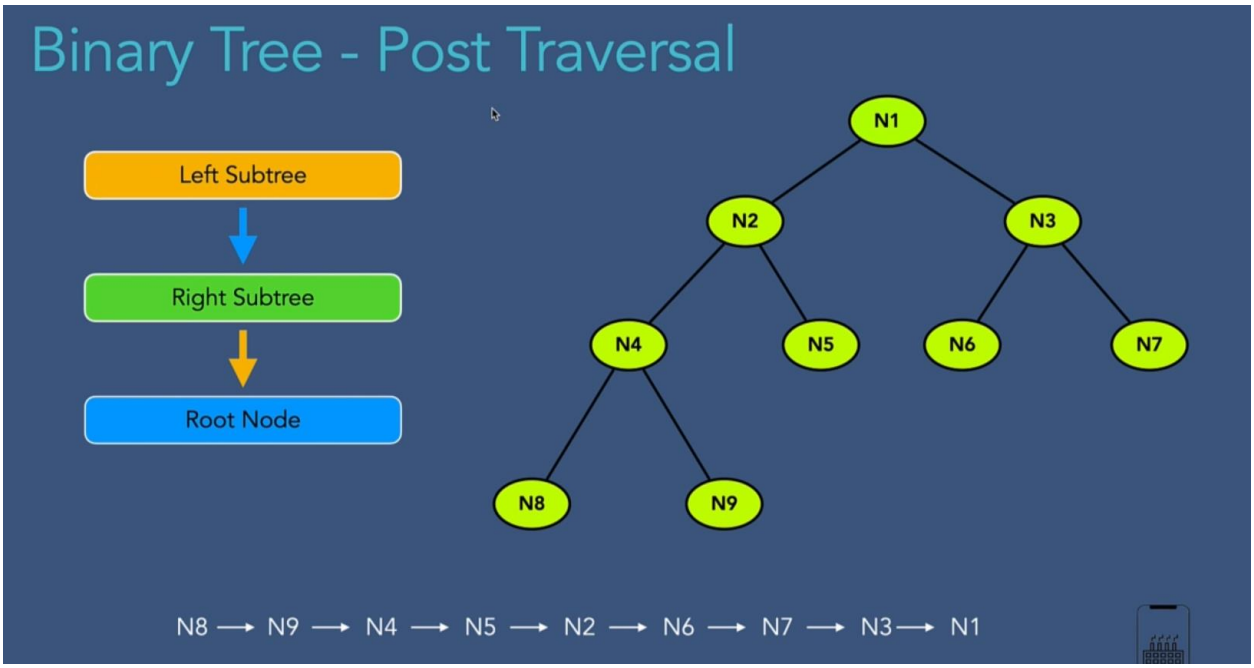


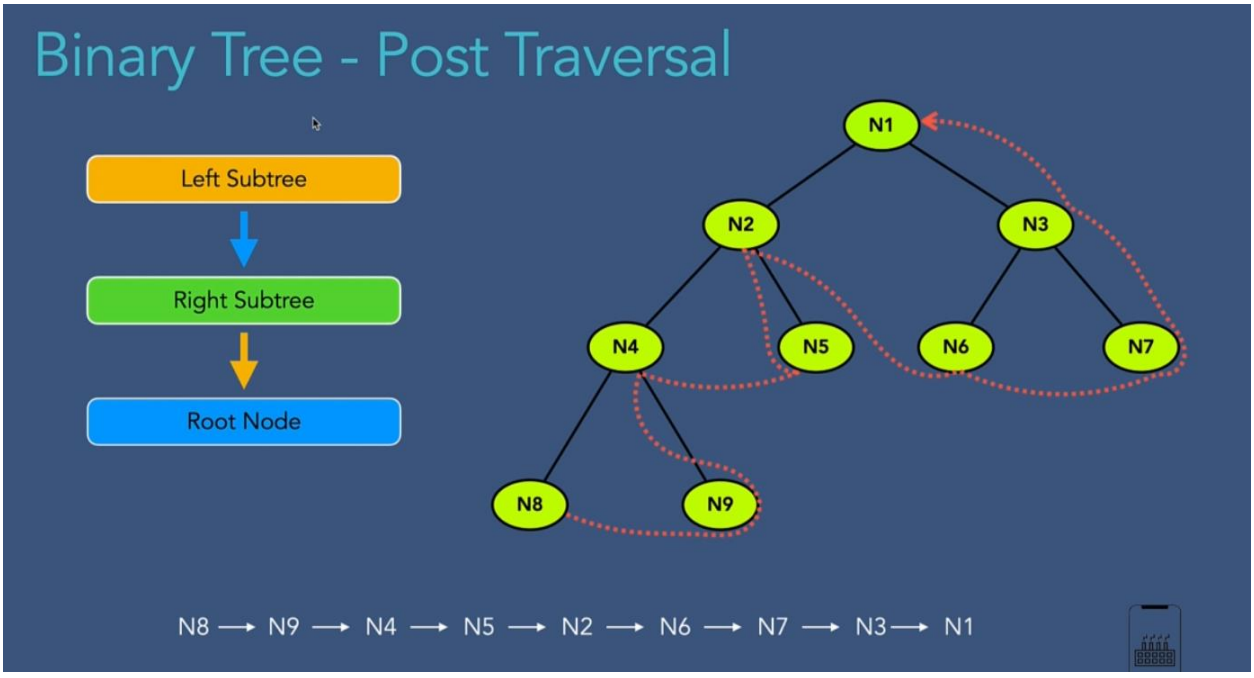

Binary Tree - PreOrder Traversal
Root Node
Left Subtree
Right Subtree
N1
N2
N3
N4
N5
N6
N7
N8
N9
N1 → N2 → N4 → N8 → N9 → N5 → N3 → N6 → N7


Binary Tree - PreOrder Traversal
Root Node
Left Subtree
Right Subtree
N1
N2
N3
N4
N5
N6
N7
N8
N9
N1 → N2 → N4 → N8 → N9 → N5 → N3 → N6 → N7

