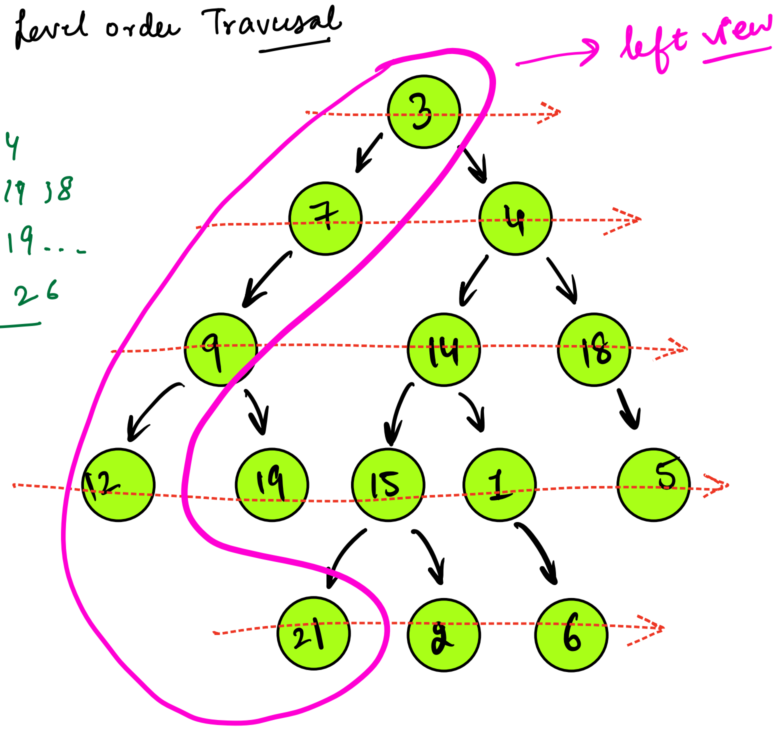
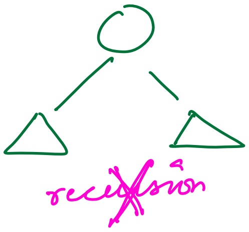
Level order Traversal

3
7 4
9 19 18
12 19 . . .
21 2 6



3 7 4 9 14 18
12 19 15 1 5
21 2 6



~~3~~ ~~7~~ ~~4~~ ~~9~~ ~~14~~ ~~18~~ ~~12~~ ~~19~~ 15 1 5 21 2

insertion at rear
removal at front
⇓ queue

~~3~~ NULL ~~7~~ ~~4~~ NULL 9 14 18 NULL

```
queue < Node > q;

T.C : O(n)
S.C :
O(max no. of node at any level)
⇓
O(N)

q.enque( root);            q.enque( NULL);
while ( q.size() >1 )
{
    Node temp = queue.front();
          q.dequeue();           if ( temp == null)
                                 {
                                    print ("\n");
                                    q.enque (null);
                                    continue;
                                 }
       print ( temp.data);
    if( temp.left != null) {   q. enqueue (temp.left)}
    if( temp.right != null) {   q. enque (temp.right)};
}
```

