DS → Linear DS → continuous and non continuous

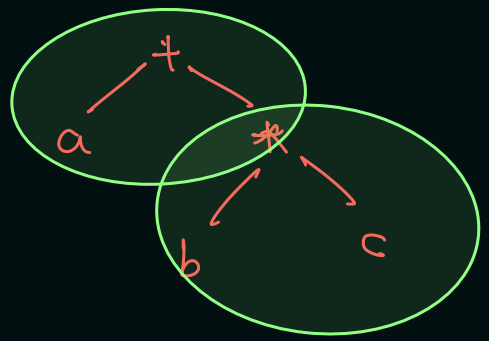
Non Linear DS → Trees , Memory

Mathematical Expression →
Heap (Binary Tree) → importance of left & Right node.

→ Stack.

Evaluation → Infix

a + b * c BDMAS

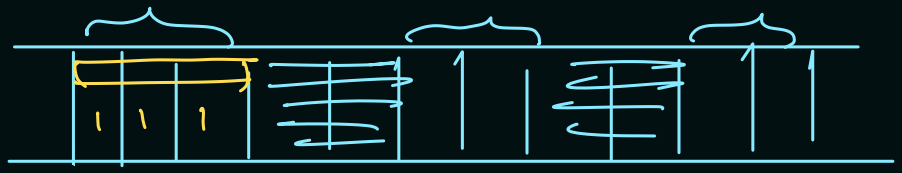


* Why Recursion considered good in trees but in LL.

Linear DS

Recursion → level of Recursion (no. of node)

Max. no. nodes → Amount of memory free Heap.



Max. size of array = 3

Max size of LL = ∞

Stack → overflow → fix sized

LL can cross size of array.

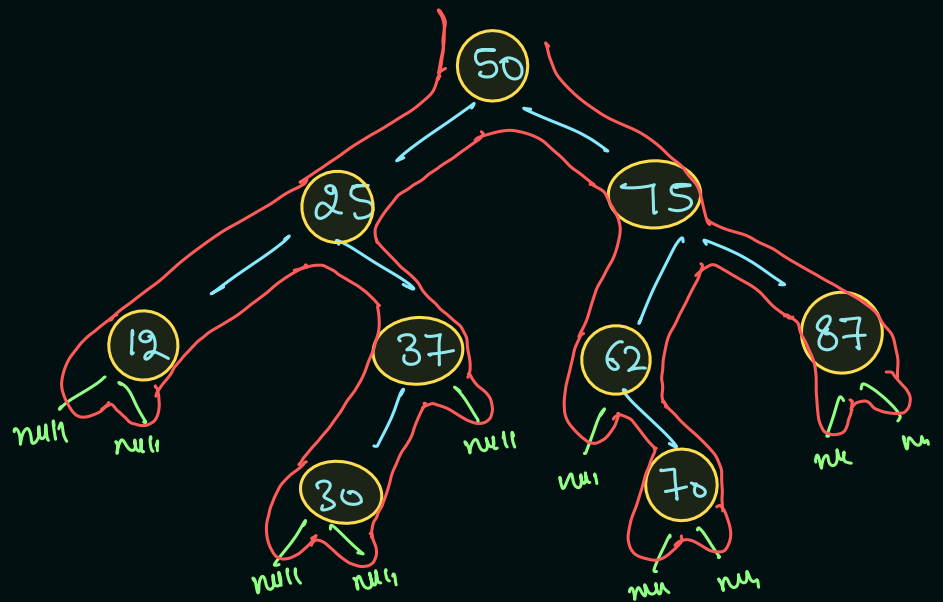
Height of Tree = $\log_2 n$

put value of n , $n = 10^9 \equiv 2^{32}$

$\log_2 2^{32} = 32$ → max level

Reverse pointer recursively.