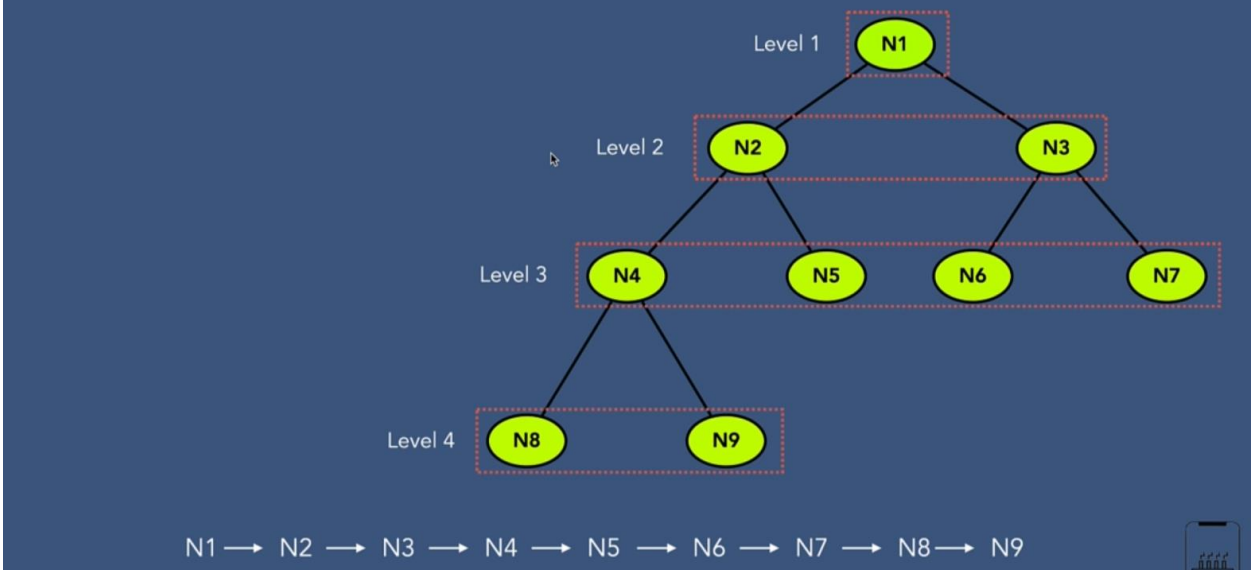

Binary Tree - InOrder Traversal
Left Subtree
Root Node
Right Subtree
N1
N2
N3
N4
N5
N6
N7
N8
N9
N9 → N4 → N9 → N2 → N5 → N1 → N6 → N3 → N7


Binary Tree - InOrder Traversal
Left Subtree
Root Node
Right Subtree
N1
N2
N3
N4
N5
N6
N7
N8
N9
N9 → N4 → N9 → N2 → N5 → N1 → N6 → N3 → N7


Binary Tree - Post Traversal
Left Subtree
Right Subtree
Root Node
N1
N2
N3
N4
N5
N6
N7
N8
N9
N8 → N9 → N4 → N5 → N2 → N6 → N7 → N3 → N1

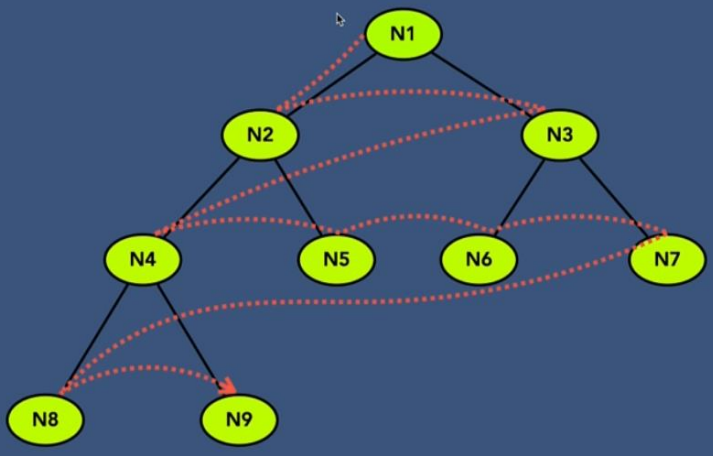

Binary Tree - Post Traversal
Left Subtree
Right Subtree
Root Node
N1
N2
N3
N4
N5
N6
N7
N8
N9
N8 → N9 → N4 → N5 → N2 → N6 → N7 → N3 → N1

