Todays Content:

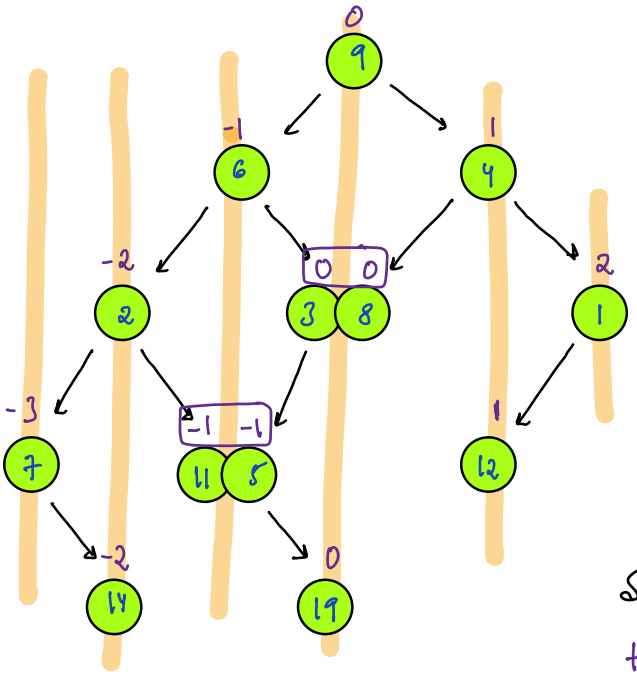
a) Vertical level order traversaal

- → top view
- → bottom view
- → diagnol view

b) Construct BT from inorder & preorder

c) Type of Trees → notes → next class, doubts sessiny

Vertical level order traversal (left → right)



Expected:

7

2 14

6 11 5

9 3 8 19

4 12

1

Storing data to print

hashmap<int, list<int>>

↳ level

-3 : 7

-2 : 2 14

-1 : 6 11 5

0 : 9 3 8 19

1 : 4 12

2 : 1

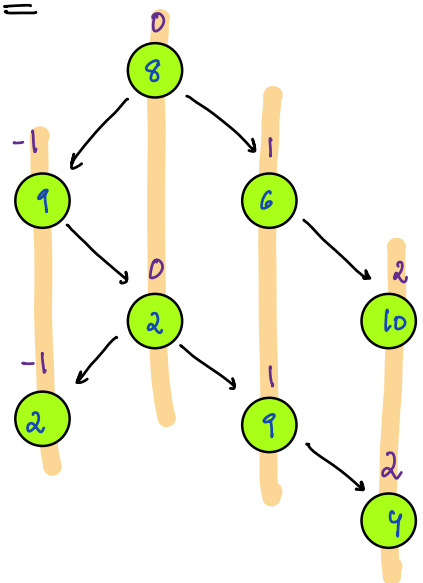
All node at given level

minlevel	maxlevel
-3	2

Iterate on all level from -3 to 2 &

print nodes level by level

En:



Hashmap

-1 : 9 2

0 : 8 2

1 : 6 9

2 : 10 4

Inorder Hashmap *

-1 : 9 2

0 : 2

1 : 9

2 : 4

preorder Hashmap *

-1 : 9 2

0 : 8 2

1 : 9

2 : 4

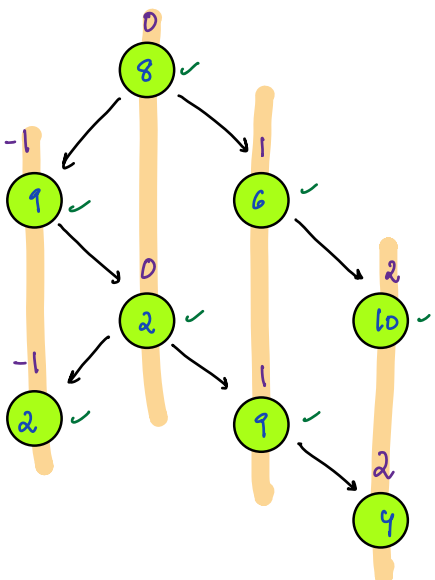
postorder Hashmap *

-1 :

