



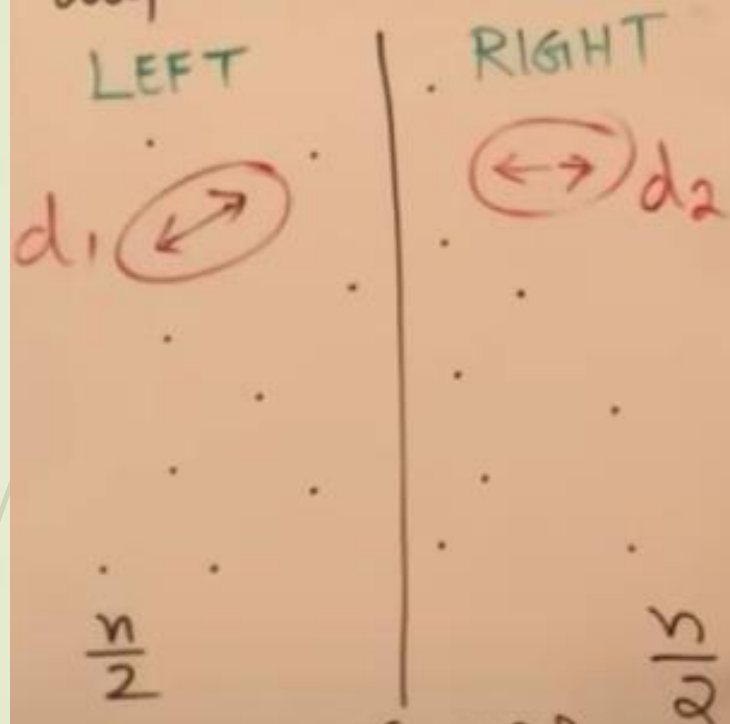
# Data Structure and Algorithms

Session-23

Dr. Subhra Rani Patra  
SCOPE, VIT Chennai

# Closest Pair Problem (in 2D)

Input: Set of  $n$  points in 2D  $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$   
Output: Find the "Closest pair"  $O(n^2) \longrightarrow O(n \log n)$



① Find the closest pair in LEFT ( $d_1$ )

② Find the closest pair in RIGHT ( $d_2$ )

③ Check if there exists a pair, s.t. one point is on the LEFT region, other point is on the RIGHT region, and the distance b/w them is

$$d = \min\{d_1, d_2\}$$

$T\left(\frac{n}{2}\right)$  (median  $O(n)$  time)

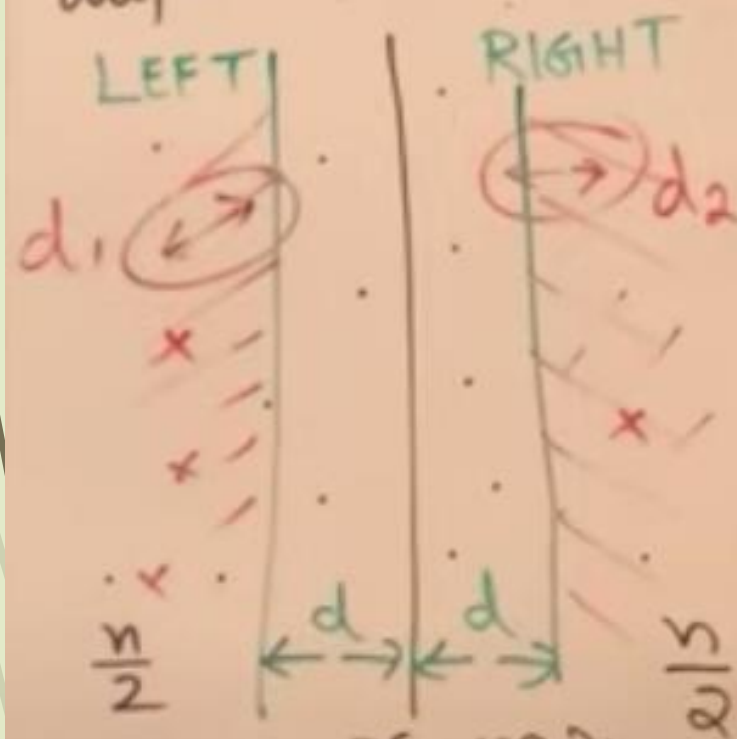
$T\left(\frac{n}{2}\right)$

# Closest Pair Problem (in 2D)

Input: Set of  $n$  points in 2D  $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$

Output: Find the "Closest pair"

$O(n^2) \longrightarrow O(n \log n)$



- ① Find the closest pair in LEFT ( $d_1$ )
- ② Find the closest pair in RIGHT ( $d_2$ )
- ③ Check if there exists a pair, s.t. one point is on the LEFT region, other point is on the RIGHT region, and the distance b/w them is