- Right to left :- interchange the enqueue process of left & right

- Left view of the tree :- first node of every level

  every first node will be inserted in the queue after NULL

  ⇓

  maintain a prev variable

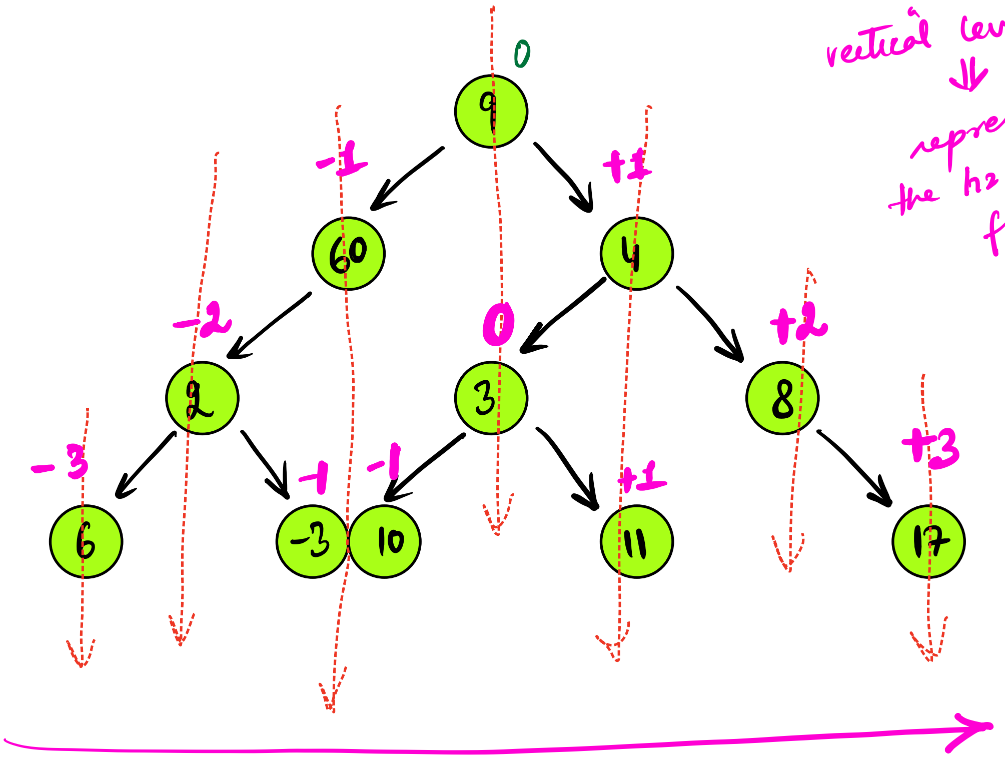
  ⇓

  what was the last node!