0 :

1 :

2 :



Hashmap level order

~~8,0~~ | ~~9,-1~~ ~~6,1~~ | ~~2,0~~ | ~~10,2~~

~~2,-1~~ ~~9,1~~ ~~4,2~~

-1 : 9 2

0 : 8 2

1 : 6 9

2 : 10 4

```
void Vertical level ( Node root) { TC: O(N)  SC: O(N)
    Hashmap < int, list<int>> hm
    int minl = 0, manl = 0
    queue < pair<node, int>> q
        node ref ↵        ↳ vertical level of each node
    q. enque ( {root, 0y)
    while ( q.size() > 0) {
        pair Node, int > data = q.front()
        q.deque()
        Node t = data. first
        int  l = data. second
        hm [l] . insert ( t.data) // insert in hashmap
        minl = min ( l, minl)
        manl = man ( l, manl)
        if ( t.left != NULL) {
            q. enque ( {t.left, l-1} )
        }
        if ( t.right != NULL) {
            q. enque ( {t.right, l+1} )
        }
    }
    i = minl; i <= manl; i++) {
        // All nodes at vertical level = i = hm[i]
        // print list hm[i] { TODO3
        // print 1st element hm[i] { Top view}
    } // print last element hm[i] { Bottom view}
}
```

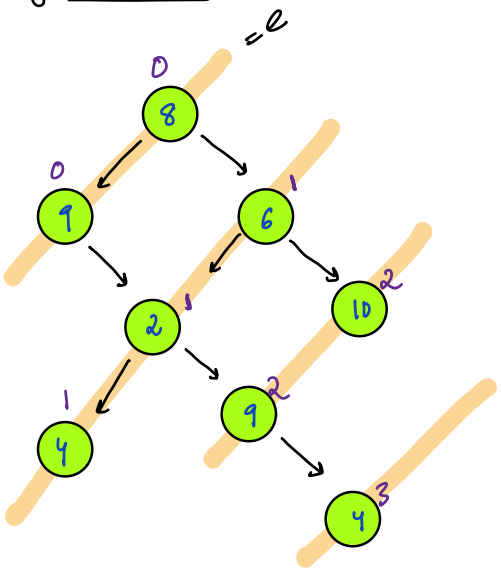
Obs:

Top view: 1st node in every vertical level

Bottom view: last node in every vertical level

{ Easier approach in doubts session }

Diagnal Traversal {R-L}



output :

0 : 8 9
1 : 6 2 4
2 : 10 9
3 : 4

obs:

if we goto left: level same

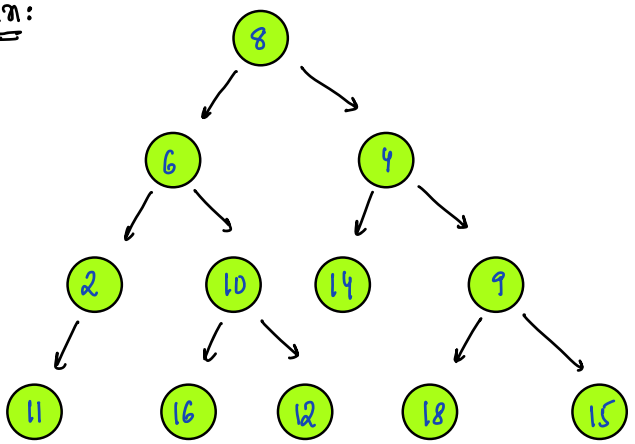
if we goto right: level by 1

1 queue on every level = diag {R-L}

Diagnal Traversal {L-R} TODO

2Q) Given preorder & inorder of BT of distinct values
print postorder traversal

Eg:



Idea: Construct Tree + Traversal ✓

preorder [ ]: (D L R)