Example (BST)



print zig-zag-

↓

Areas- in Recursion

- Pre Area
- In Area
- Post Area

Traversal — ~~pre~~ Pre Order Traversal

~~In~~ In order Traversal

~~Post~~ post order Traversal

* Level order Traversal

Run - function

[Pre Area [Area Before call]]

Call ①

⟶ In Area [Area After first call & Before last call]

Call ②

⟶ Post Area [Area after all calls-

Pre Order ① → 50 25 12 87 30 75 62 70 87

In Order ② → 12 25 80 37 50 62 70 75 87

Post order ③ → 12 30 37 25 70 62 87 75 50

Construction using pre order

Binary Tree

Node → data]

left pointer] Node

Right pointer] Node-

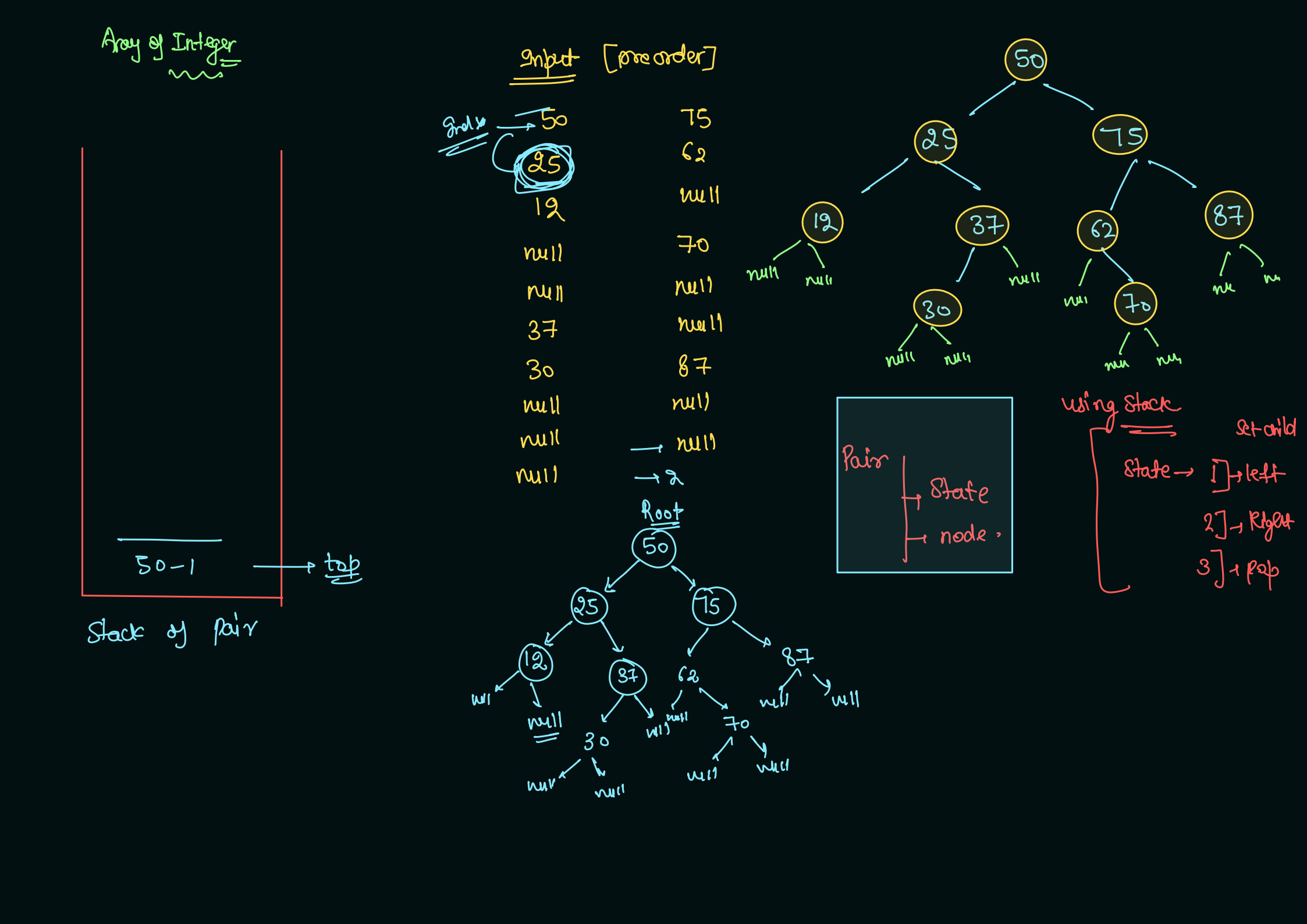
Memory-

LL- Generic Tree

data

children

Generic Tree

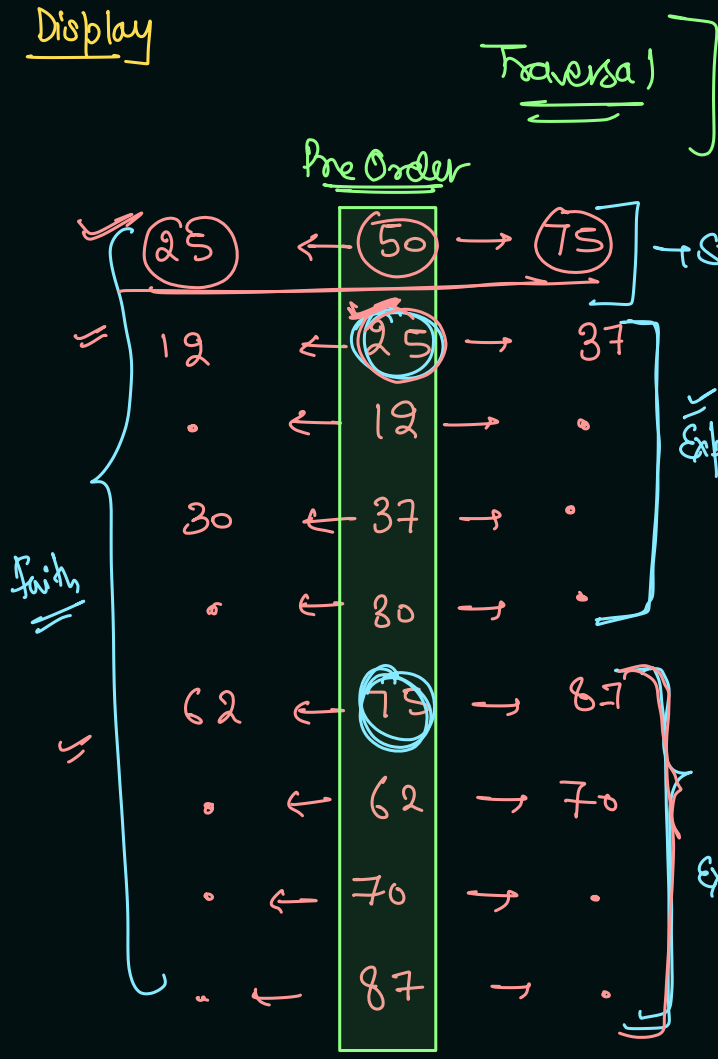

Array of Integer
Input [preorder]
50
25
12
null
null
37
30
null
null
75
62
null
70
null
null
87
null
null
idx
50
25
12
null
null
37
30
null
null
75
62
null
70
null
null
87
null
null
50-1
top
Stack of pair
Root
Pair
State
node
using Stack
Set child
State → 1] → left
2] → Right
3] → Pop