Binary Tree - LevelOrder Traversal

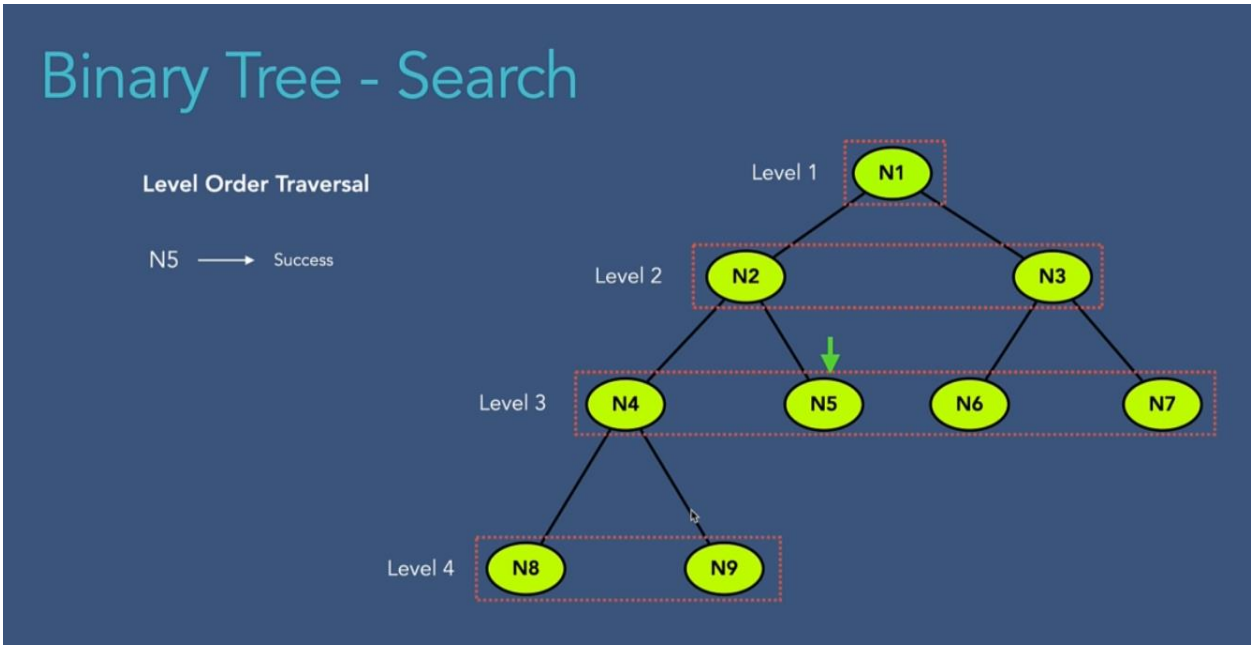


N1 → N2 → N3 → N4 → N5 → N6 → N7 → N8 → N9

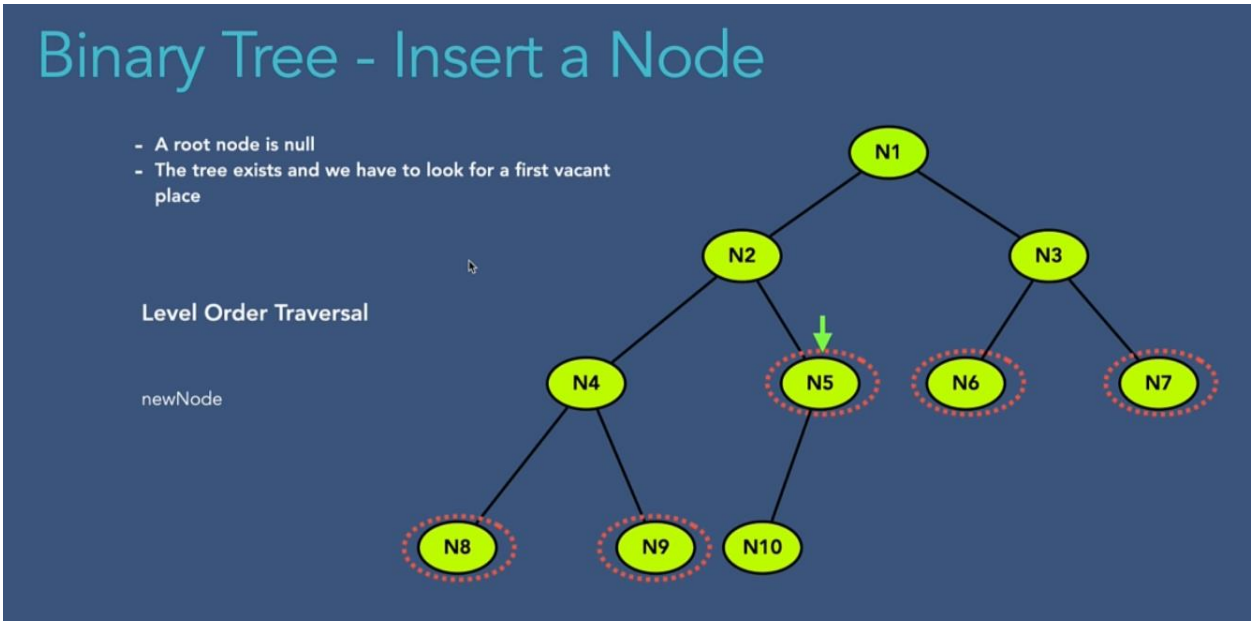
Binary Tree - LevelOrder Traversal



N1 → N2 → N3 → N4 → N5 → N6 → N7 → N9 → N10



Search is done using LEVEL ORDER TRAVERSAL



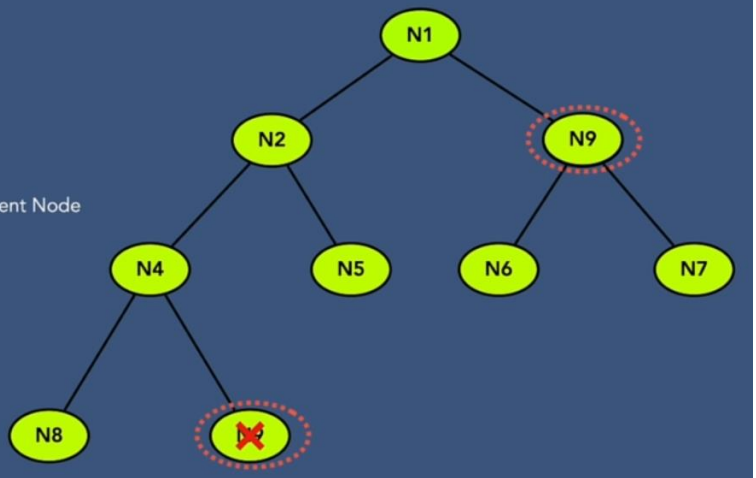
Insertion is done using LEVEL ORDER TRAVERSAL

Binary Tree - Delete a Node

Level Order Traversal

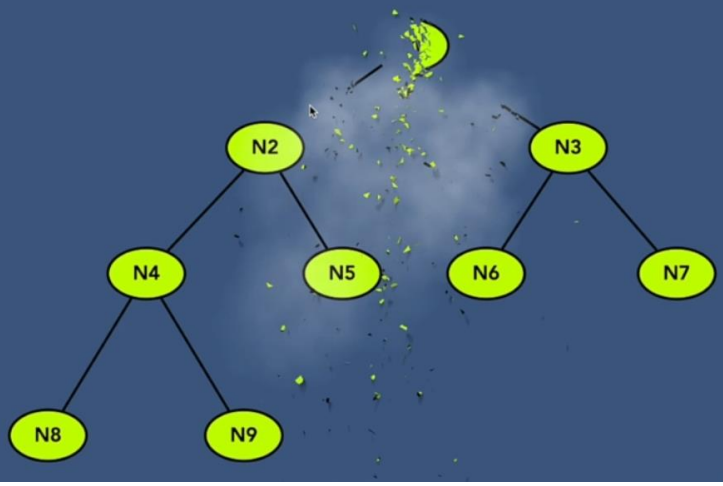
N3

Step 1 - Find the Node
Step 2 - Find Deepest Node
Step 3 - Set Deepest Node's value to Current Node
Step 4 - Delete Deepest Node



