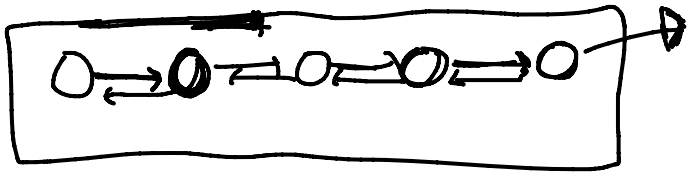
CLASS 55

12 April 2023 21:07

Doubly LL

Node — head = head → next
head = head → prev

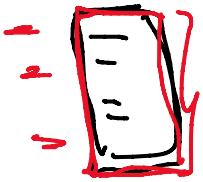


Trees

class Node
{ int data;
Node* next;
Node* prev;
}

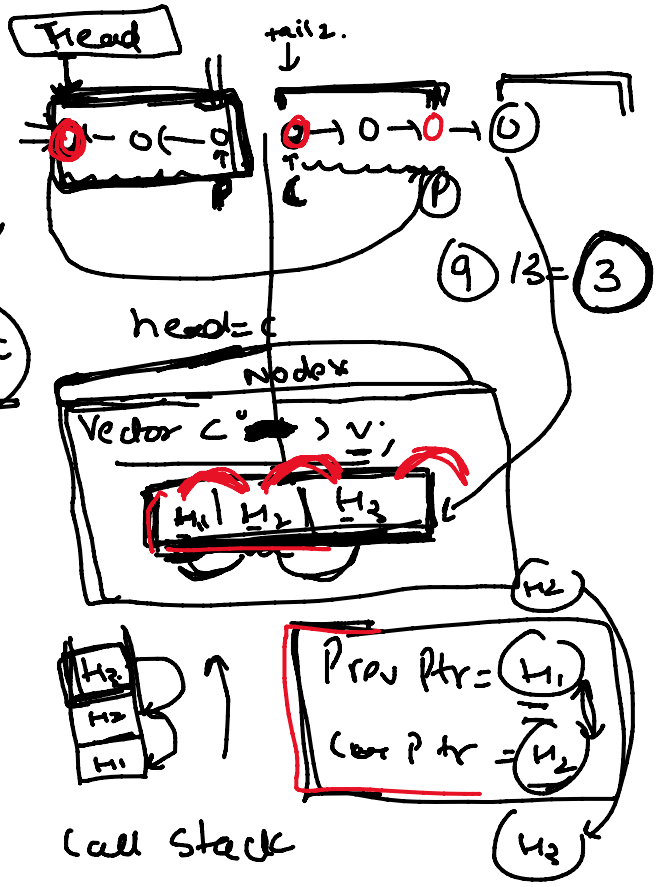
void reverse()
{
while (head)
{ int length = len(head);
}
}

