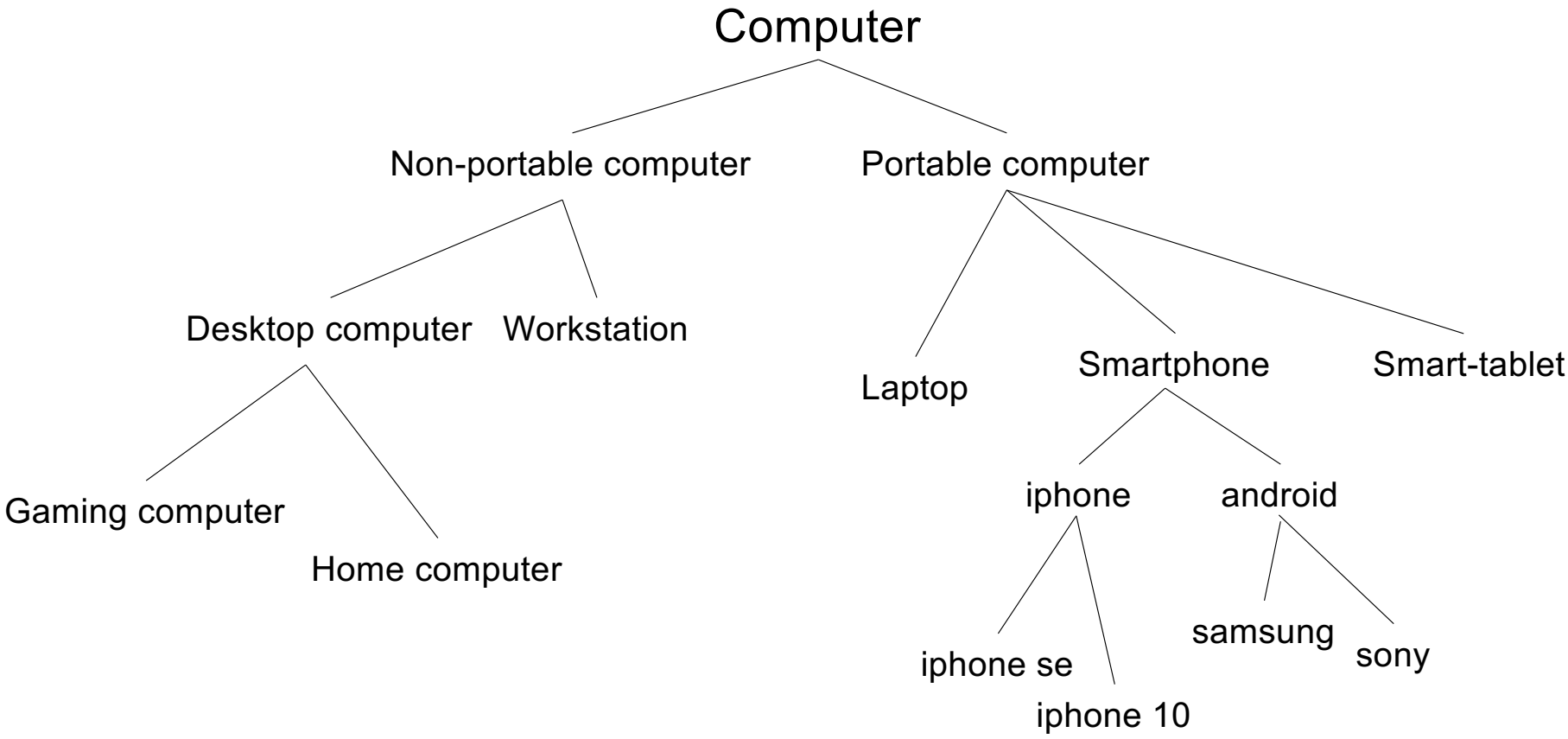