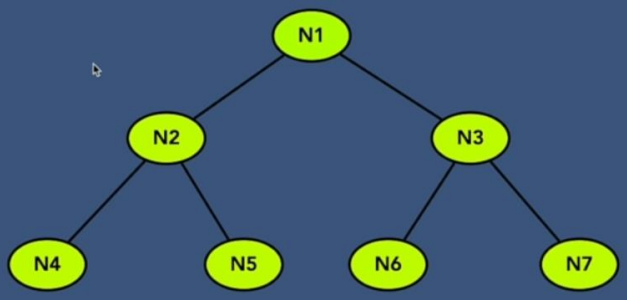
Binary Tree - Delete Binary Tree

rootNode = Null

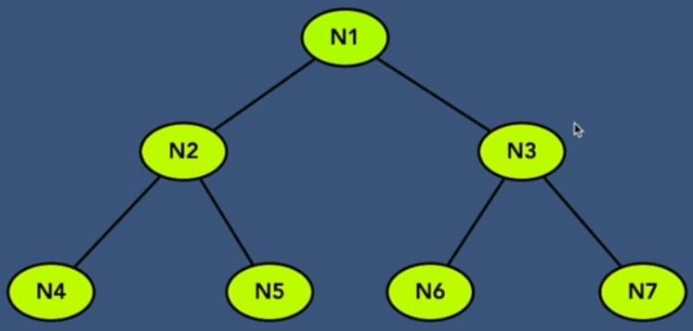


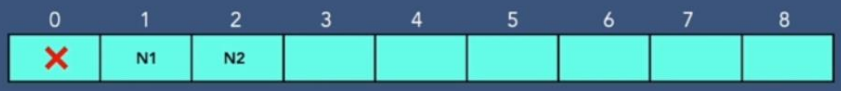
Binary Tree using Array

- Creation of Tree
- Insertion of a node
- Deletion of a node
- Search for a value
- Traverse all nodes
- Deletion of tree