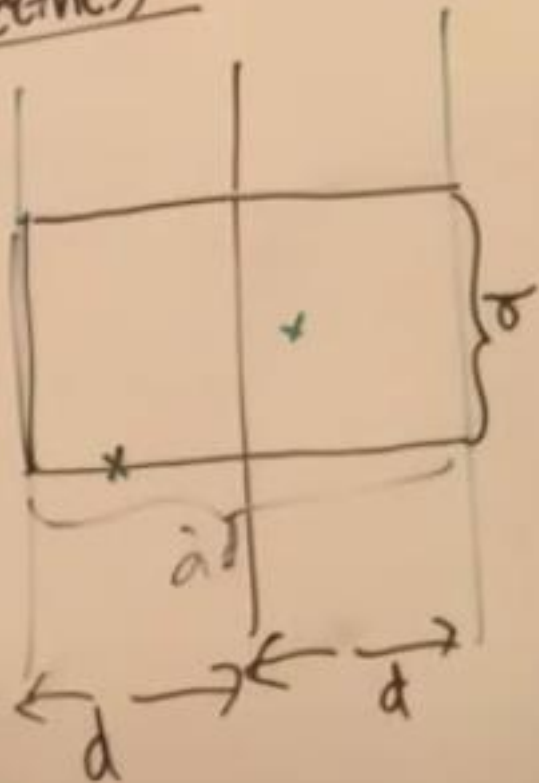
$$d = \min\{d_1, d_2\}$$

$P_1, P_2, P_3, P_4, P_5, \dots, P_{50}, P_{51}, \dots$

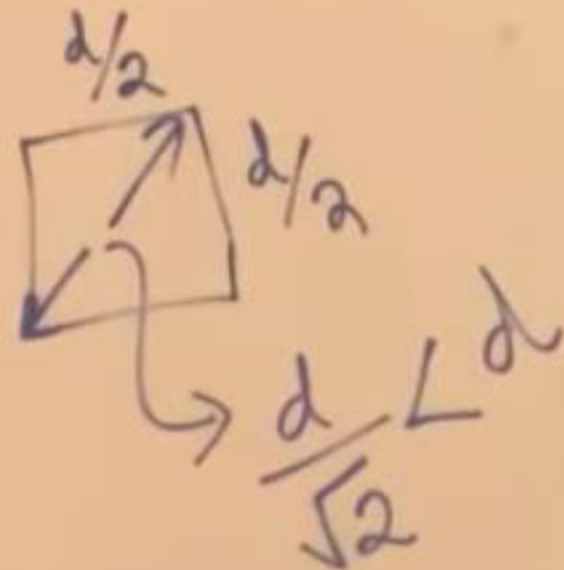
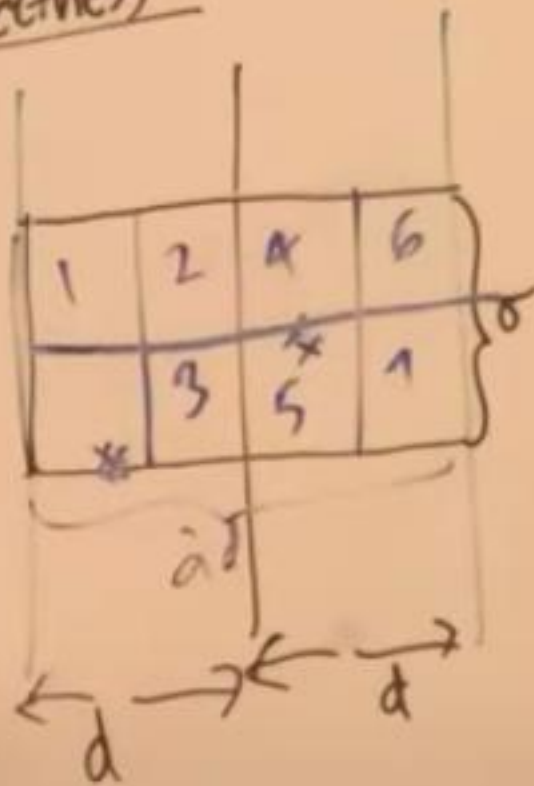
$T\left(\frac{n}{2}\right)$  (median  $O(n)$  time)


$T\left(\frac{n}{2}\right)$


Correctness



Correctness




$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) + O(n \log n)$$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n) = O(n \log^2 n)$$




# Algorithm:-

As a pre-processing step, input array is sorted according to x coordinates.

1) Find the middle point in the sorted array, we can take  $P[n/2]$  as middle point.

$P[ ] = \{13, 11, 12, 0, 14, 16, 1, 10, 17, 9, 2, 15, 3, 8, 4, 5, 7, 6\}$



$$P[n/2] = P[9] = 17$$

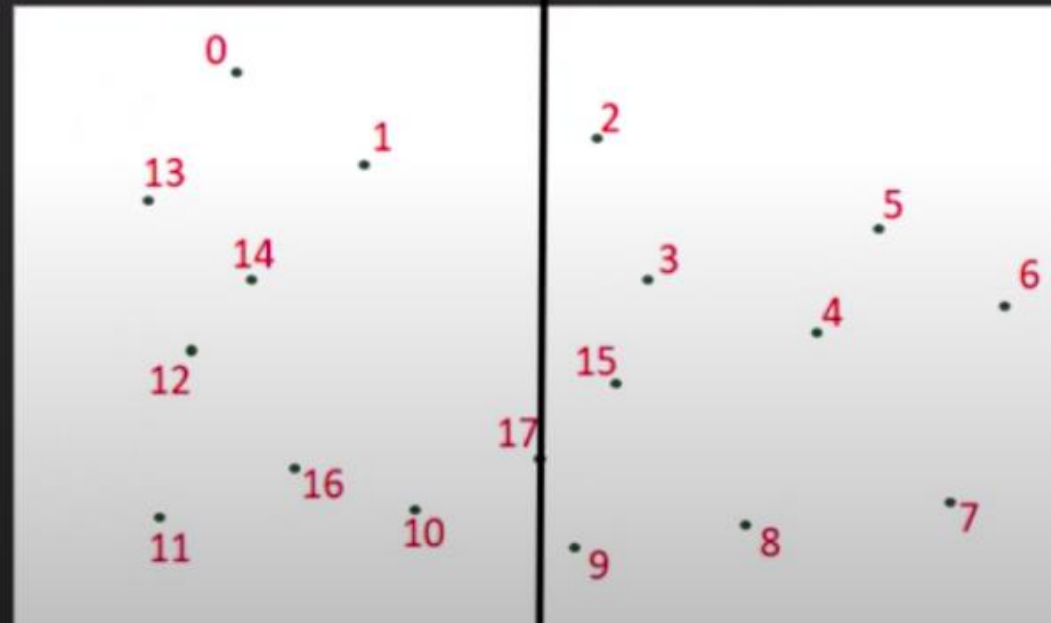
# Algorithm:-

2) Divide the given array in two halves. The first subarray contains points from  $P[0]$  to  $P[n/2]$ . The second subarray contains points from  $P[n/2+1]$  to  $P[n-1]$ .