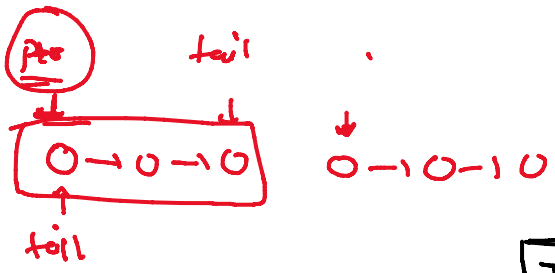
Iterative
Recursive



reverse ka part

head =
Prev Ptr = H2
Cur Ptr = H3





Trees

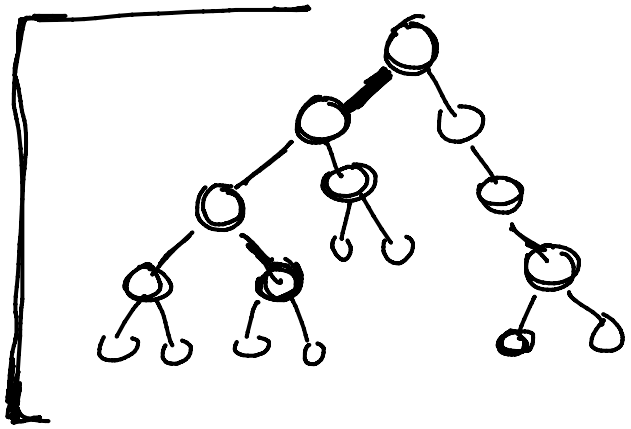
→ linked list

Singly LL :- O→O→O→O→O

1 Dimensional
linear Data Structure

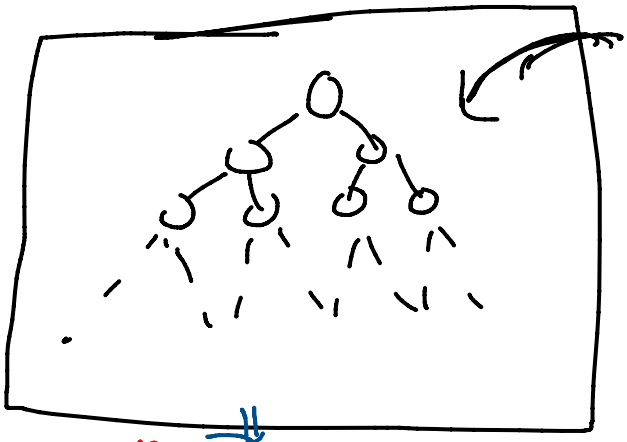
linear Data Structure

→ Trees Non linear linked list

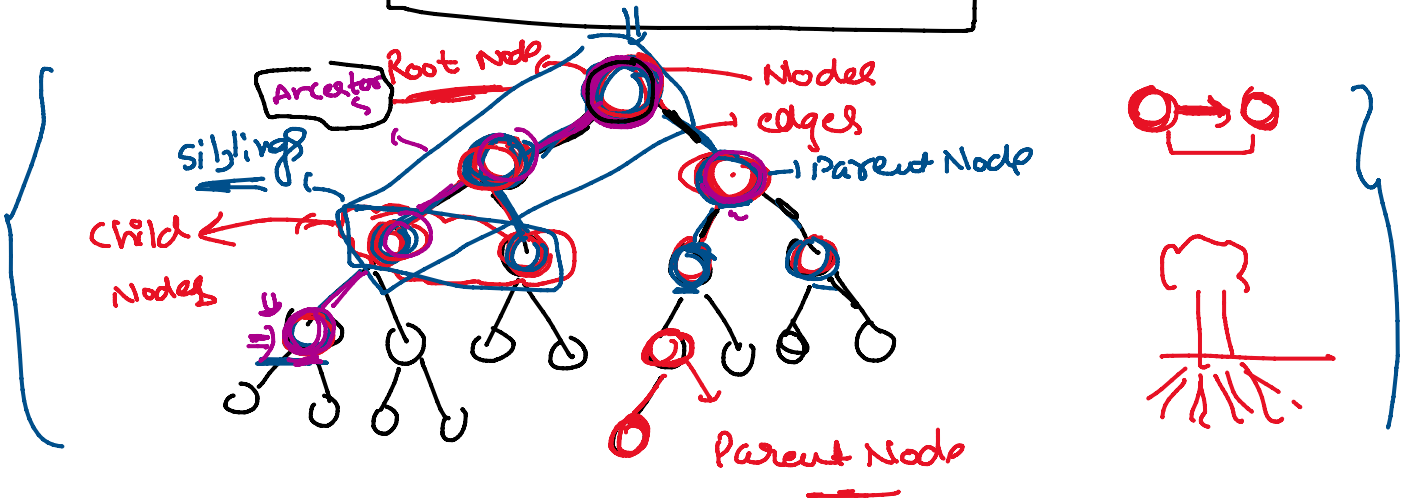


Recursive Tree

```
Void f()
{
    f()
    f();
}
```



=> Tree Datastructure



Node -

Edges

Root Node : Parent Node, children

Ancestor :