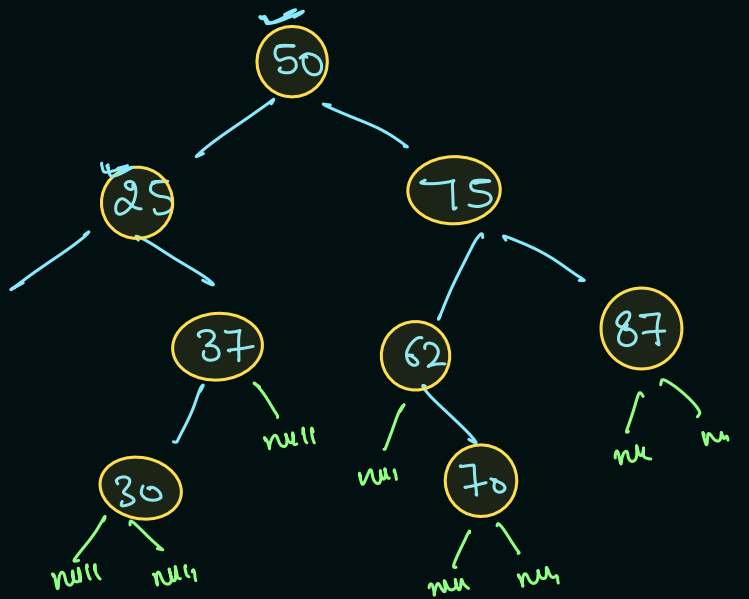
Display

Traversal → work Area

↳ Pre Area

Pre Order





Recursion

- faith
- Expectation
- Merging of Both

faith → Root → Display

Expectation → Exp. ①
Exp ②

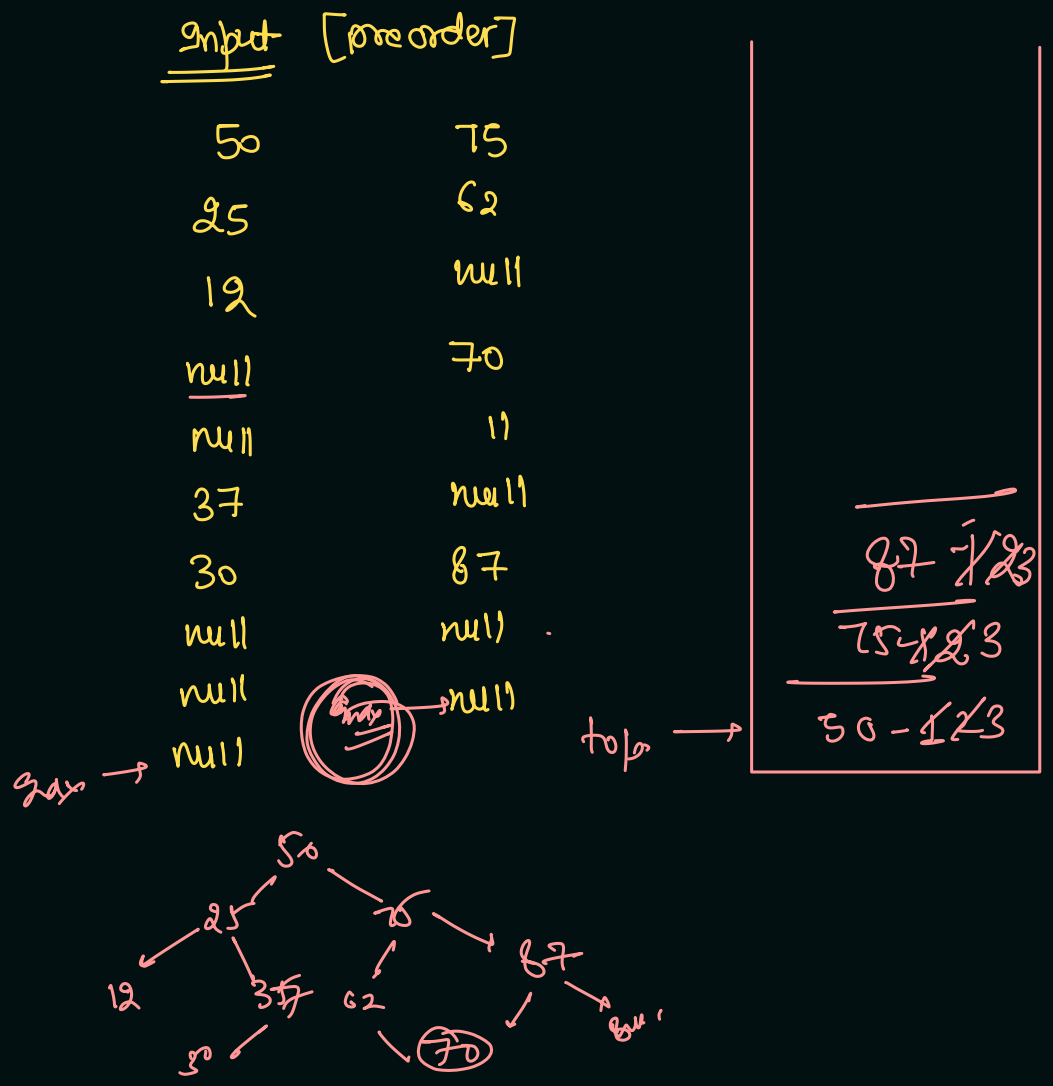
Merging → Self work
Exp ①
Exp ②

DS → get
set
Retrieve
→ optimise

↳ Display

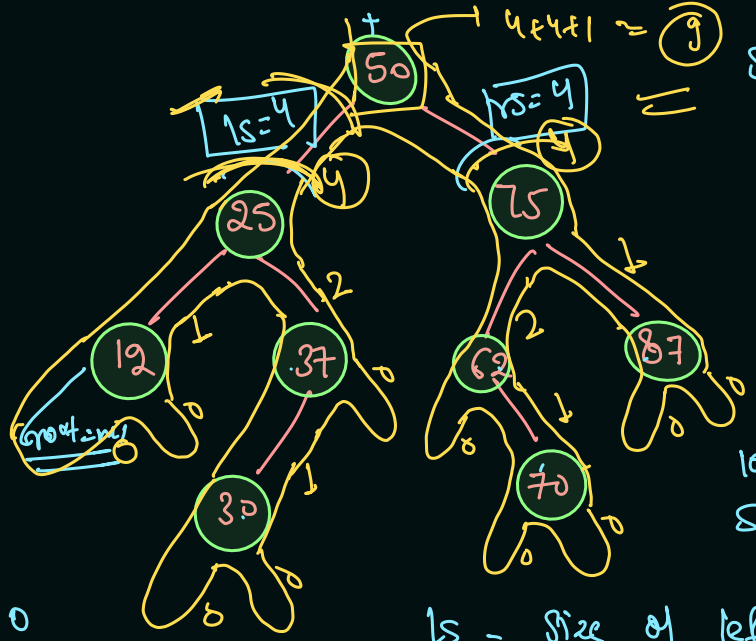
```java
int indx = 0;
while(indx < data.length && st.size() > 0) {
    Pair top = st.peek();
    if(top.state == 1) {
        indx++;
        if(data[indx] != null) {
            Node nn = new No
            top.node.left = nn;
            st.push(new Pair(nn, 1));
        }
        top.state++;
    } else if(top.state == 2) {
        indx++;
        if(data[indx] != null) {