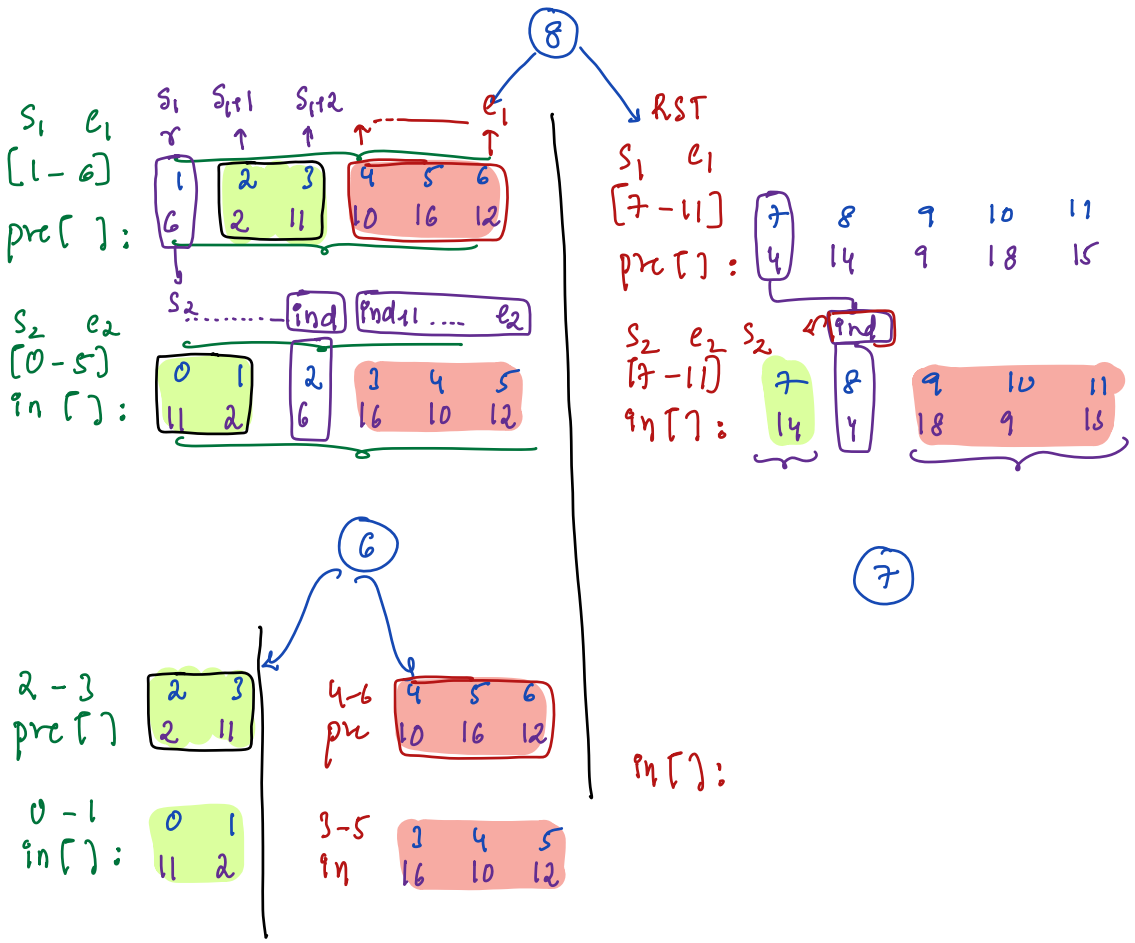
0	1	2	3	4	5	6	7	8	9	10	11
8	6	2	11	10	16	12	4	14	9	18	15