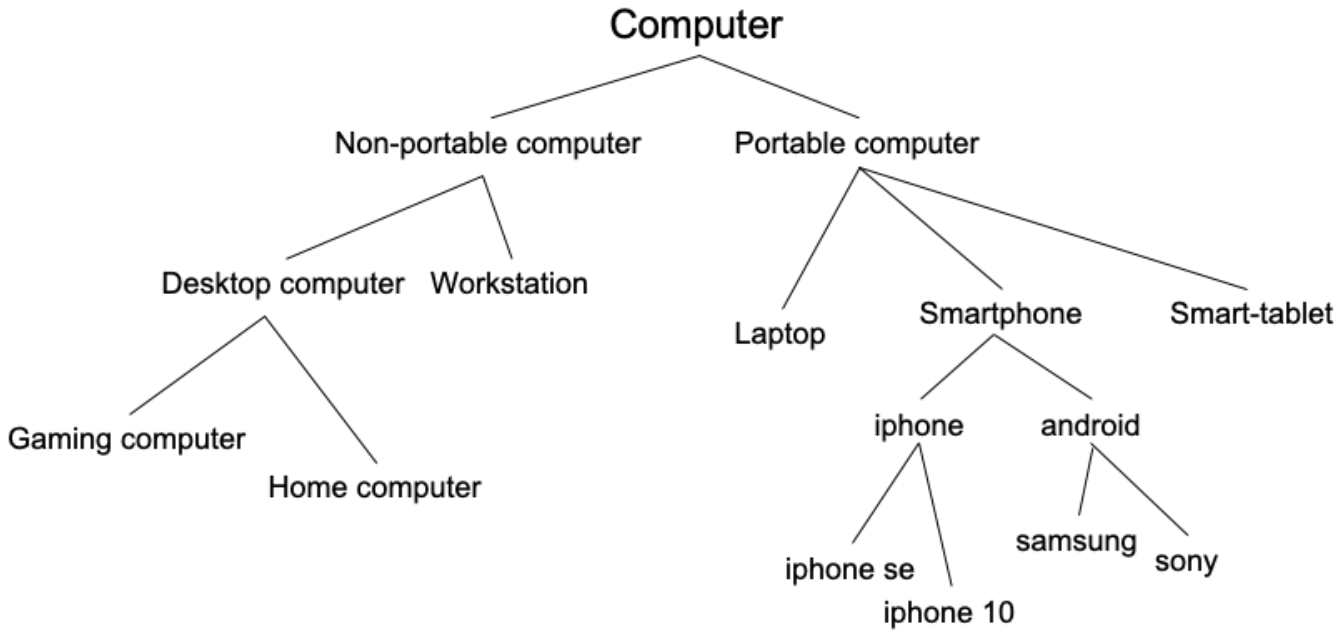
DS:
Binary Tree

