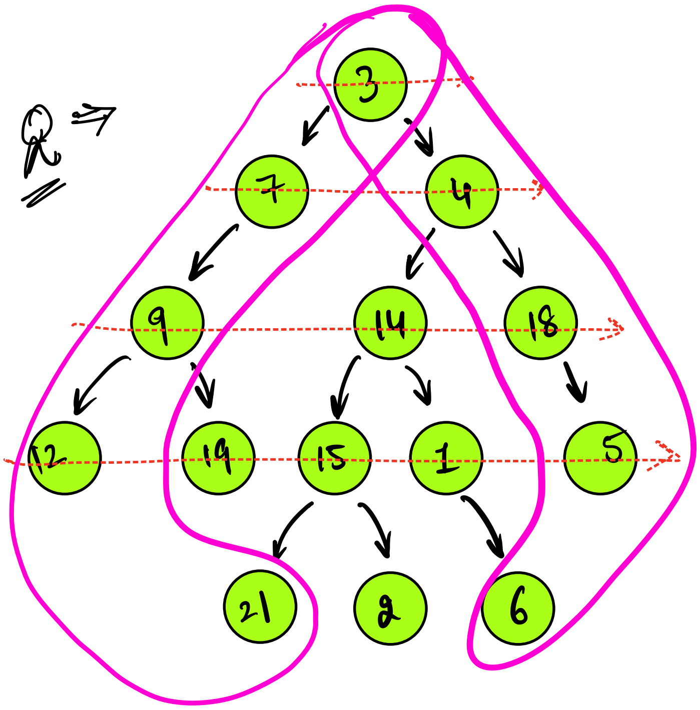
Level order Traversal



left to right

3 7 4 9 14 18

12 19 15 1 5 21 2 6

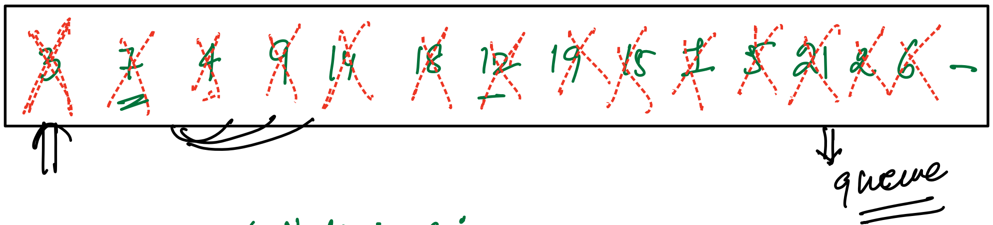
~~Recursion~~

3 \n

7 14 \n

9 14 18 \n

12 19 --

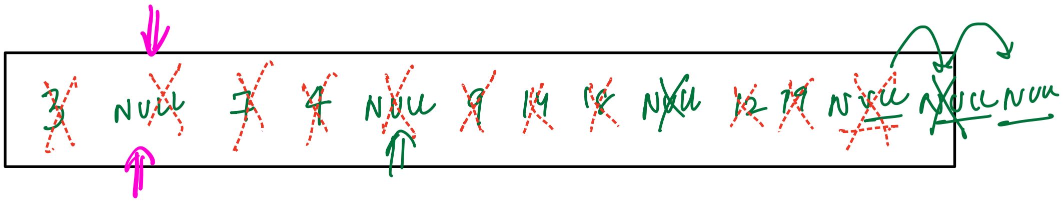


```
queue < Node > q ;
  q. enqueue( root);

while ( !q.empty())
{
   Node temp = q.front();
               q. dequeue;
            print( temp.data);
   if( temp.left != null ) { q. enquee ( temp.left);}
   if ( temp. right != null ) { q. enquee ( temp.right);}
}
```

T.C: O(n)