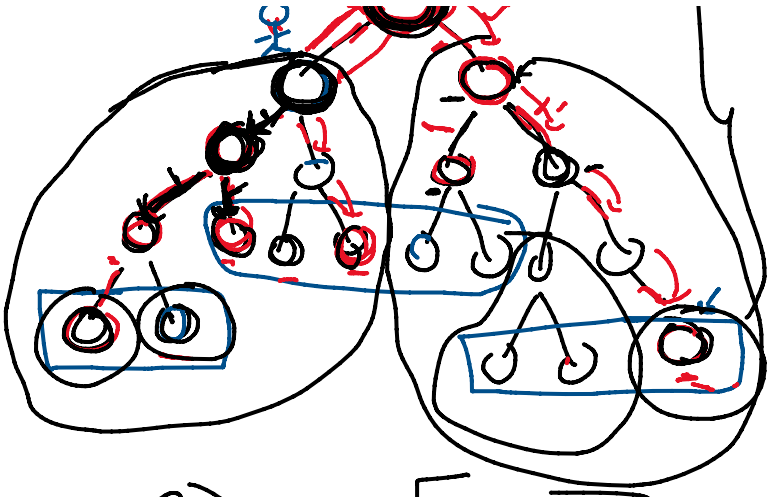
Descendents :

Path btw 2 Nodes

Descendants :-

Leaves

Path?



Path b/w 2 Nodes
=
sequence of edges b/w two Nodes

Height of the tree

Longest Path b/w root and a leaf Node

Degree of a Node - 3

Indegree + Outdegree

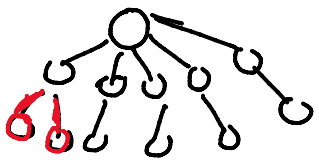
Depth of a Node ⇒ Height of that Node

Sub Tree

Types of Tree

[n-ary Tree

→ Generic tree



Max → 2 child

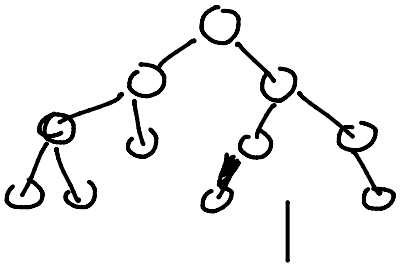
↳ Binary Tree

Max → 3 child

→ Ternary Tree

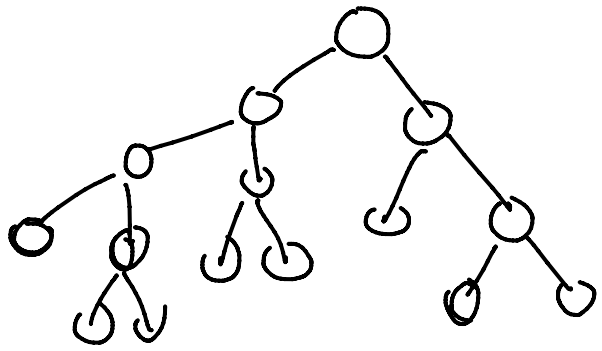
④ → 4-ary Tree

→ Binary Tree

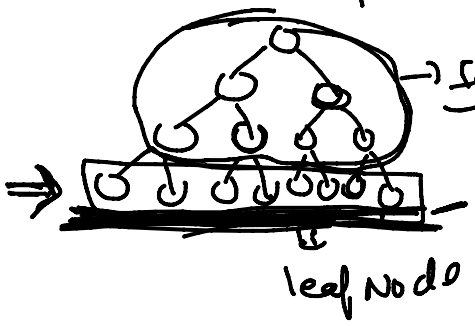


→ Full BT → every Node has 2 child or 0 child

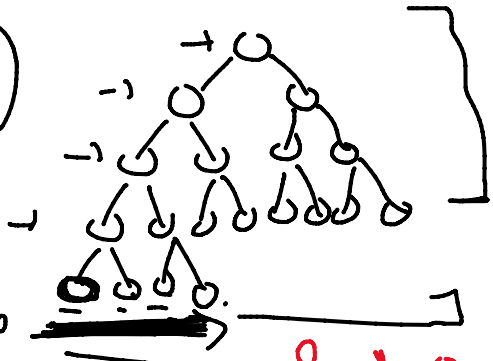
→ Full BT → Every Node has 2 child or 0 child