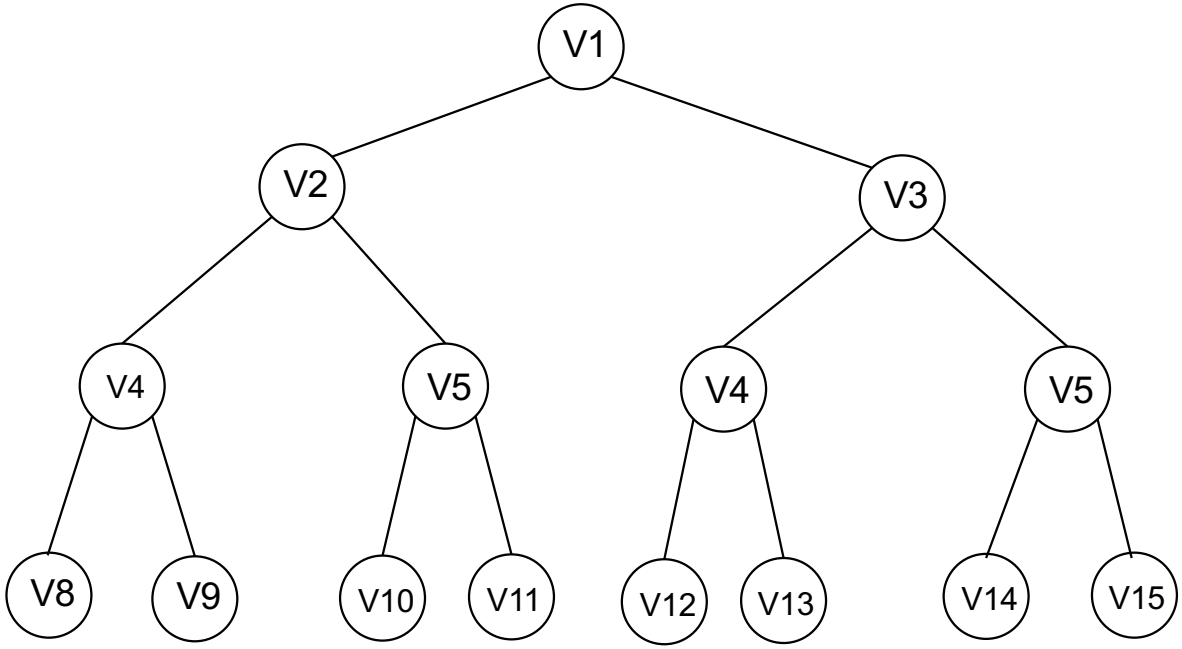
Liwei

What is tree


Computer
Non-portable computer
Portable computer
Desktop computer
Workstation
Laptop
Smartphone
Smart-tablet
Gaming computer
Home computer
iphone
android
samsung
sony
iphone se
iphone 10



- Properties of the above graph
  - Used to represent data in hierarchical form
  - Every node has 2 components (Data & References to its sub-category)
  - At top of it has Base product and 2 products called Left sub-catetory and Right sub-category under it.



- Properties of tree
  - Userd to represent data in hierarchical form
  - Every node (ideally) has 2 components (Data & References)
  - It has a Root node and 2 disjoint binary tree called left subtree and right subtree

Tree terminologies

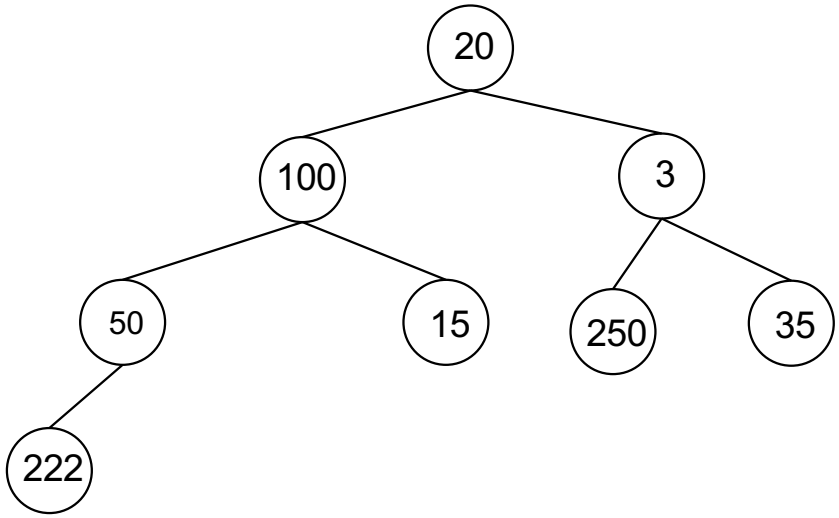
- Root: Node with no parent
- Edge: Link from partent to child
- Leaf: Node with no children
- Sibling: Children of same parent
- Ancestor: means Parent, grand-Parent, and so on for a given node
- Depth of node: Length of the path from root to node
- Height of node: Length of the longest path from that node to a leaf
- Height of tree: Max heigth among all the nodes

A root node will have a depth of 0.
A leaf node will have a height of 0.

Binary Tree

What is Binary Tree

- A tree is called as Binary tree if each node has zero, one or two children.