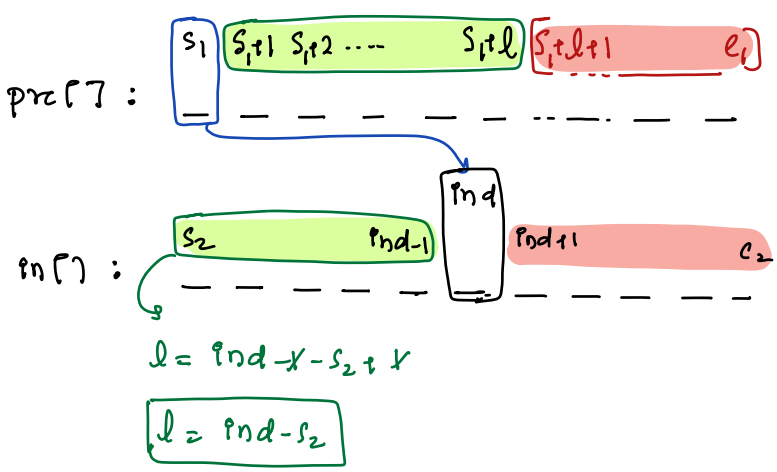
inorder [ ]: (L D R)

0	1	2	3	4	5	6	7	8	9	10	11
11	2	6	16	10	12	8	14	4	18	9	15

left sub tree

right sub tree


8
RST
7
6
pre[ ]:
in[ ]:


pre[ ]:
in[ ]:
ind
ind+1
l = ind - x - s2 + x
l = ind - s2

Ass: Given preorder & inorder construct tree & return root node

```
Node Tree(int pre[], int s1, int e1, int in[], int s2, int e2)
    // no data
    if(s1 > e1) { return null }

    Step:1 // root node is pre[s1]
    Node root = new Node(pre[s1])

    Step:2 // search for pre[s1] in in[s2 .. e2]
    int ind = ____
    i = s2; i <= e2; i++){
        if(pre[s1] == in[i]) { ind = i, break }
    }

    Step:3 // Splitting
    l = ind - s2 + 1 = ind - s2 {# no: of in LST}
    LST: → pre[s1+1  s1+l] → in[s2, ind-1]
    RST: → pre[s1+l+1, e1] → in[ind+1, e2]

    // Construct LST & return root of LST
    root.left = Tree(pre, s1+1, s1+l, in, s2, ind-1)

    // Construct RST & return root of RST
    root.right = Tree(pre, s1+l+1, e1, in, ind+1, e2)

    return root;
}
// After Tree Construction print postorder
// Add in[] in hashmap in main & pass it as parameter in Tree()
```

2 nulls

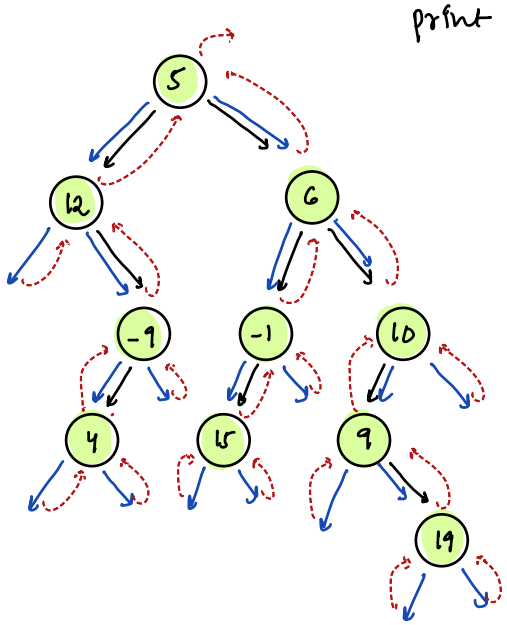
For each search ⇒ O(N)
Repeat search for every elem in pre[] ⇒ TC: O(N²)