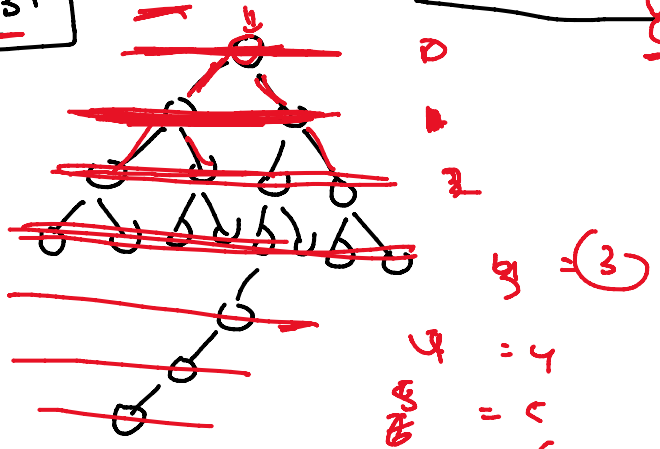
→ Perfect BT → Every Node has 2 child



→ Complete BT →



levels in BT ⇒ Deepest Node



0, 1, 0

Arrays / String

Recursion

DP

Graph - Tree

Graph - Tree

DSA

CP ⇒ 1700 - 1800

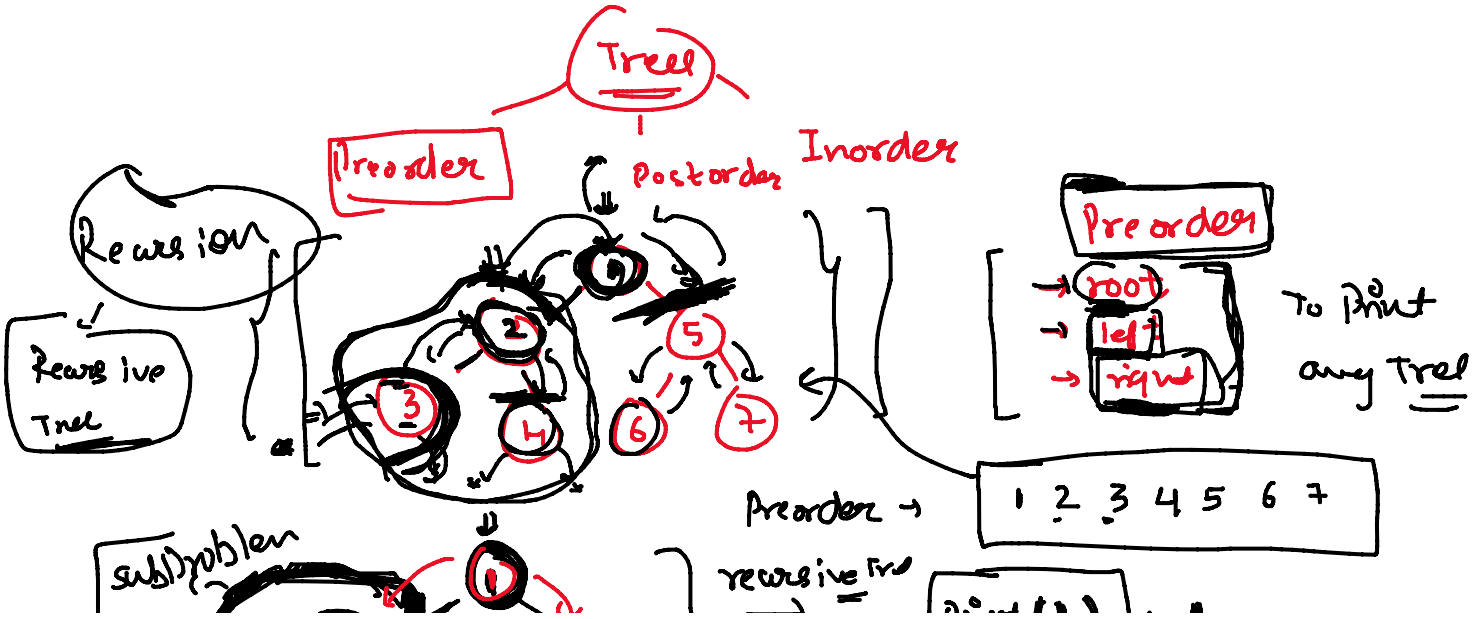
→ Hashing / sorting / set