- Right view :- right to left

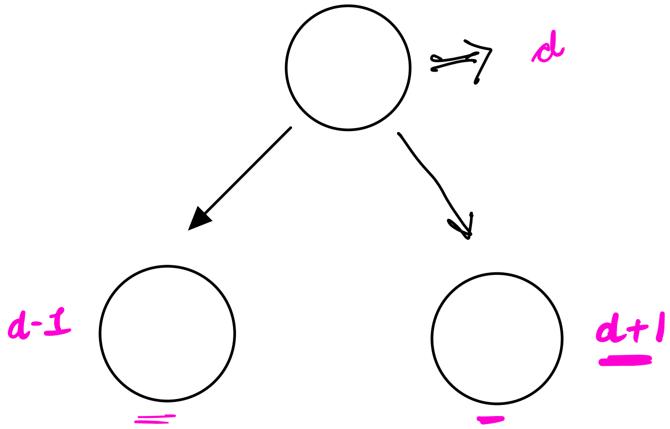
  └ first node

vertical level order traversal



vertical level
⇓
represents the hz distance from root

6 2 6 -3 10 9 3 4 11 8 17



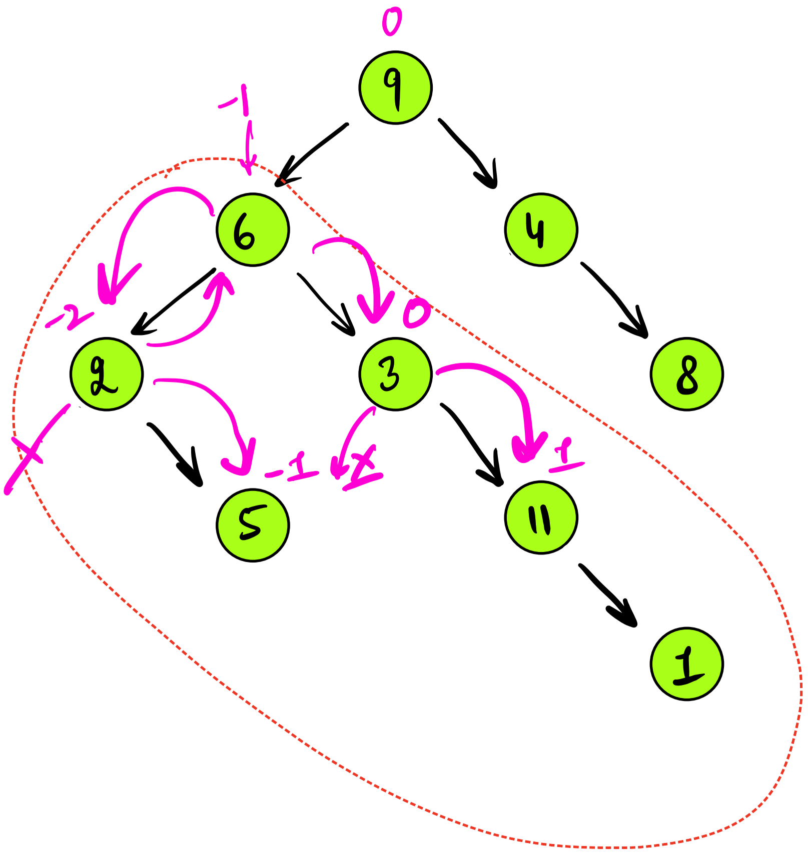
level → Nodes?