            Node nn = new Node(data[indx]);
            top.node.right = nn;
            st.push(new Pair(nn, 1));
        }
        top.state++;
    } else {
        st.pop();
    }
}
```

Node nn - btree.co



1. Size
2. Sum
3. max
4. Height

of Binary Tree



ls = size of leftsubtree

rs = size of Rightsubtree

complete size = ls + rs + 1

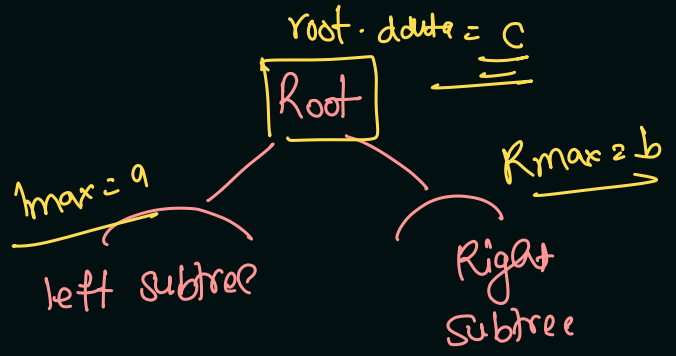
```java
public static int size(Node node) {
    if(node == null) return 0;
    int lsize = size(node.left);
    int rsize = size(node.right);
    int msize = lsize + rsize + 1;
    return msize;
}
```

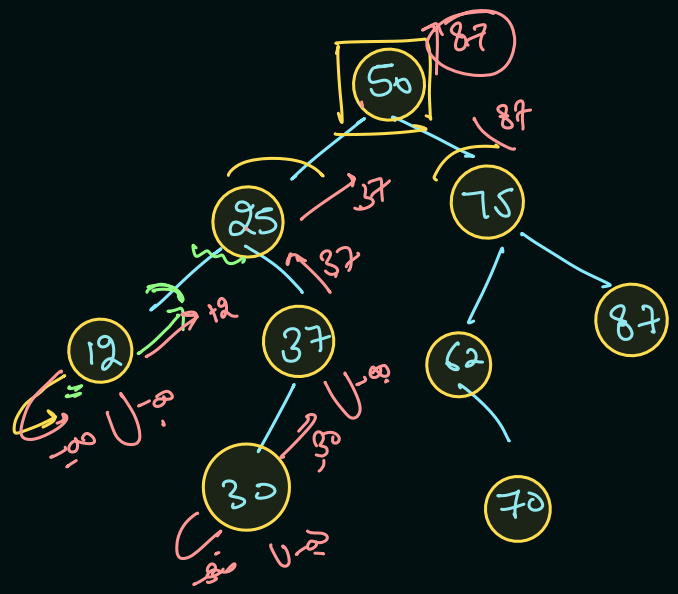
sum & max is similar to size

sum
- lsum = from left subtree
- rsum = from Right subtree
- Total sum = lsum + rsum + root data