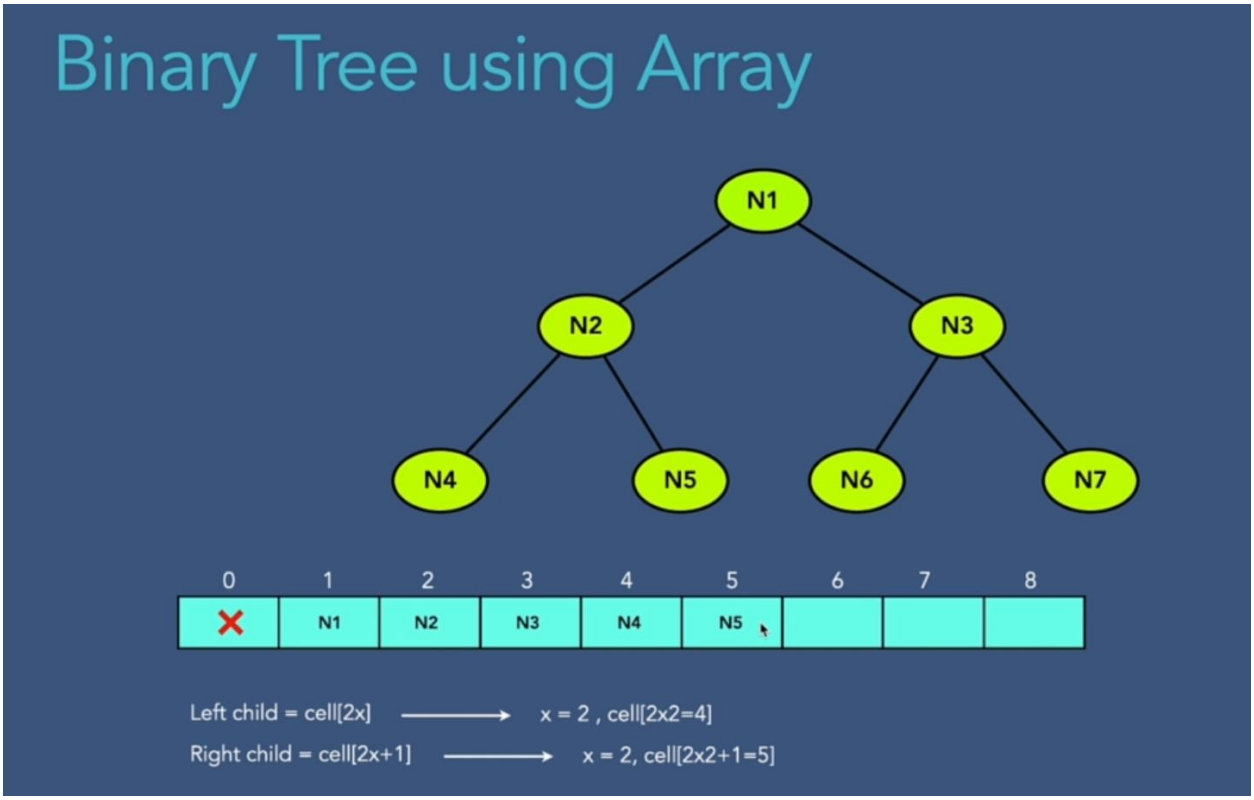
Binary Tree using Array



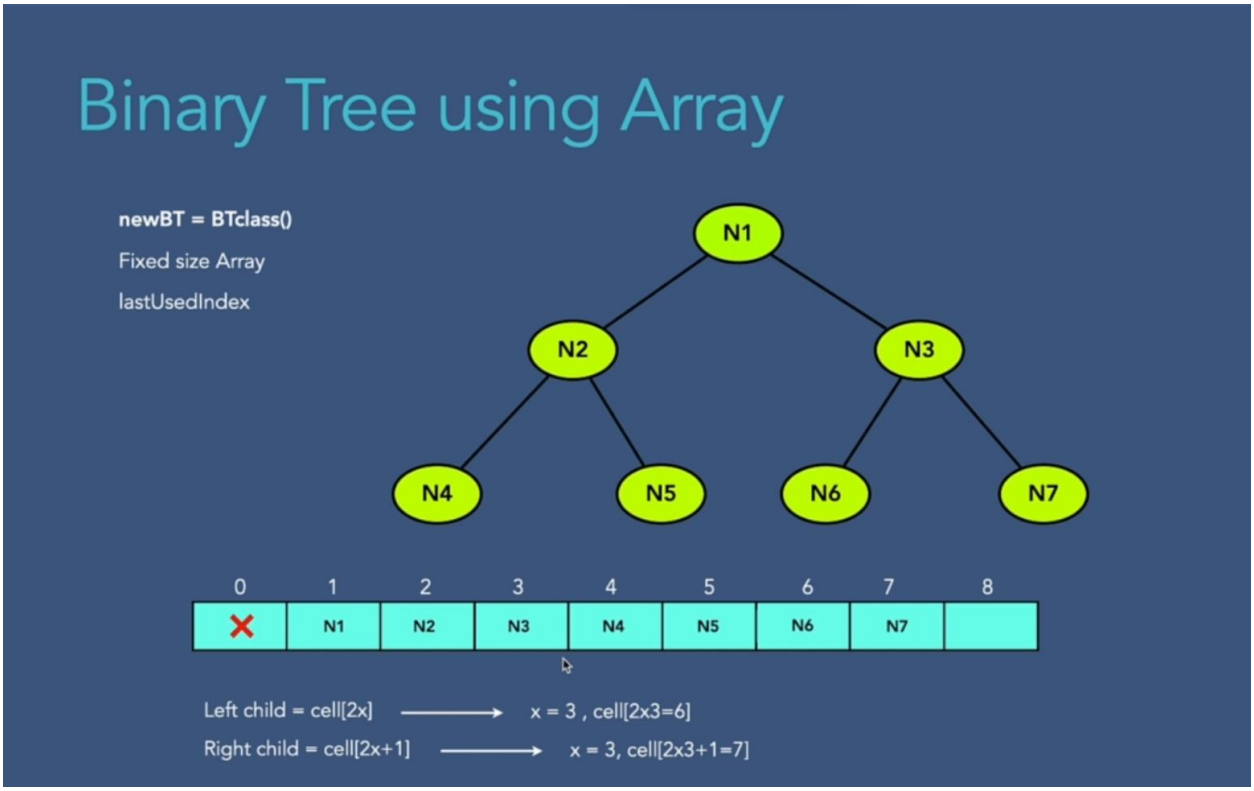


Left child = cell[2x] ⟶ x = 1 , cell[2x1=2]

Right child = cell[2x+1] ⟶ x = 1 , cell[2x1+1=3]



Use the parent node as x value to calculate the position of the child nodes in the array



Binary Tree (Array) - Insert a Node

- The Binary Tree is full
- We have to look for a first vacant place

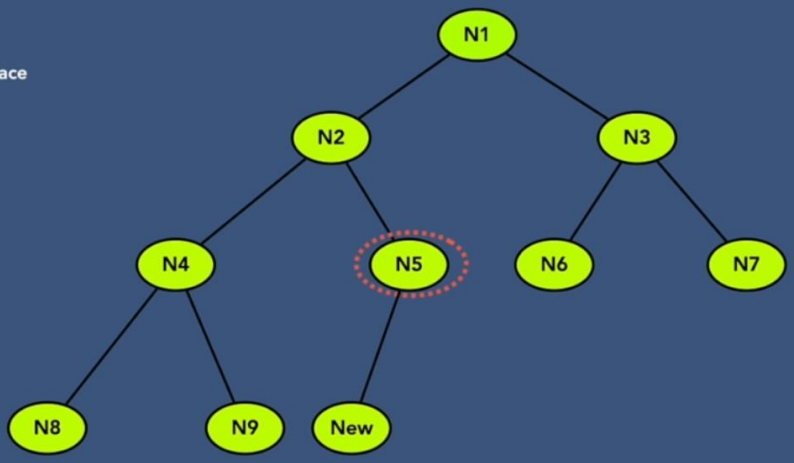
lastUsedIndex = 9

newNode

Index = 10

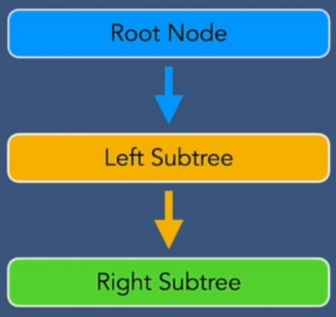
Left child = cell[2x]