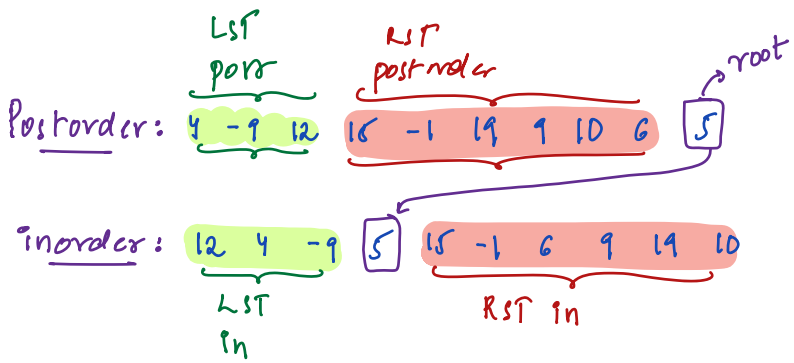
Optimize by storing in[] elements & its index pos in hashmap:
TC: O(N*1 + N) SC: O(N)

Ass: Given preorder & inorder, print postorder

```
void Tree( int pre[], int s1, int e1, int in[], int s2, int e2)
{
    // no data
    if( s1 > e1) { return; }

    Step:2 // search for pre[s1] in in[s2 .. e2]      // using hashmap for optimization
    int ind = ____
    for( i = s2; i <= e2; i++){
        if( pre[s1] == in[i]) { ind = i, break }
    }

    // print postorder LST
    Tree( pre, s1+1, s1+l, in, s2, ind-1)

    // print postorder RST
    Tree( pre, s1+l+1, e1, in, ind+1, e2)

    print( pre[s1]) // root node
}
```

4Q) Given in[] & post[] construct tree[] } TODO
print preorder[]





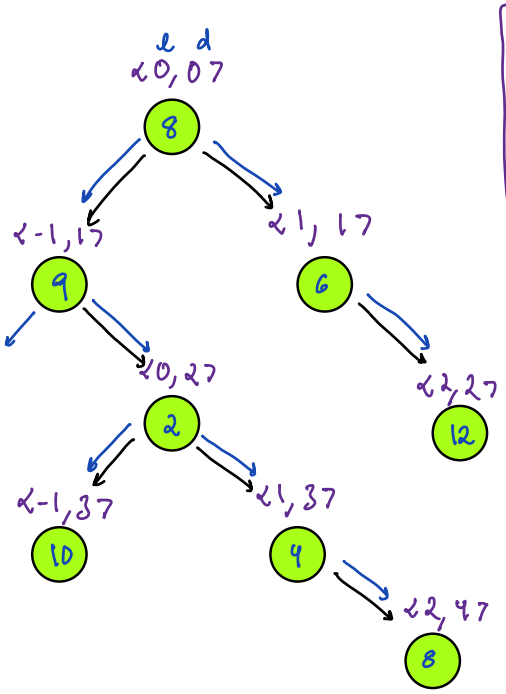
5Q) Given pre[] & post[] construct tree[] ?

Postorder: 4 -9 12 15 -1 19 9 10 6 5

Preorder: 5 12 -9 4 6 -1 15 10 9 19

Note: We cannot separate LST & RST hence
Tree construction is not possible

Top view easy approach:



hash <int, pair<int, int>> hm

-1	<9,1>	<10,3>
0	<8,0>	~~<7,2>~~
1	~~<4,3>~~	<6,1> replace
2	<8,4>	<12,2> replace

root data, depth

vertical level

hashmap<int, pair<int, int>> hm

minL = 0, maxL = 0

any traversal is fine

```
void preorder(Node root, int vl, int d){
    if(root==null){return;}
    minL = min(minL, vl)   maxL = max(maxL, vl)
    if(hm.search(vl)==false){
        hm[vl] = {root.data, d};
    }
    else{                    1st  2nd
        pair<int, int> data= hm[vl]
        int cd = data.second
        if(d<=cd){//update hm   → if(d>cd){ bottom view
            hm[vl] = {root.data, d};
        }
    }
    preorder(root.left, vl-1, d+1) // vl = diagnol view L-L
    preorder(root.right, vl+1, d+1)
}
```

Iterate from minL to maxL on hashmap & print it