Base case ⇒ root = null
↳ return 0

Max →





max. of left subtree

max. of Right subtree,

Final Result = Max (a, b, c)

what if (root == null)

Every operator have an Identity.

a * e = a

operator

e → Identity of that operation.

op	Identity
+	0
-	0
*	1
max()	$-\infty$
min()	$+\infty$

if(root == null)

↳ Identity of operator

a + 0 = a

0 is identity of +

a * 1 = a

a / 1 = a

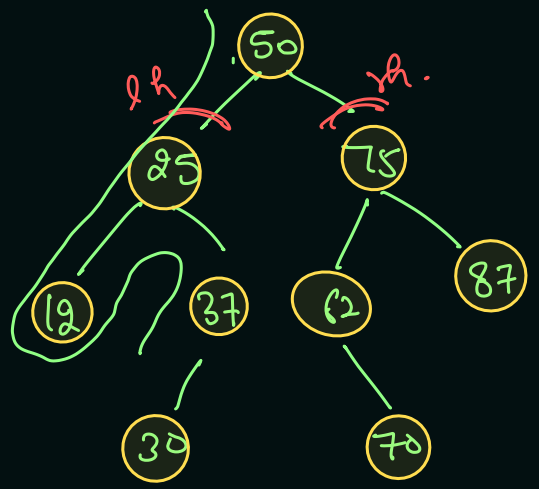
max(a, e) = a

e = $-\infty$

Height → Structure based problem

what is Height?

Max [distance] between Root to leaf is Height of the tree.



Height = 4

OR

Height = 3

distance

distance = 2[Node] OR 1 [Edge]

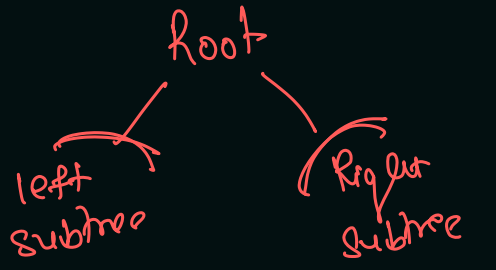
Height → on the basis of Edge

on the basis of node.

left Subtree Height = lh

Right subtree Height = rh

Max distance, my Height = max(rh, lh) + 1



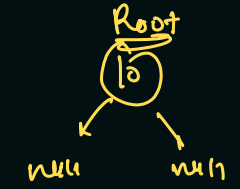
Node base

height = ? = 1

max(lh, rh) + 1

↳ 0

Edge base



Height = ? = 0

max(lh, rh) + 1

↳ -1

// Node basis

```
if(rooot == null)
    return 0
```

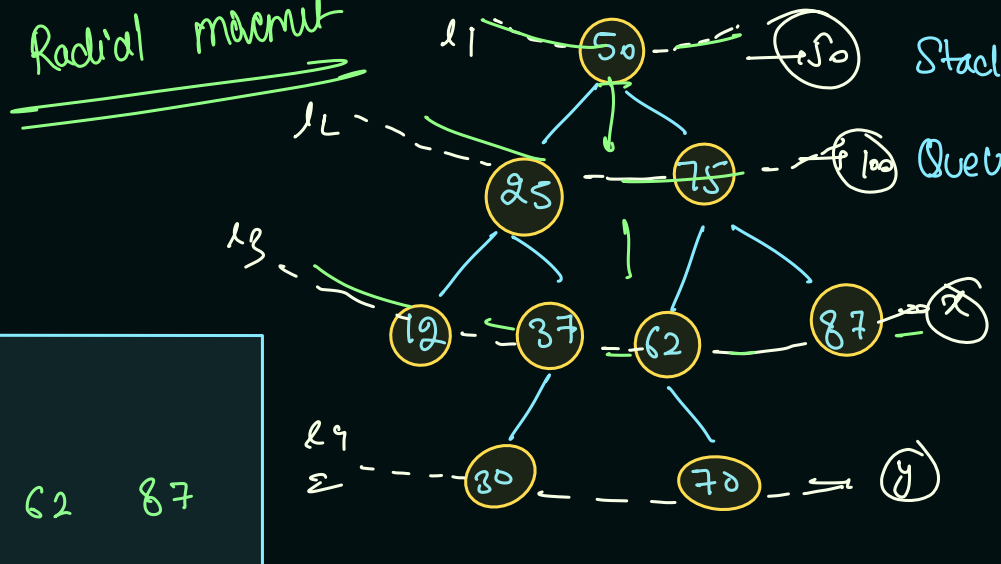
Edge basis

```
if(root == null)
    return -1;
```