$P[ ] = \{13, 11, 12, 0, 14, 16, 1, 10, 17, 9, 2, 15, 3, 8, 4, 5, 7, 6\}$

$P_L[ ] = \{13, 11, 12, 0, 14, 16, 1, 10, 17\}$

$P_R[ ] = \{9, 2, 15, 3, 8, 4, 5, 7, 6\}$



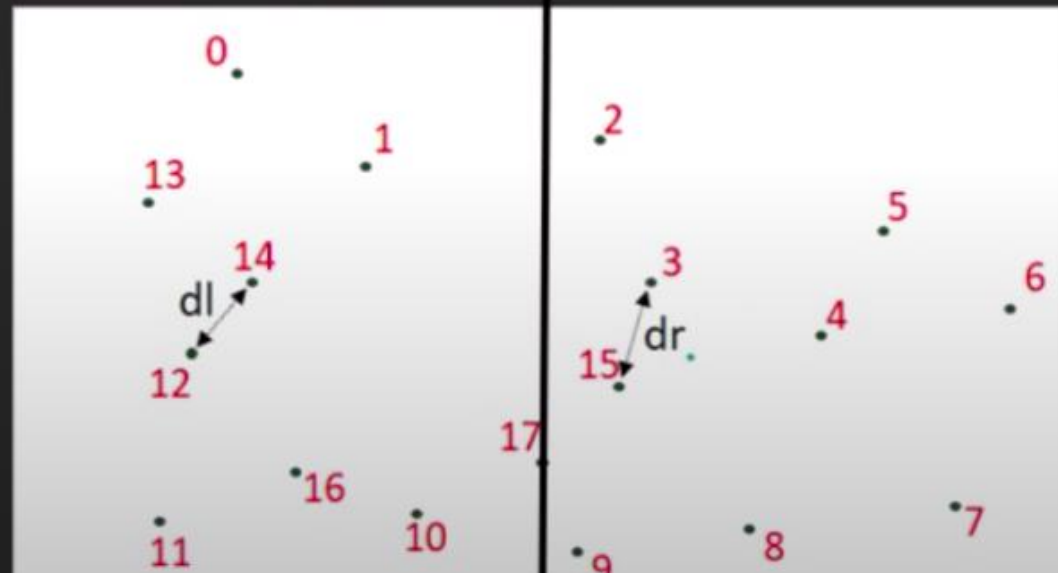
# Algorithm:-

3) Recursively find the smallest distances in both subarrays. Let the distances be  $dl$  and  $dr$ . Find the minimum of  $dl$  and  $dr$ . Let the minimum be  $d$ .

$$P[] = \{13, 11, 12, 0, 14, 16, 1, 10, 17, 9, 2, 15, 3, 8, 4, 5, 7, 6\}$$

$$P_L[] = \{13, 11, 12, 0, 14, 16, 1, 10, 17\}$$

$$P_R[] = \{9, 2, 15, 3, 8, 4, 5, 7, 6\}$$



$$d = \min(dl, dr)$$



# Algorithm:-

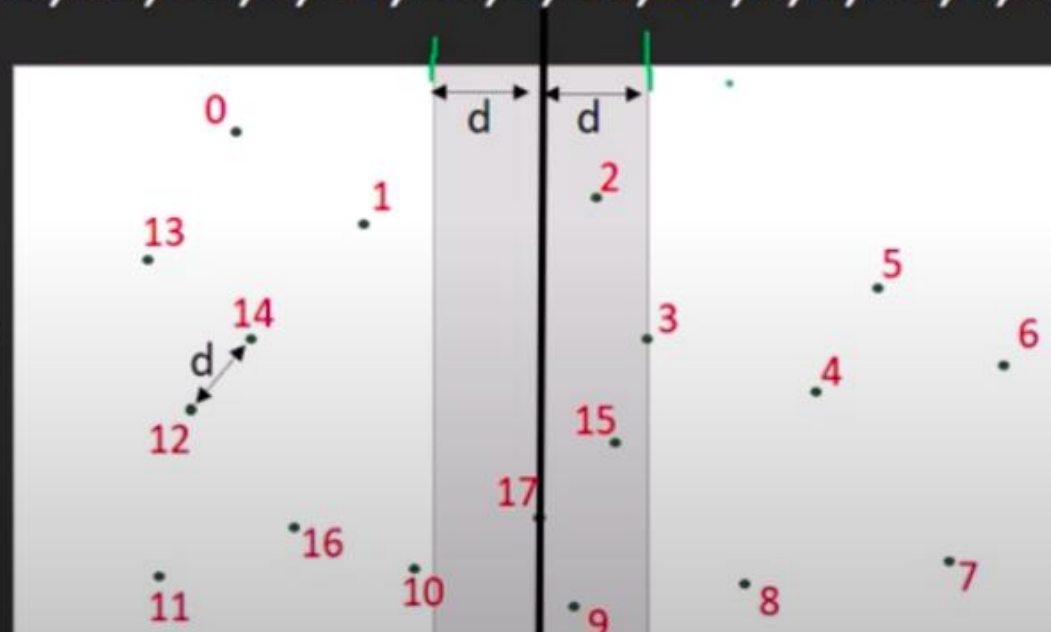
From above 3 steps, we have an upper bound  $d$  of minimum distance.

4) Now we need to consider the pairs such that one point in pair is from left half and other is from right half. Consider the vertical line passing through  $P[n/2]$  and find all points whose  $x$  coordinate is closer than  $d$  to the middle vertical line. Build an array `strip[]` of all such points.