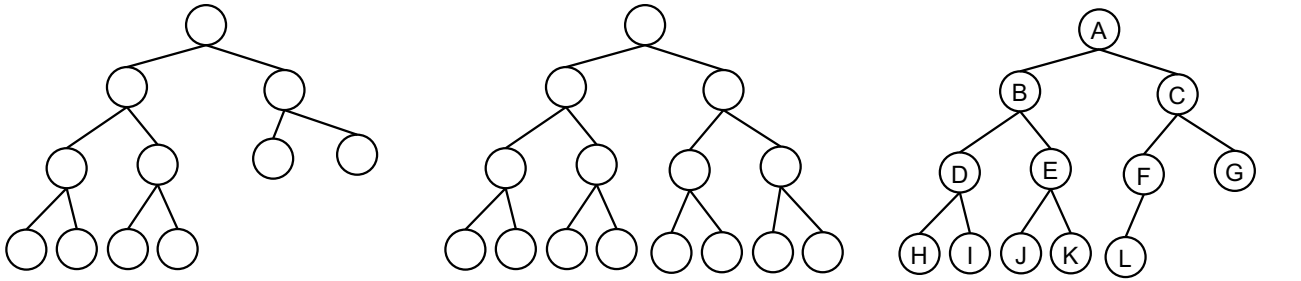


Why should we learn Binary Tree?

- Prerequisite for more advanced trees (BST, AVL, Red-Black tree, etc…)
- Is used in sovling specific problems like:
  - Huffman coding
  - Heap (priority queue)
  - Expression parsing

Types of Binary Tree

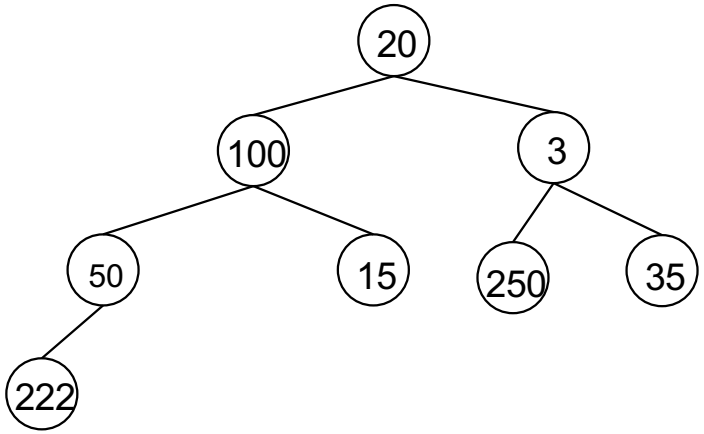
- Strict Binary Tree: if each node has either 2 children or none
- Full Binary Tree: if each non-leaf node has 2 children and all leaf nodes are at same level
- Complete Binary Tree: if all levels are completely filled except possibly the last level and the last level has all keys as left as possible