level Order of a Binary Tree

Algo = BFS from Graph

Radial manner

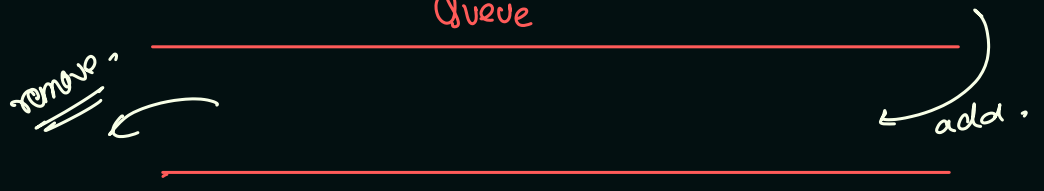


level order →

level 1 →	50
level 2 →	25 75
level 3 →	12 87 62 87
level 4 →	30 70

level order dsm

D.S. used for level order → Queue



Steps of Algo.

1. Get + Remove
2. work → print
3. Add children

level order → BFS → Height

Queue size = ~~1~~ ~~2~~ 4 2

hit Enter

50

25 75

12 37 62 87

30 70

```
while(que.size() != 0)
    size = que.size
    while(size-- > 0)
        get + remove
        work
        Add children
    hit Enter.
```

LinkedList as a queue → [add → addLast ; remove → removeFirst]

↳ getFirst

Size is common in Both.

level order

Delimiter → null

using two queue

while → in while

→ LinkedList

get + Remove] → C++ . [get ; remove] remove doesn't return anything in C++

while JAVA return removal element

Work → print

add children

till Right] height → level ,

height

n+ = 4

function] task Root

① → ⓐ

② ⓑ ⓒ

③

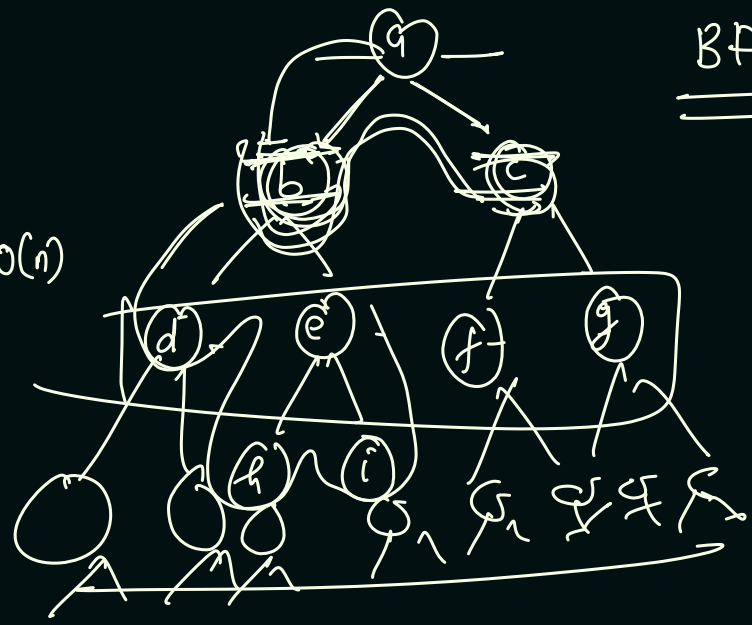
①

Singletravele - O(n)

Recursively

O(n²)

BFS:



Iterative Pre, Post And In order:

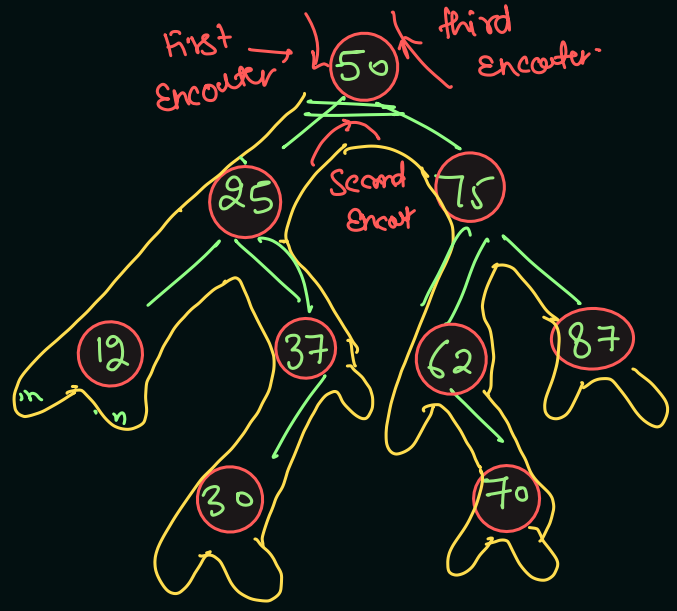
Base problem → construction of tree

State -

First Encounter → Pre Area] → Pre order] → ①

Second Encounter → In Area] → In order] → ②

Third Encounter → Post Area] → Post order] → ③



Pre Order → 50 25 12 37 30 75 62 70 87

In order → 12 25 30 37 50 62 70 75 87

Post order → 12 30 37 25 70 62 87 75 50

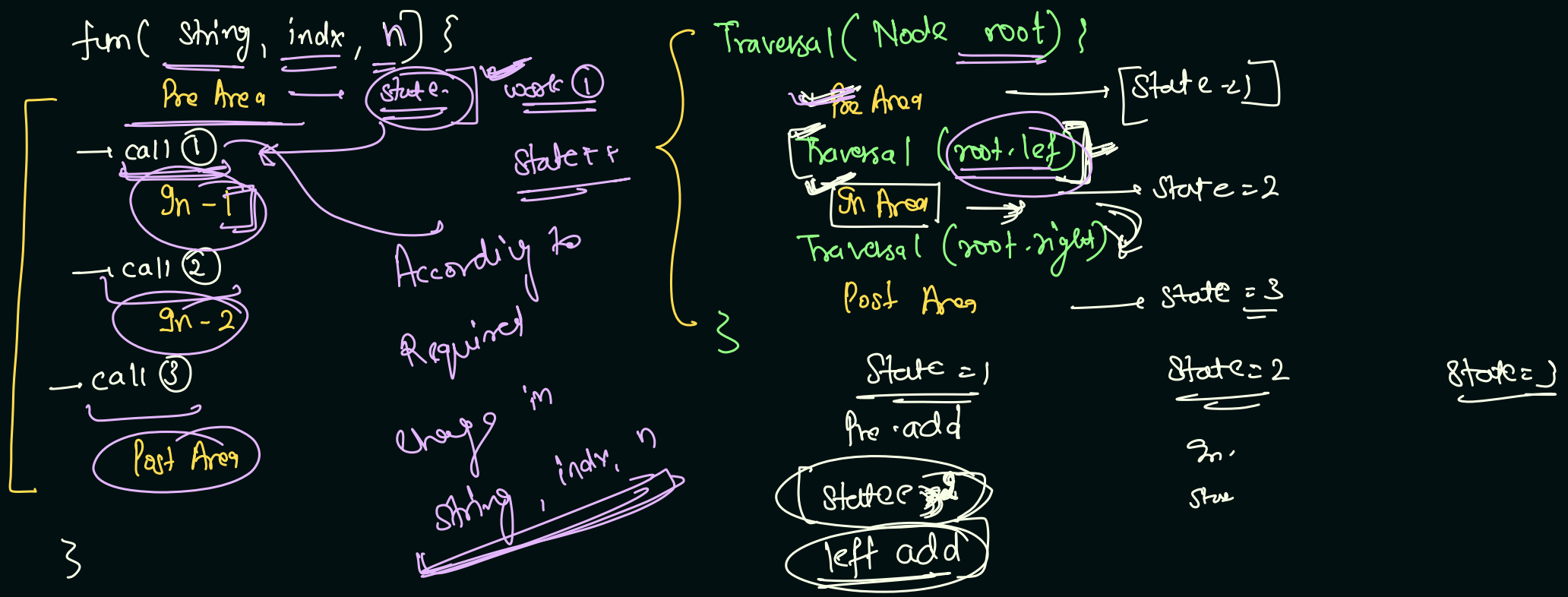
root = 50

get top :

top.state = 1 → Pre Add
↳ left child add in stack
top.state ++

= 2 → In Add
↳ Right child add in stack
top.state ++

= 3 → Post add pop from stack



Task] — [Tower of Hanoi]

[using Iterative method]