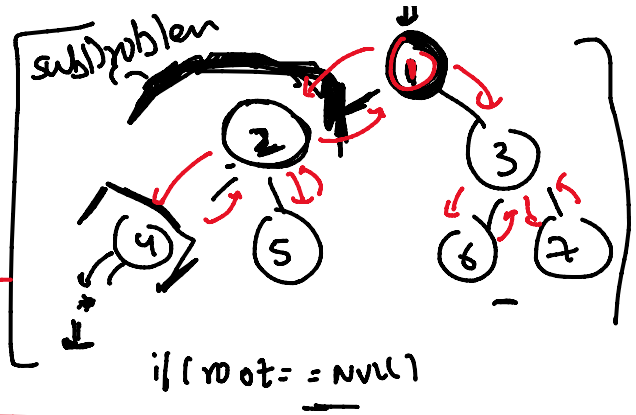
⇒

Structure of Node ?

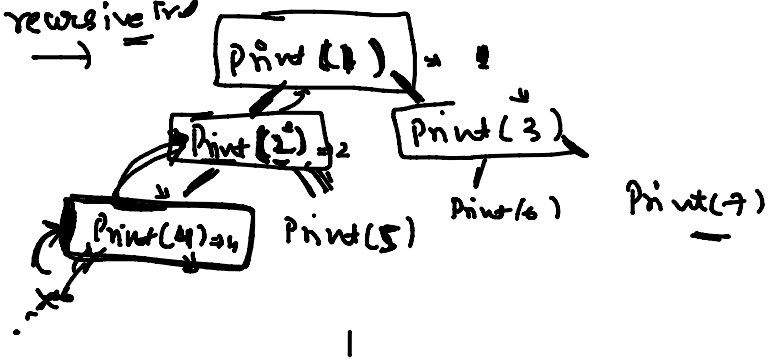

```
Class Node {
int data;
Node * left;
Node * Right;

Node(int d)
{ data = d;
  left = NULL;
  right = NULL;
}
};
```

Traversals in a Node





recursive fn

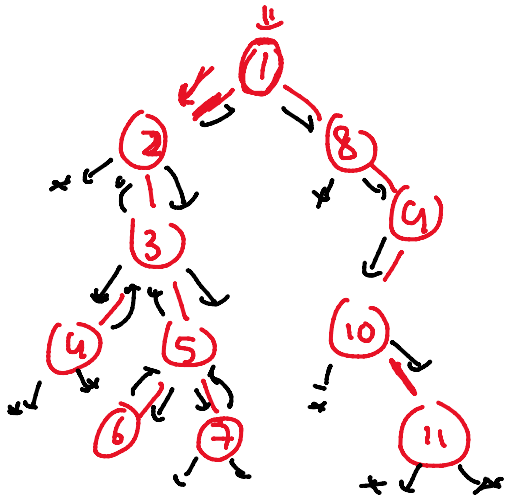


if (root == NULL)