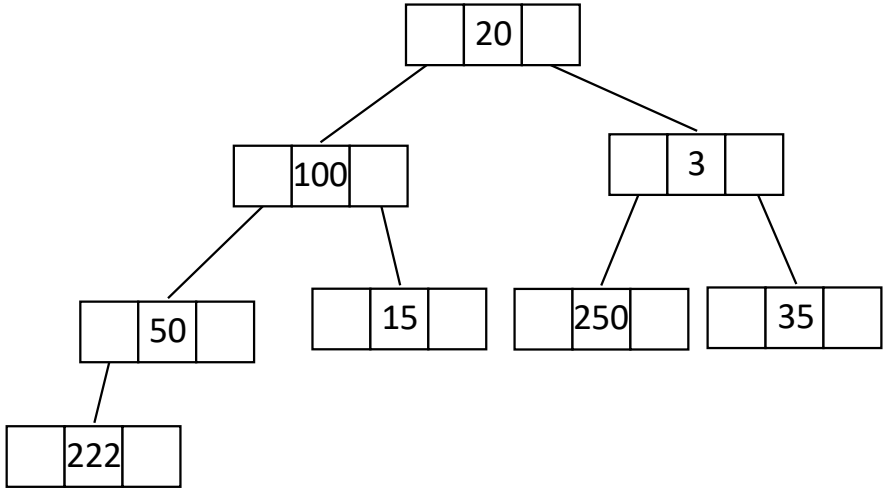


Strict Binary Tree Full Binary Tree Complete Binary Tree

Tree Representation

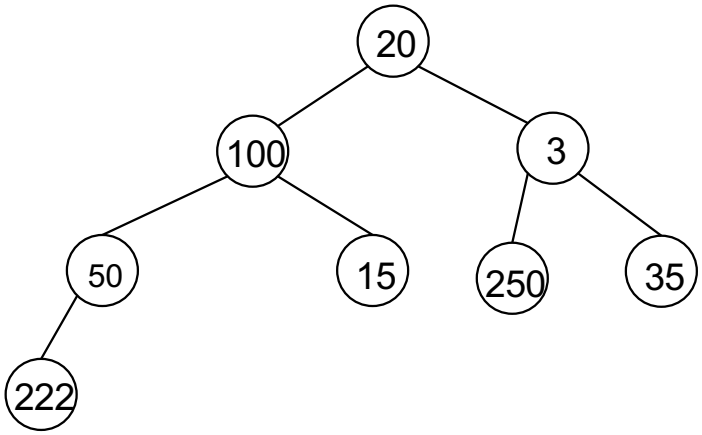
- How does tree look like at logical level?
- How does tree look when implemented via Linked-List?





Tree Representation

- How does tree look like at logical level?



- How does tree look when implemented via Array?

cell #	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
value		20	100	3	50	15	250	35	222									

Left Child - cell [2x]
Right Child - cell [2x+1]
x: index of parent node in the array

Binary Tree – Common operations

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

Binary Tree
(Linked-List implementation)

Creation of binary tree

createBinaryTree()

Create an object of Binary Tree class. O(1)

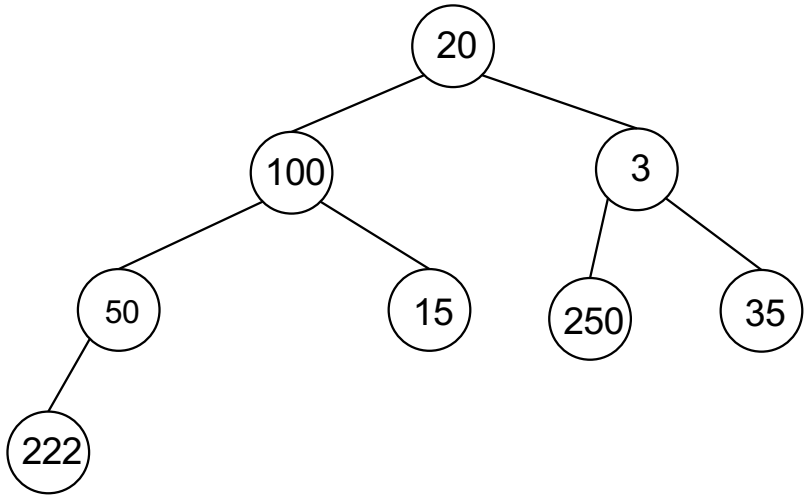
- Time complexity : O(1)
- Space complexity: O(1)

Traversing all nodes of Binary tree

- Depth first search:
  - Preorder Traversal
  - InOrder Travesal
  - PostOrder Travesal
- Breadth First Search:
  - LevelOrder Travesal

Pre-Order Traversal of Binary Tree

- Root
- Left subtree
- Right subtree



PreOrder: 20, 100, 50, 222, 15, 3, 250, 35

Pre-Order Traversal of Binary Tree

```
preOrderTraversal(root)
if (root equals null)
    return error message
else
    print root
    preOrderTraversal(root.left)
    preOrderTraversal(root.right)
```