$P[ ] = \{13, 11, 12, 0, 14, 16, 1, 10, 17, 9, 2, 15, 3, 8, 4, 5, 7, 6\}$

$P_L[ ] = \{13, 11, 12, 0, 14, 16, 1, 10, 17\}$

$P_R[ ] = \{9, 2, 15, 3, 8, 4, 5, 7, 6\}$



$$|pq| = \sqrt{(px - qx)^2 + (py - qy)^2}$$

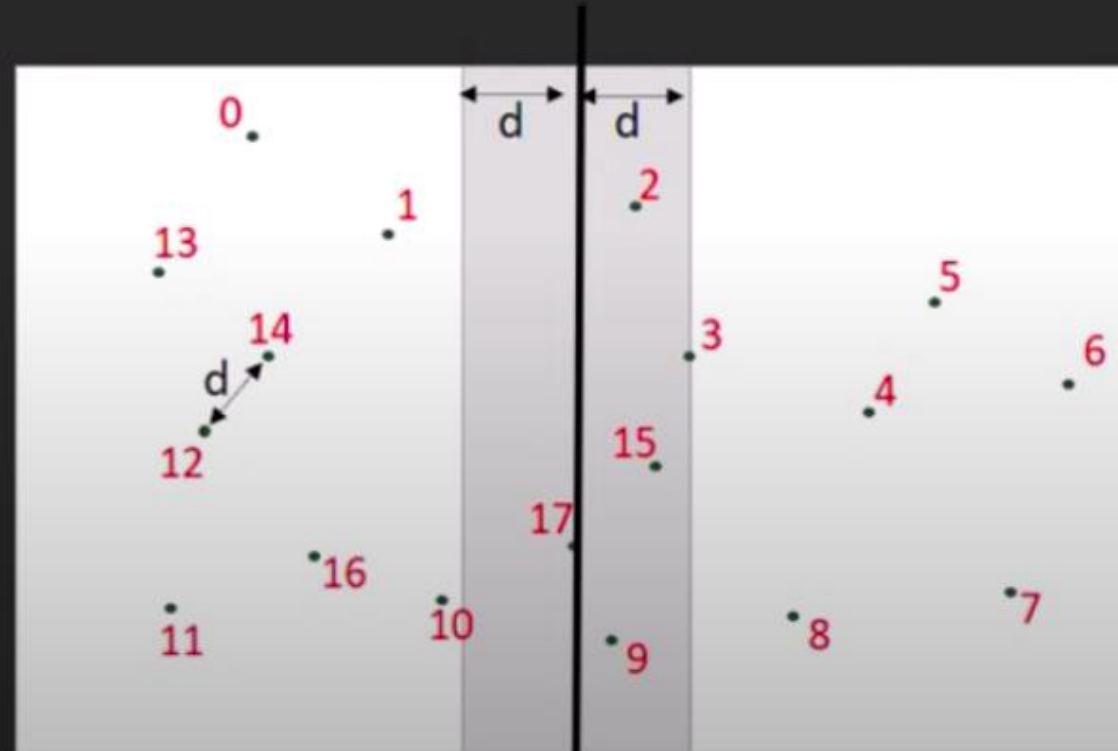
$$d = \min(d_l, d_r)$$

$\text{strip}[ ] = \{17, 9, 2, 15, 3\}$

# Algorithm:-

5) Sort the array `strip[]` according to y coordinates. This step is  $O(n \log n)$ .

`strip[ ] = {2,3,15,17,9}`

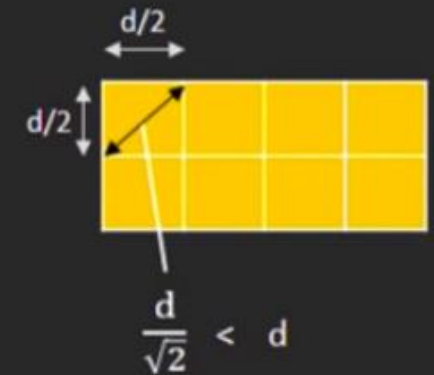
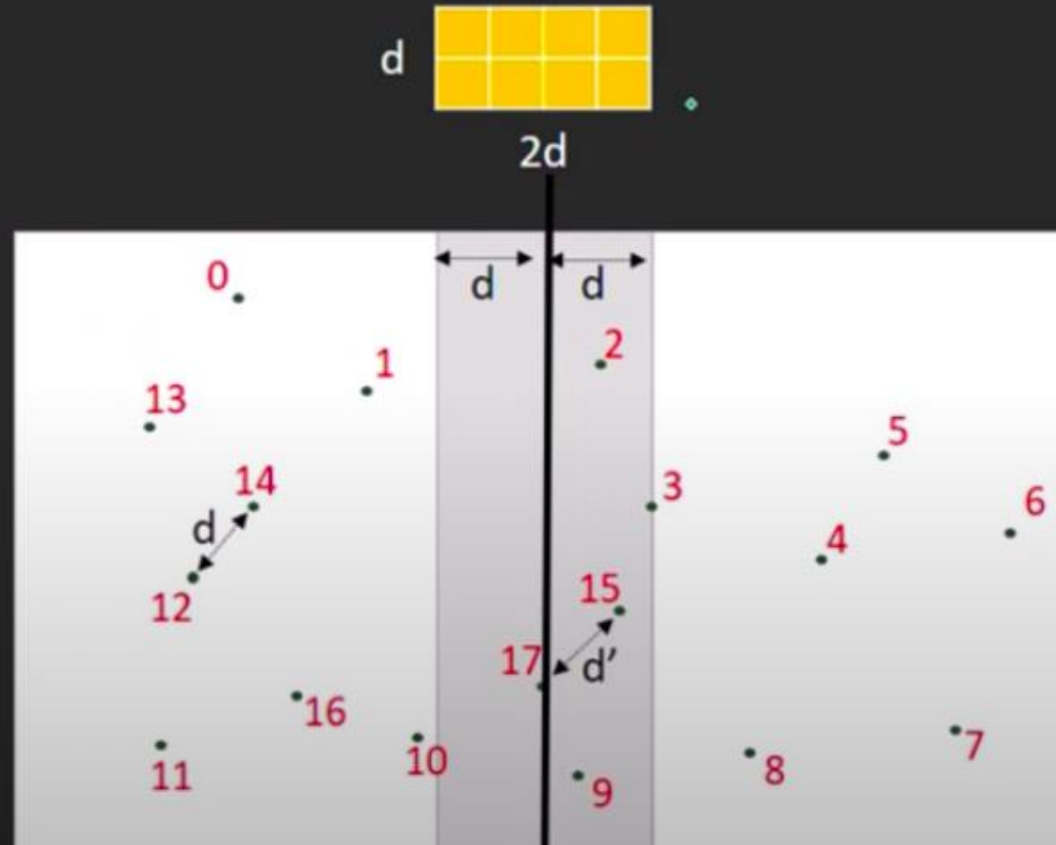


# Algorithm:-

6) Find the smallest distance  $d'$  in strip[].

strip[ ] = {2,3,15,17,9}

No square is shared between 2 halves. So, it's either on left or right.