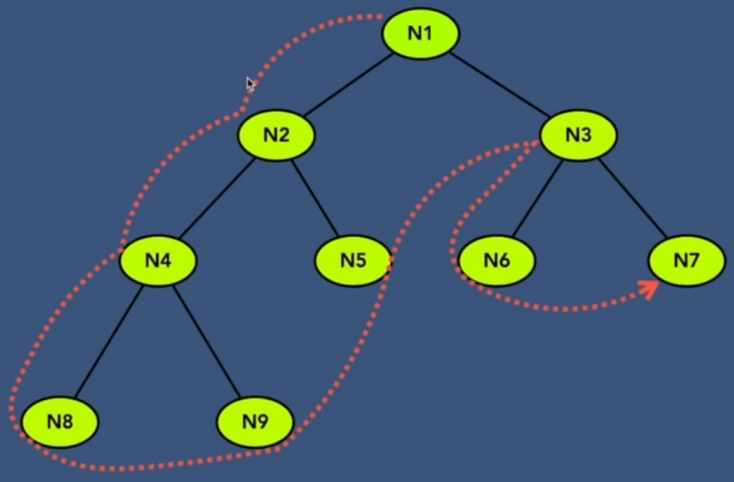
indexOfParent = 10/2 = 5

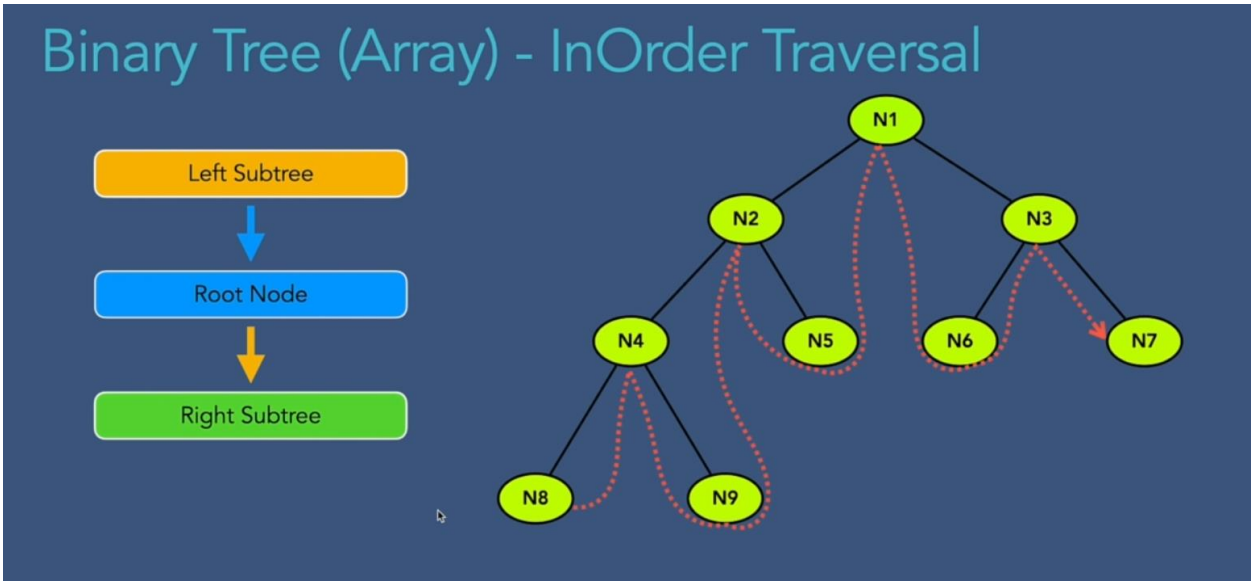


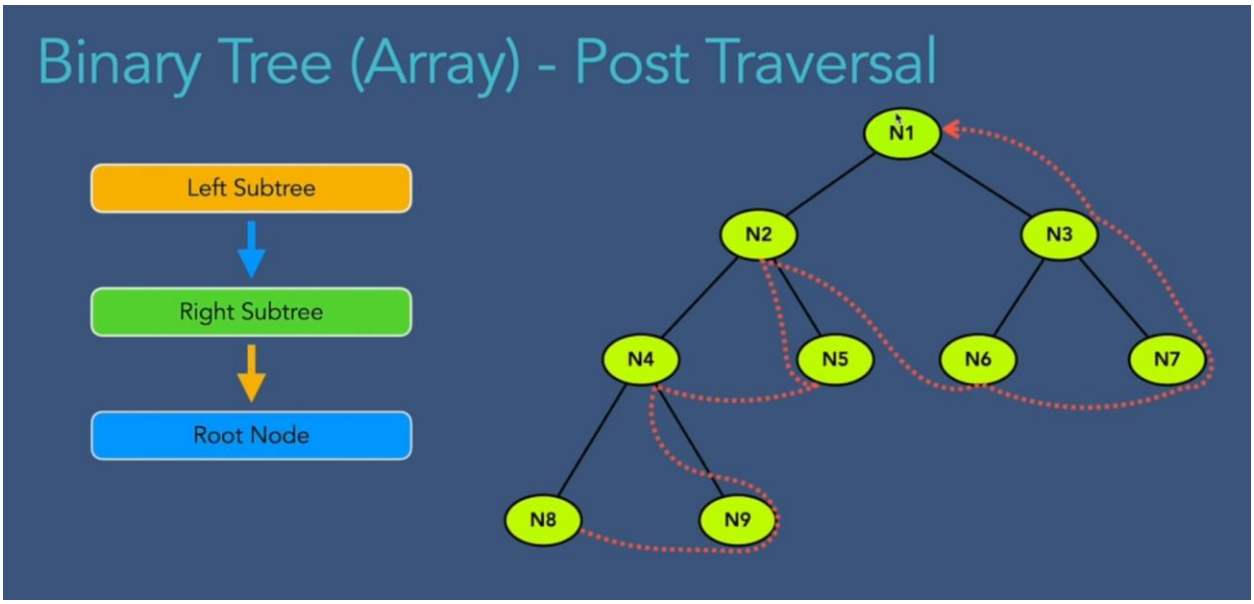
0	1	2	3	4	5	6	7	8	9	10	11
✗	N1	N2	N3	N4	N5	N6	N7	N8	N9	New	