1 2 4 5 3 6 7

```
void Print (Node * root)
{
    if (root == NULL)
        return;
    Cout << root -> data;
    Print ( root -> left);
    Print (root -> right);
}
```

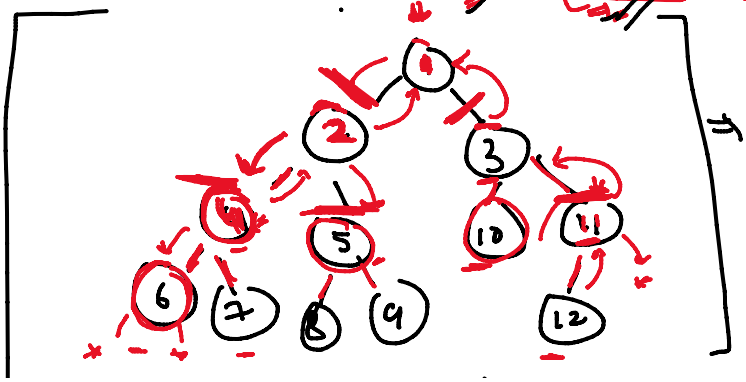
Pre-order traversal



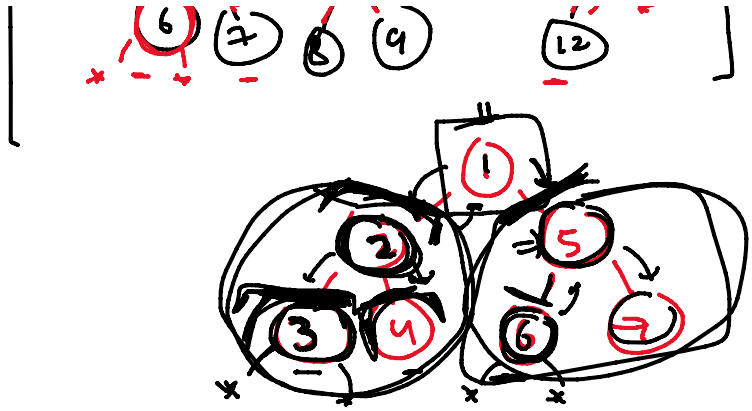
⇒ 1 2 3 4 5 6 7 8 9 10 11

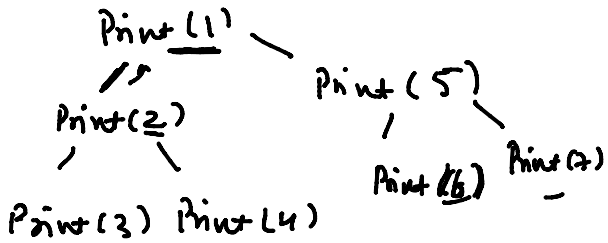
Post-order Traversal

⇒ left right root