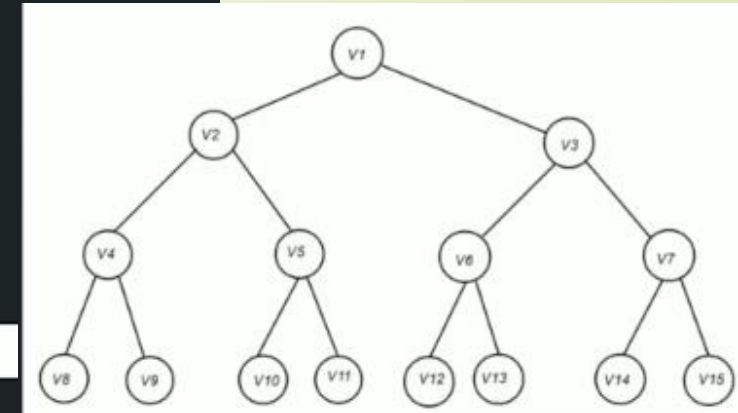
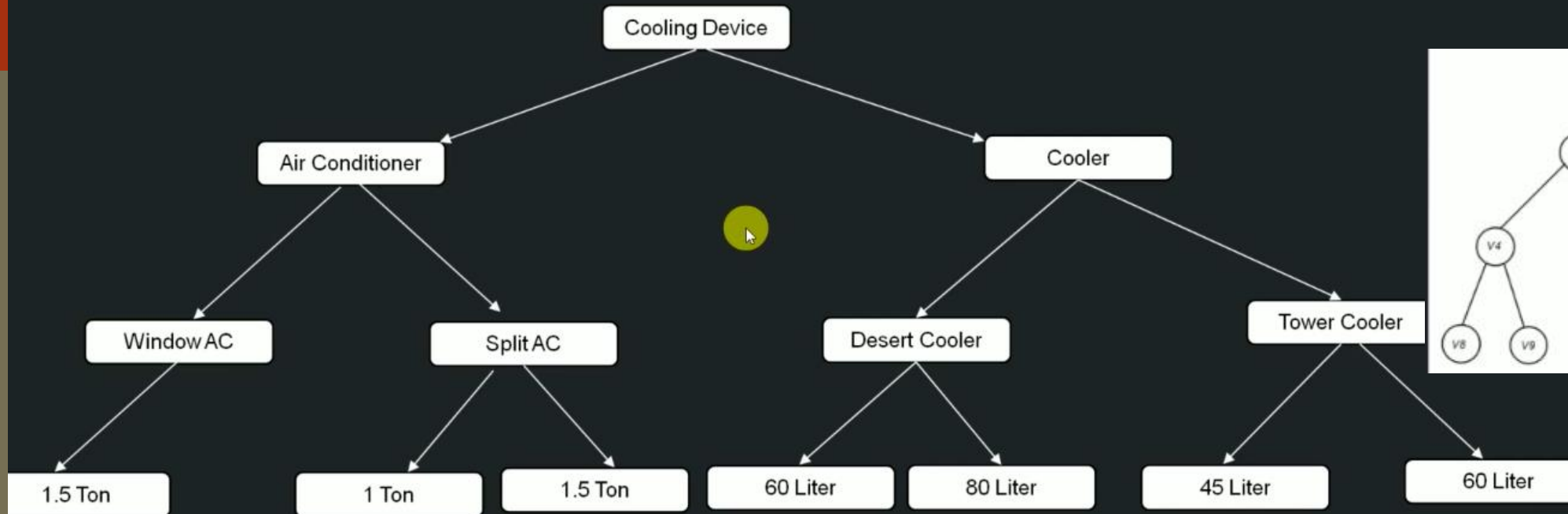


If 15 & 17 are the closest points in strip,  
 $d' = \text{distance b/w 15 \& 17.}$



# **Tree Data Structure**

# Tree



## Properties of above Pic:

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

## Properties of Tree:

- ✓ Used to represent data in hierarchical form
- ✓ Every Node(ideally) has 2 components (Data & Reference)
- ✓ It has a Root node and 2 disjoint binary tree called left subtree and Right subtree.



# Why should we learn Tree ?

| Operation                             | Array  | Linked List | Tree                             |
|---------------------------------------|--------|-------------|----------------------------------|
| Creation                              | $O(1)$ | $O(1)$      | Can we improve ?<br>Let's see... |
| Insertion                             | $O(n)$ | $O(n)$      |                                  |
| Deletion                              | $O(n)$ | $O(n)$      |                                  |
| Searching                             | $O(n)$ | $O(n)$      |                                  |
| Traversing                            | $O(n)$ | $O(n)$      |                                  |
| Deleting entire Array/LinkedList/Tree | $O(1)$ | $O(1)$      |                                  |
| Space Efficient ?                     | No     | Yes         |                                  |
| Implementation                        | Easy   | Moderate    |                                  |