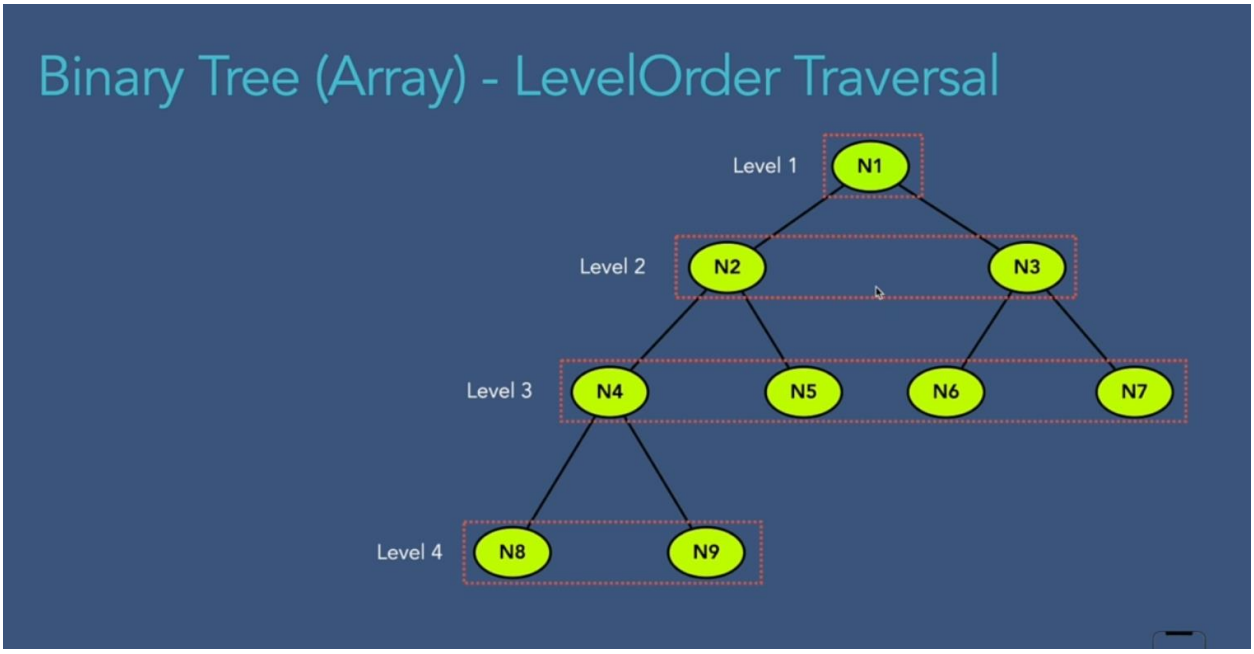
Binary Tree (Array) - PreOrder Traversal