↓

HM

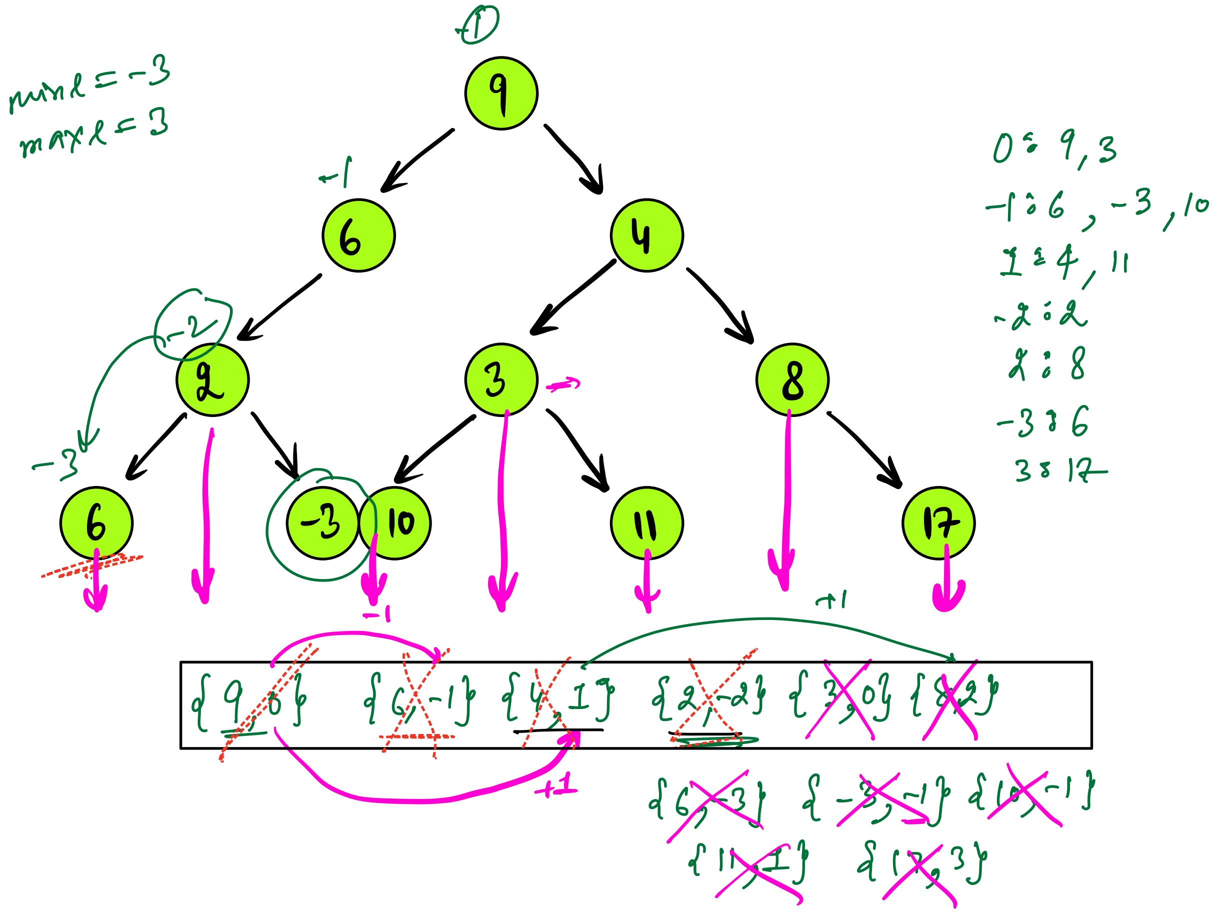
< int, list<Node>>

preorder:- Root left right



0 : 9, 3
-1 : 6, 5
-2 : 2
1 : 11

Level order Traversal



minl = -3
maxl = 3

0 : 9, 3
-1 : 6, -3, 10
1 : 4, 11
-2 : 2
2 : 8
-3 : 6
3 : 17

```
HM < int , list <Node> > hm;                 maxl = 0
queue <pair< Node , int > > q;               minl = 0;

   q. enque( {root, 0})
  while (            )
  {
 <Node,int> temp = q.front()
          // get your node & level     - level
                                      → { maxl = max(maxl,
                                                         level)
          // insert the node in HM          minl = min(     ); }
         // put your left child in queue
                                    level - 1
         // put your right child in queue
                                  level + 1;
  }
  for( minl → maxl)
   {      // get the list from HM
             // print the list
   }
```