# Tree Terminologies:

Root: Node with no parent

Edge: Link from Parent to Child

Leaf: Node with no children

Sibling: Children of same parent

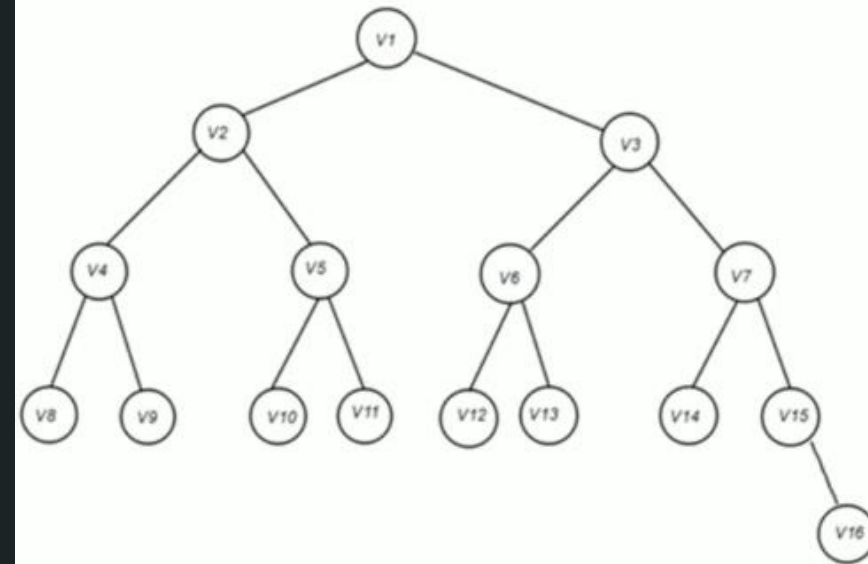
Ancestor: means Parent, grand-Parent, great grand parent, and so on for a given node.

Depth of node: Length of the path from root to node.

Height of node: Length of the path from that node to the deepest node.

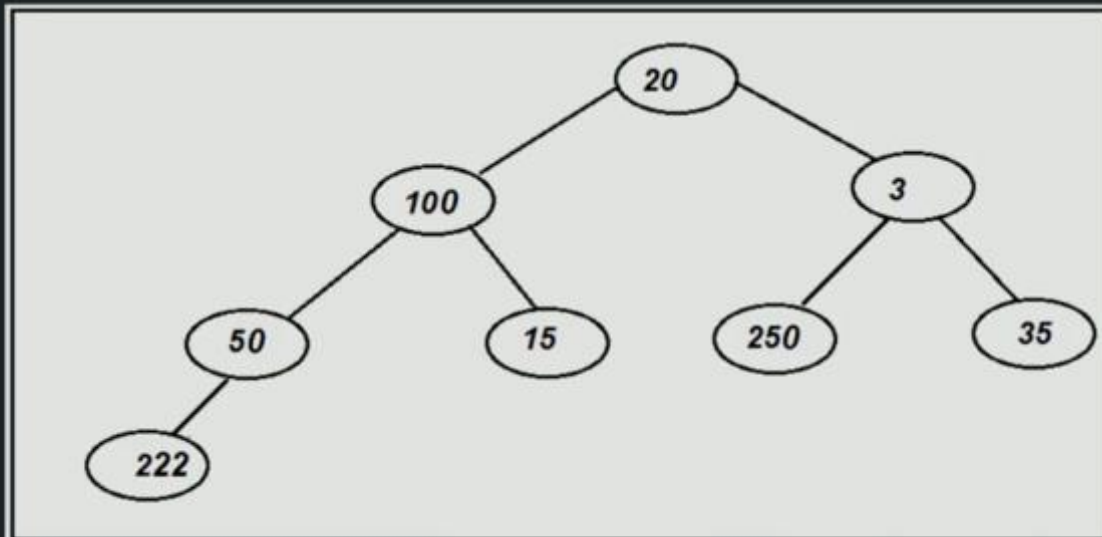
Height of tree: Same as height of Root node

Depth of tree: Same as depth of Root node



# What is Binary Tree:

- ✓ A tree is called as binary tree if each node has zero, one or two child.
- ✓ It is a family of Data structure (BST, Heap tree, AVL, Red-Black, Syntax Tree, Huffman Coding Tree, etc.)
- ✓ Example:



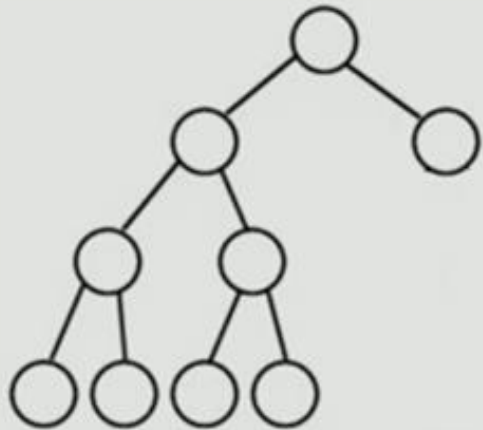
# Why should we learn Binary Tree ?



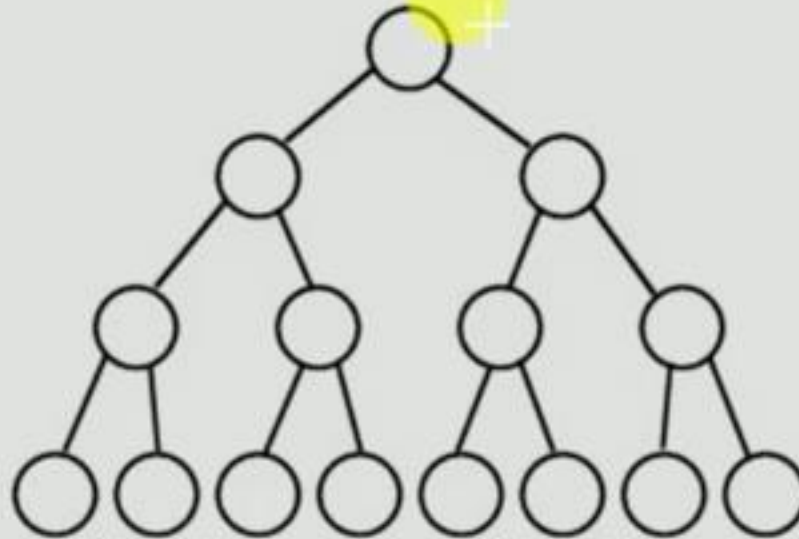
- ✓ Prerequisite for more advanced trees(BST, AVL, Red-Black, Expression Tree, etc...).
- ✓ Is used in solving specific problems like:
  - ✓ Huffman Coding
  - ✓ Heap(Priority Queue)
  - ✓ Expression parsing