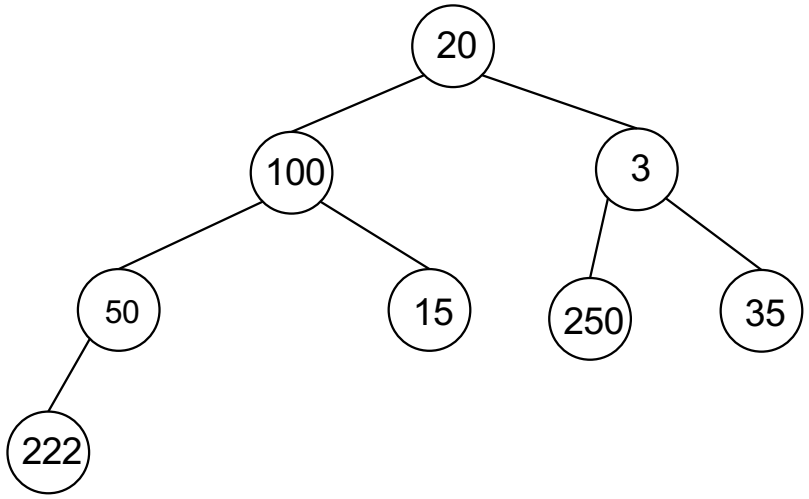
Pre-Order Traversal of Binary Tree

preOrderTraversal(root)	T(n)
if (root equals null)	O(1)
return error message	O(1)
else	O(1)
print root	O(1)
preOrderTraversal(root.left)	T(n/2)
preOrderTraversal(root.right)	T(n/2)

Time complexity: O(n)

In-Order Traversal of Binary Tree

- Left subtree
- Root
- Right subtree



InOrder: 222, 50, 100, 15, 20, 250, 3, 35

In-Order Traversal of Binary Tree

```
inOrderTraversal(root)
if (root equals null)
    return error message
else
    inOrderTraversal(root.left)
    print root
    inOrderTraversal(root.right)
```

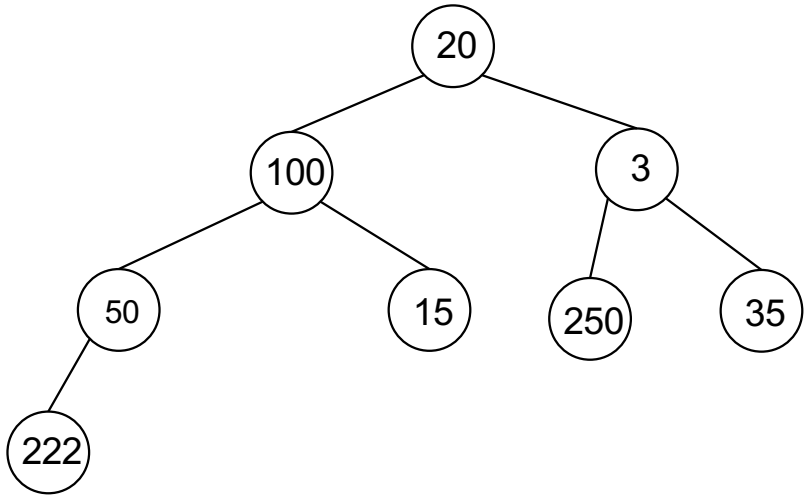
In-Order Traversal of Binary Tree

inOrderTraversal(root)	T(n)
if (root equals null)	O(1)
return error message	O(1)
else	O(1)
inOrderTraversal(root.left)	T(n/2)
print root	O(1)
inOrderTraversal(root.right)	T(n/2)

Time complexity: O(n)

Post-Order Traversal of Binary Tree

- Left subtree
- Right subtree
- Root



PostOrder: 222, 50, 15, 100, 250, 35, 3, 20

Post-Order Traversal of Binary Tree

```
postOrderTraversal(root)
if (root equals null)
    return error message
else
    postOrderTraversal(root.left)
    postOrderTraversal(root.right)
    print root
```

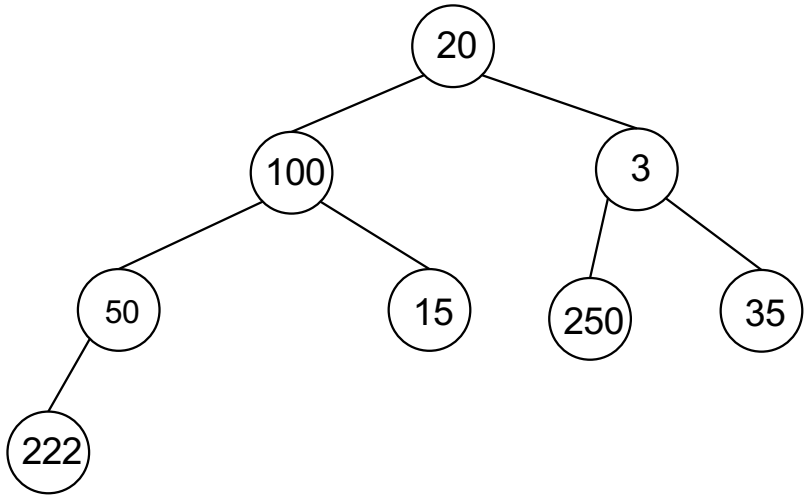
Post-Order Traversal of Binary Tree

postOrderTraversal(root)	T(n)
if (root equals null)	O(1)
return error message	O(1)
else	O(1)
postOrderTraversal(root.left)	T(n/2)
postOrderTraversal(root.right)	T(n/2)
print root	O(1)