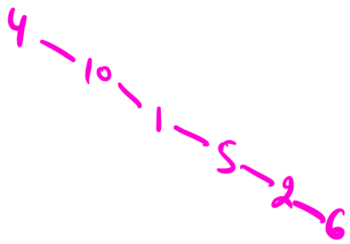
top view → first nodes of every level

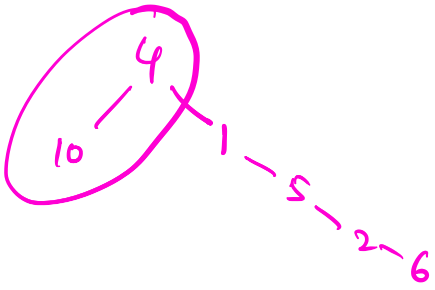
bottom view → last nodes of every level

preorder — Root L R

4 10 1 5 2 6

root of tree





post : 4 10 1 5 2 6 → root

L Root Right

inorder : 4 10 1 5 2 6

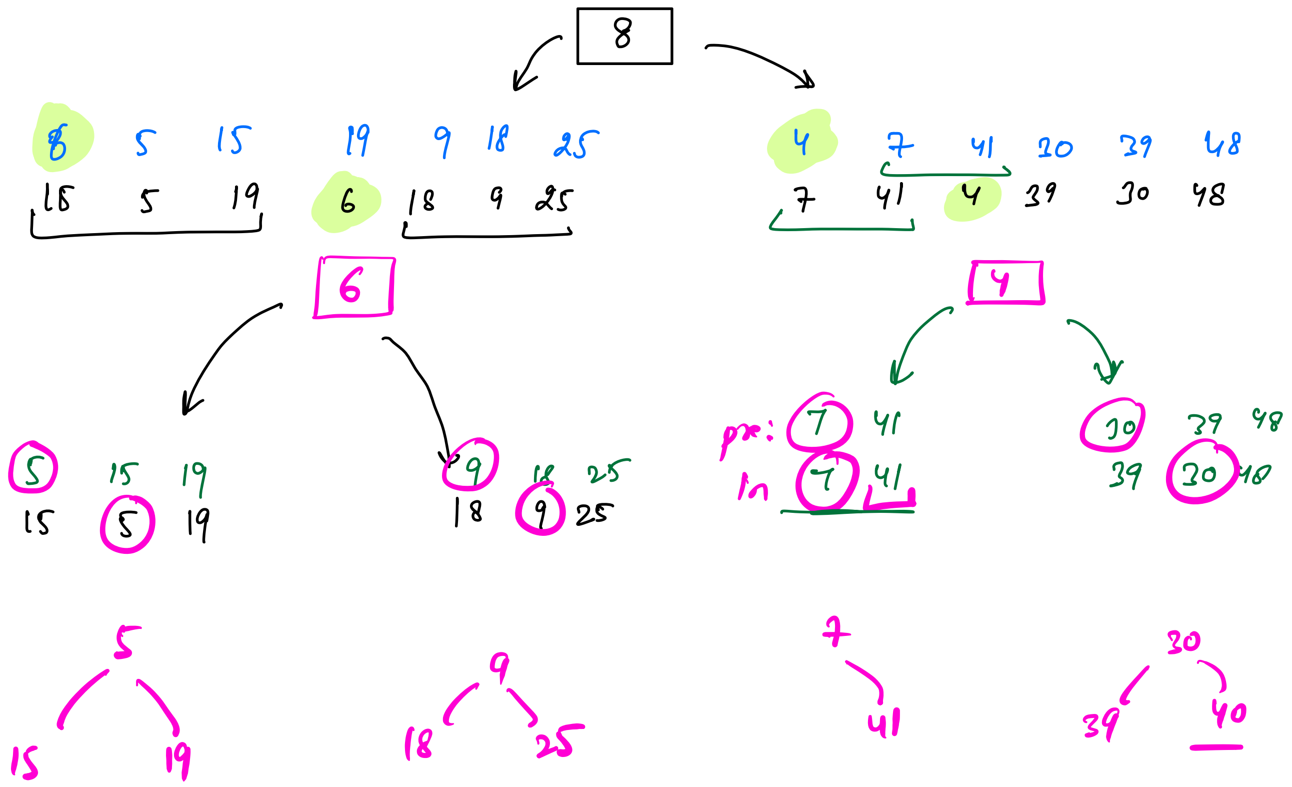
LST root RST

destinct

root

Root L R

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Pre:	8	6	5	15	19	9	18	25	4	7	41	30	39	48
Inorder:	15	5	19	6	18	9	25	8	7	41	4	39	30	48



Find root from preorder

⇓

search root in inorder

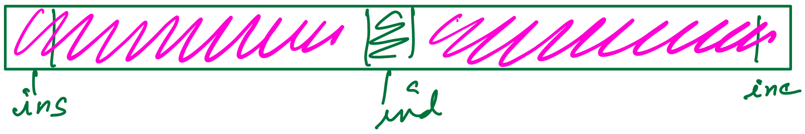
⇓

divide your LST & RST in both
pre & inorder

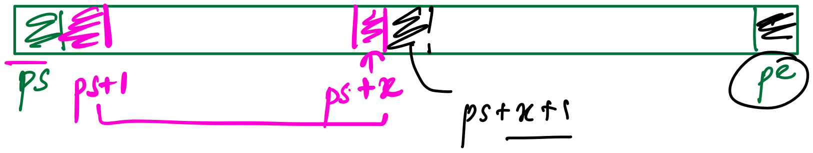
construct will creat the tree on the basis of ino/pre & will retu th root

```
Node construct( int pre[], int ps, int pe, int in[], int ins,
                                                     int ine)
{
          if( ps > pe) return NULL;

          Node temp = new Node( pre[ ps]);
          int ind = -1;
          for( int i = ins ; i<= ine ; i++)
          {
                   if( in[i] == pre[ps])
                   {      ind = i;
                          break;
                   }
          }                                          HM
```



```
               x = ind - ins;
```



```
     temp.left  = construct( pre, ps+1, ps+x, in, ins, ind-1);
     temp.right = construct( pre, ps+x+1, pe, in, ind+1, ine);
       return temp;
}
```

T.C: $O(n^2)$ — $O(n)$ — HM

postorde + inorder

~~post + pre~~