⇒ 6 7 4 8 9 5 2 10 12 11 3 1

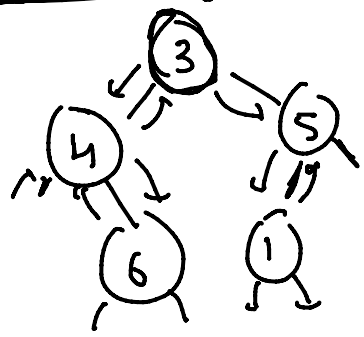




3 4 2 6 7 5 1

Preorder

Postorder

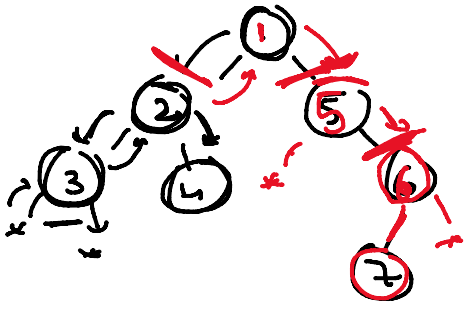


⇒ 3 4 6 5 1

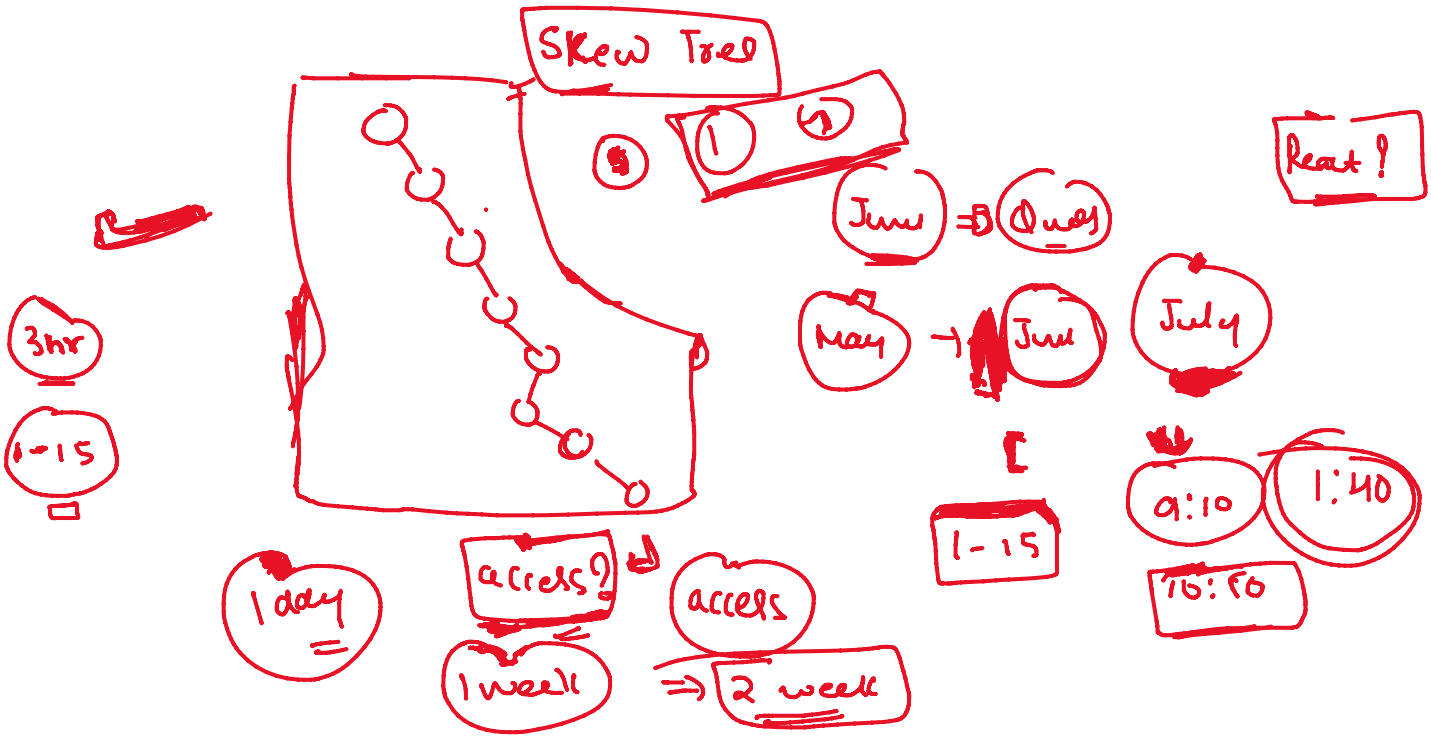
⇒ [3 4 -1 6 -1 -1 5 1 -1 -1 -1]

Inorder Traversal

left root right



3 2 4 1 5 7 6


Skew Tree
React ?
3hr
1-15
1 day
=
access?
1 week
access
⇒ 2 week
June
⇒ 1 day
May
→ June
July
1-15
9:10
1:40
16:50