Time complexity: O(n)

LevelOrder Traversal of Binary Tree

- By levels



LevelOrder: 20, 100, 3, 50, 15, 250, 35, 222

LevelOrder Traversal of Binary Tree

levelOrderTraversal(root)

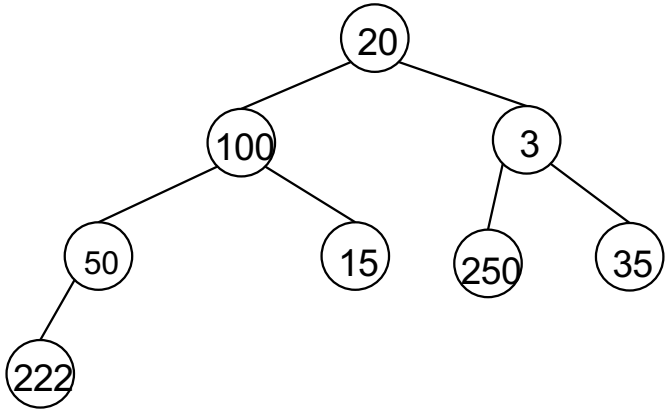
create a queue(Q)

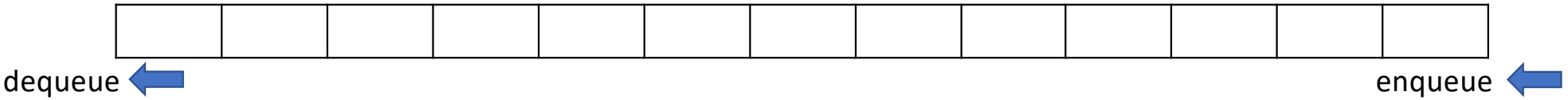
enqueue(root)

while(Queue is not empty)

enqueue() the child of first element

dequeue() and print





LevelOrder Traversal of Binary Tree

```
levelOrderTraversal(root)
      create a queue(Q)                                  O(1)
      enqueue(root)                                      O(1)
     while(Queue is not empty)                           O(n)
          enqueue() the child of first element           O(1)
         dequeue() and print                             O(1)
```

Time complexity: O(n)
Space complexity: O(n)

Searching a node in Binary Tree

```
searchForGivenValue(value)
        if root == null                          O(1)
                return error message             O(1)
        else                                     O(1)
                do a level order traversal       O(n)
        if value found                           O(1)
                return success message           O(1)
        return unsuccessful message              O(1)
```

Time complexity: O(n)

Insertion of node in Binary Tree

- Insertion:
  - When the root is blank
  - Insert at first vacant child

Insertion of node in Binary Tree

```
insertNodeInBinaryTree ()
        if root is blank                                                    O(1)
                insert new node at root                                     O(1)
        else                                                                O(1)
                do a level order traversal and find the first blank space   O(n)
        insert in that blank place                                          O(1)
```

Time complexity: O(n)

Deletion of node in Binary Tree

- deletion:
  - When the value to be deleted is not existing in the tree
  - When the value to be deleted exists in the tree

Deletion of node in Binary Tree

deleteNodeFromBinaryTree ()

search for the node to be deleted	O(n)
find deepest node in the tree	O(n)
copy deepest node's data in current node	O(1)
delete deepest node	O(1)

Time complexity: O(n)

Time and Space Complexity (Binary Tree)

	Time Complexity
Creation of Tree	O(1)
Insertion of value in Tree	O(n)
Deletion of value from Tree	O(n)
Searching for a value in Tree	O(n)
Traversing a Tree	O(n)