# Types of Binary Tree:

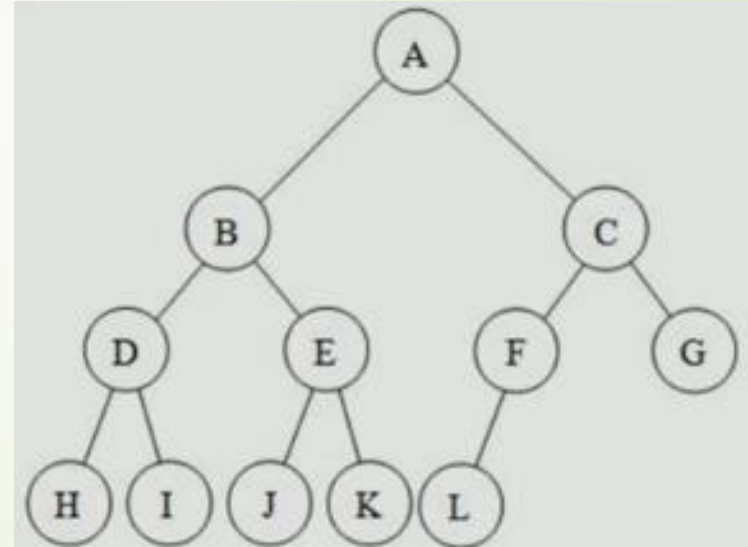
- ✓ **Strict Binary Tree:** if each node has either 2 children or none.
- ✓ **Full Binary Tree:** if each non leaf node has 2 children and all leaf nodes are at same level
- ✓ **Complete Binary Tree:** if all levels are completely filled except possibly the last level and the last level has all keys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



**Strict Binary Tree**



**Full Binary Tree**

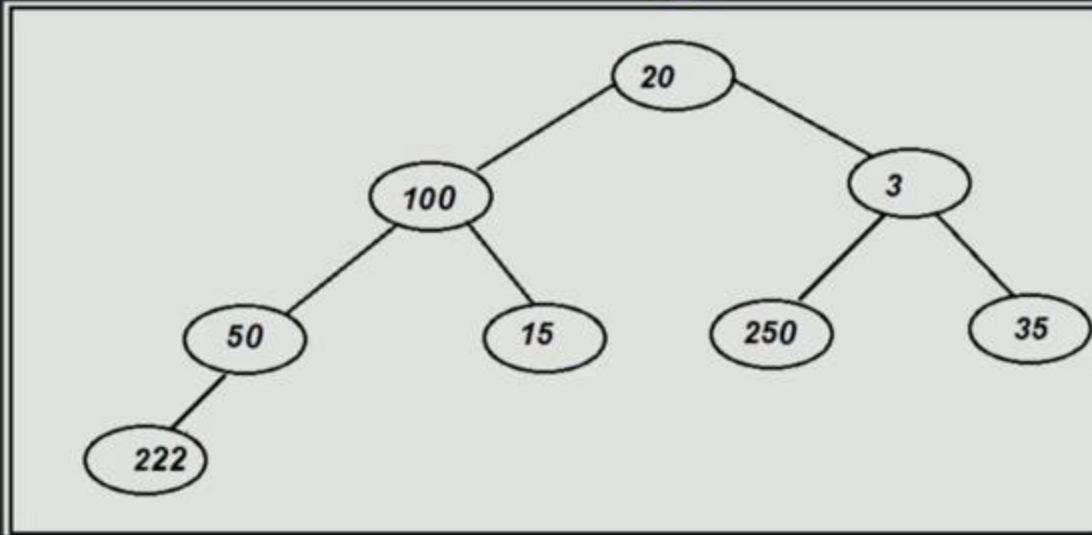


**Complete Binary Tree**



# Tree Representation(Continued...):

✓ How does tree looks like at logical level ?



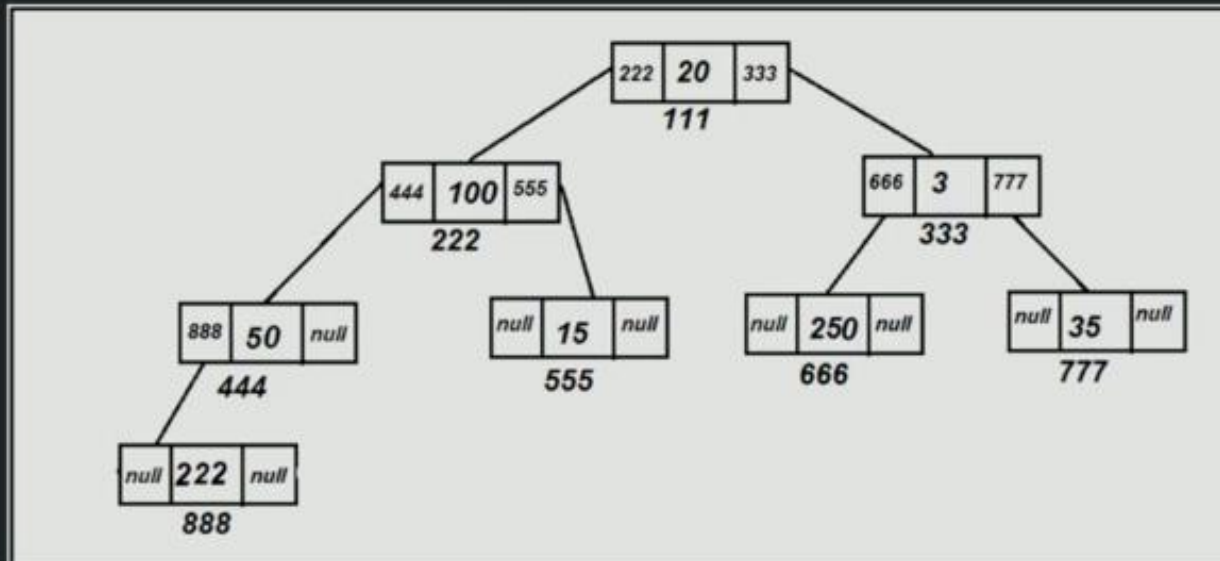
✓ How does tree looks when implemented via Array:

| 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  $[2x + 1]$

✓ How does tree looks when implemented via Linked-List:



# Binary Tree – Common operations

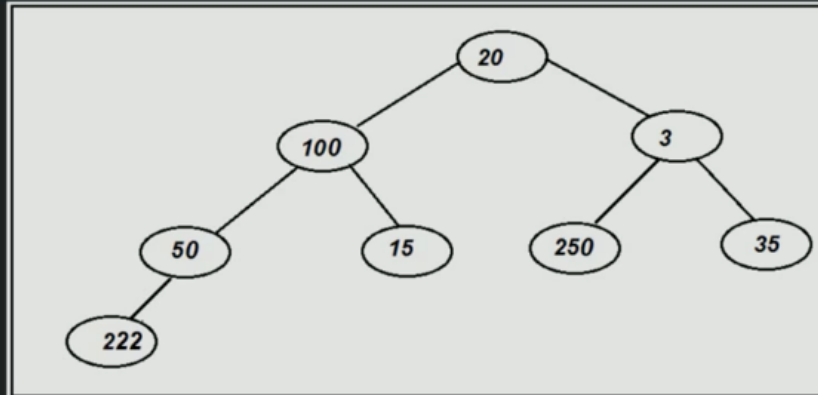
- ✓ *Creation of Tree*
- ✓ *Insertion of a node*
- ✓ *Deletion of a node*
- ✓ *Search for a value*
- ✓ *Traverse all nodes*
- ✓ *Deletion of Tree*

## Algorithm- Creation of Binary Tree (Linked-List implementation):

*createBinaryTree()*

*Create an object of Binary Tree class*

## Traversing all nodes of Binary Tree(Linked-List implementation):



### ✓ Depth First Search:

- ✓ PreOrder Traversal
- ✓ InOrder Traversal
- ✓ PostOrder Traversal

### ✓ Breadth First Search:

- ✓ LevelOrder Traversal

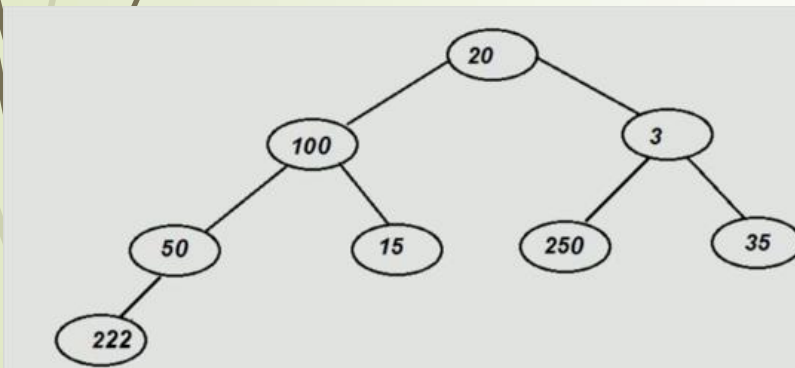


## Algorithm- 'Pre-Order Traversal' of Binary Tree(Linked-List implementation):

Root

Left Subtree

Right Subtree

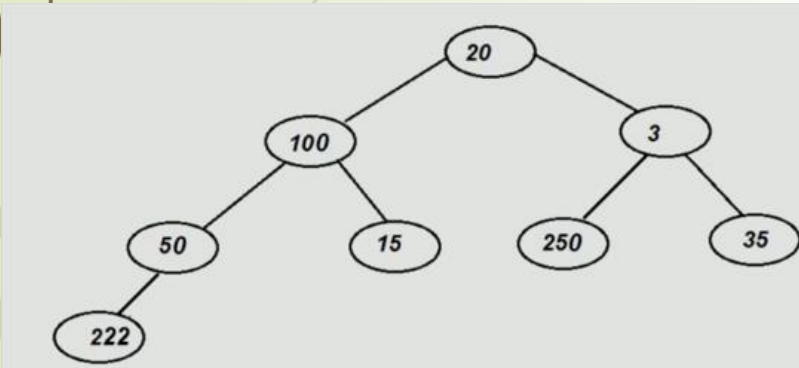


20 (100 (3) ( ) ) (3 ( ) ( ) )

20 (100 ( ) ( ) ) (3 (250) (35) )

20 (100 (50, 222) (15) ) (3 (250) (35) )

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



```
preorderTraversal(root)
```

```
if (root equals null)
```

```
    return error message
```

```
+
```

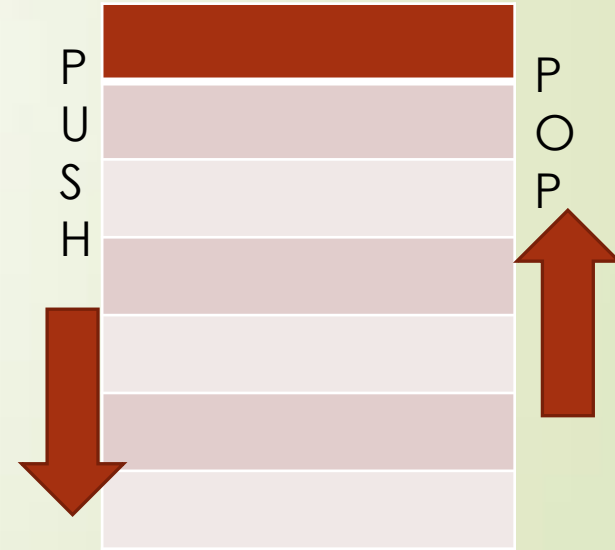
```
else
```

```
    print root
```

```
    preorderTraversal (root.left)
```

```
    preorderTraversal(root.right)
```

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





*Thank  
you*