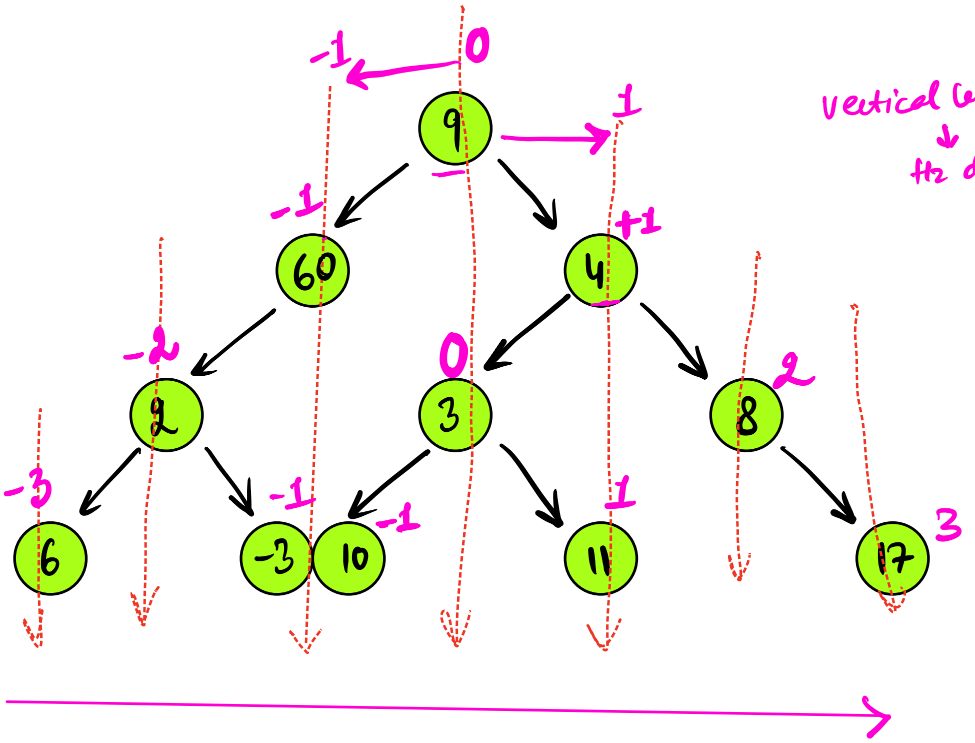
S.C: O(n)



```
queue < Node > q ;
   q. enqueue( root);                prev = NULL;
      q. enque( null);
  while (   q. size() > 1 )
  {
    Node temp  =  q. front();
                 q. dequeue;  →  if ( prev = = NULL)
                                    temp is part of left view
prev = temp;  ←
              if( temp == NULL)
                {      print( "\n"); q. enque (NULL);
                    continue; }
             print( temp.data);
  if ( temp.left != null ) {  q. enque ( temp.left); }
   if ( temp.right != null ) {  q. enquee ( temp.right); }
 }
```

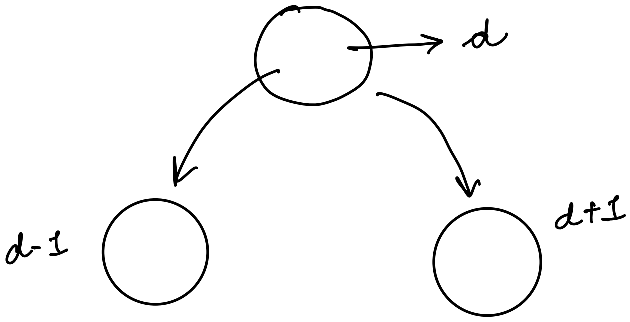
- Right to left :- change the order of insertion of children.
- Left view   first node of every level
- Right view :-   Level order from R→L.
  ↓
  first node of the level