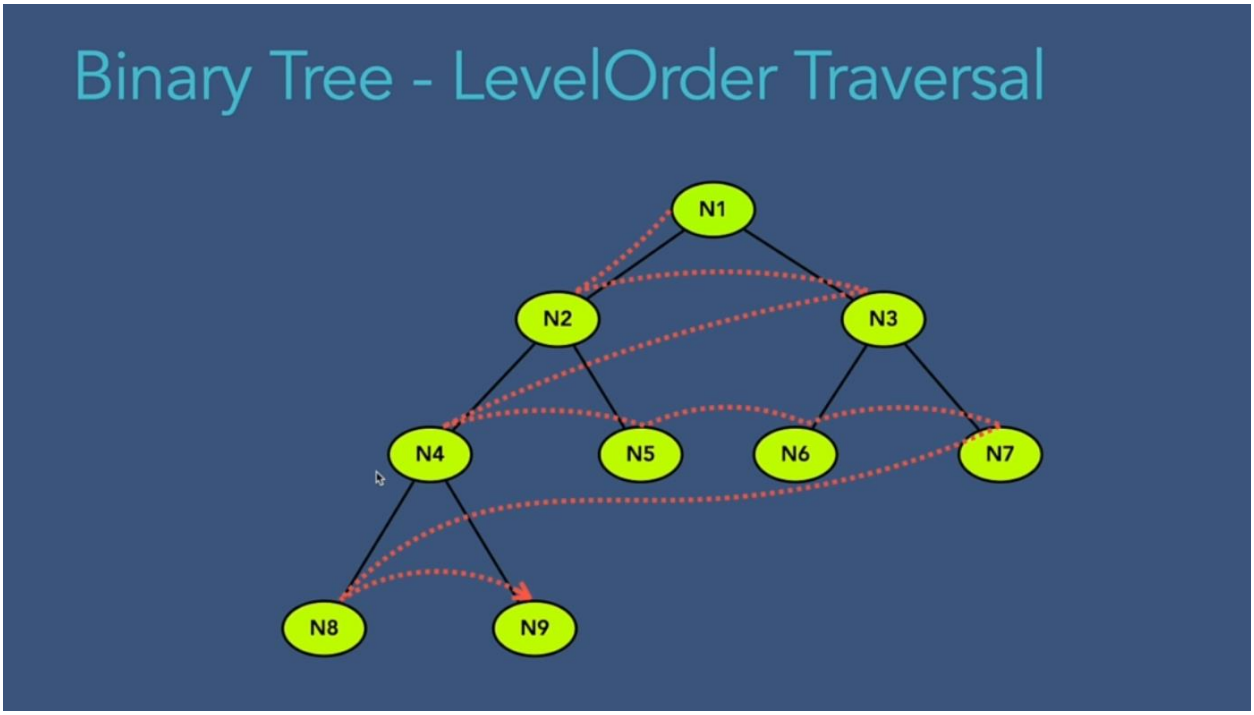


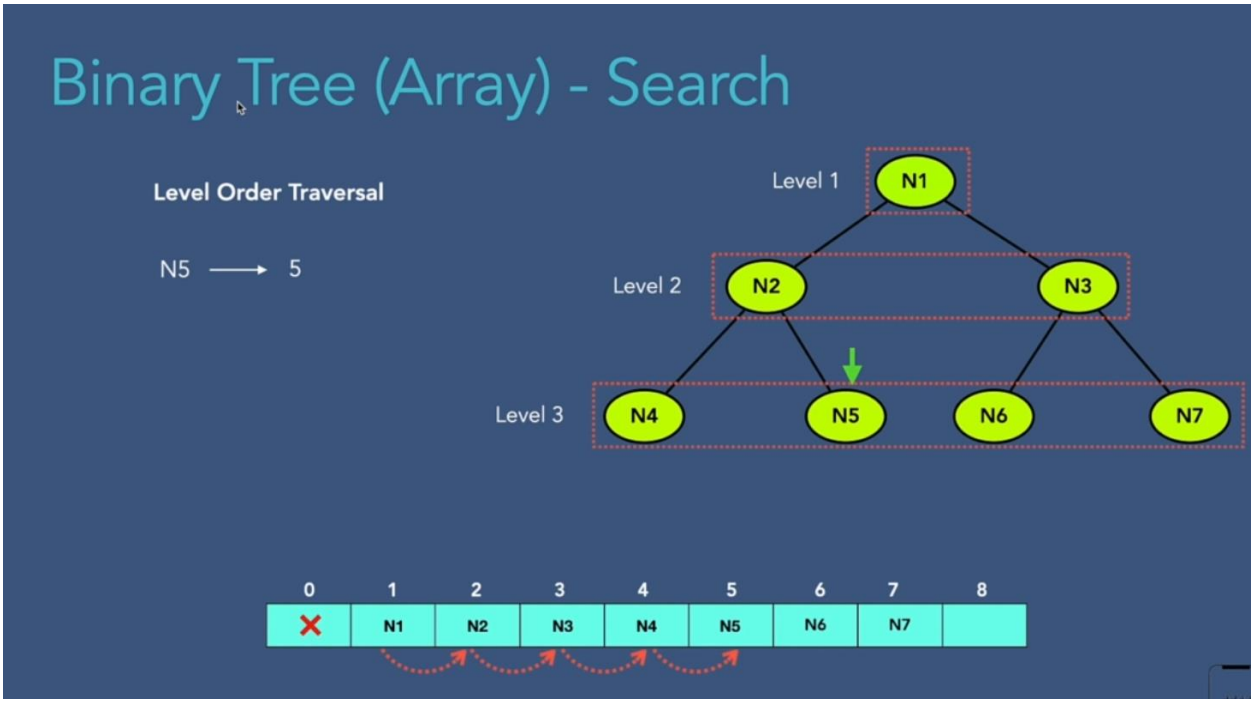


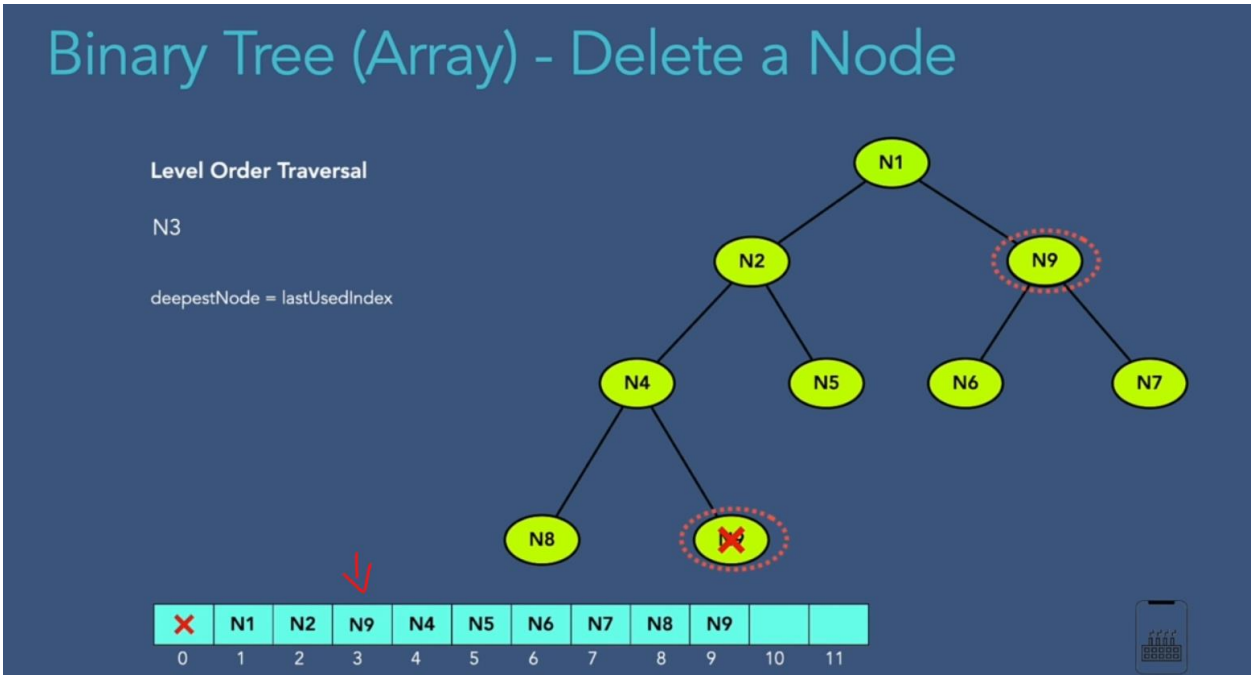

Binary Tree (Array) - InOrder Traversal
Left Subtree
Root Node
Right Subtree
N1
N2
N3
N4
N5
N6
N7
N8
N9


Binary Tree (Array) - Post Traversal
Left Subtree
Right Subtree
Root Node
N1
N2
N3
N4
N5
N6
N7
N8
N9


Binary Tree (Array) - LevelOrder Traversal
Level 1
N1
Level 2
N2
N3
Level 3
N4
N5
N6
N7
Level 4
N8
N9


Binary Tree - LevelOrder Traversal
N1
N2
N3
N4
N5
N6
N7
N8
N9





To delete a node first we find the deepest node and change the value of the deepest node to the node to be deleted and the delete the deepest node.

Binary Tree (Array) - Delete Binary Tree

arr = Null

Binary Tree (Array vs Linked List)

	Array		Linked List	
	Time complexity	Space complexity	Time complexity	Space complexity
Create Binary Tree	O(1)	O(n)	O(1)	O(1)
Insert a node to Binary Tree	O(1)	O(1)	O(n)	O(n)
Delete a node from Binary Tree	O(n)	O(1)	O(n)	O(n)
Search for a node in Binary Tree	O(n)	O(1)	O(n)	O(n)
Traverse Binary Tree	O(n)	O(1)/O(n)	O(n)	O(n)
Delete entire Binary Tree	O(1)	O(1)	O(1)	O(1)
Space efficient?		No		Yes