vertical level order traversal



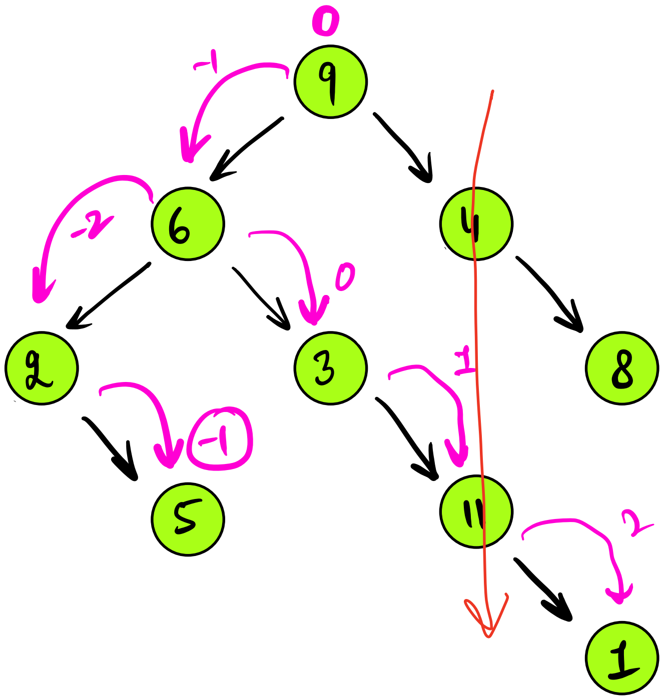
vertical level
↓
the dist from root

6 2 60 -3 10 9 3 4 11 8 17



Hashmap
↓
key , value
↓ ↓
level (int) list of nodes

preorder



0 : 9 , 3

-1 : 6 , 5

-2 : 2

+1 : 11

2 : 1

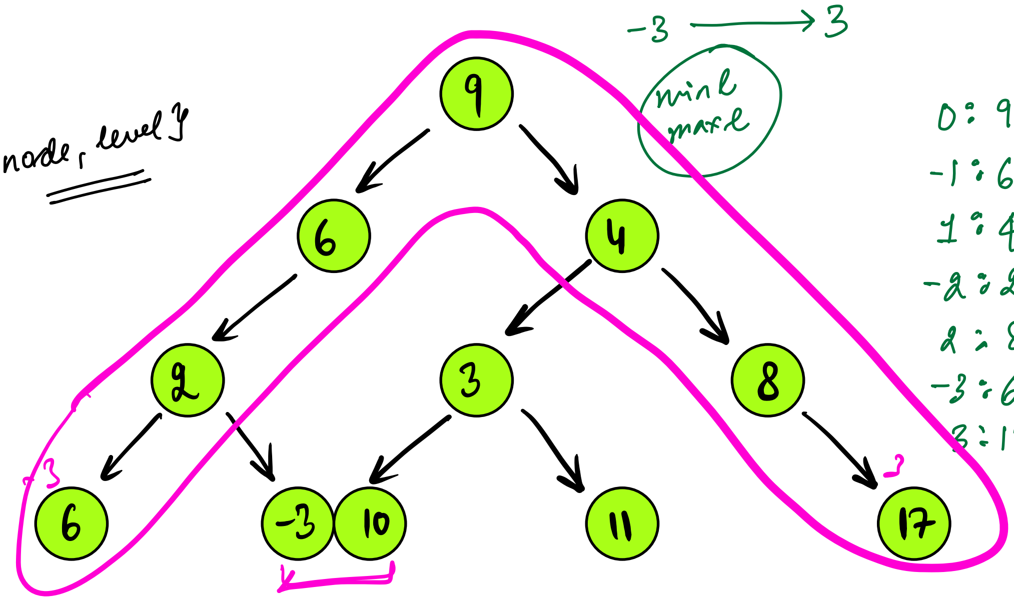
~~pre~~

~~post~~

~~in~~

level order

{node, level}



HL

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

~~{9, 0}~~ ~~{6, -1}~~ ~~{4, 1}~~ ~~{2, -2}~~ ~~{3, 0}~~ ~~{8, 2}~~

~~{6, -3}~~ ~~{-3, -1}~~ ~~{10, -1}~~, ~~{11, 1}~~ ~~{17, 3}~~

```
HM < int, list<Node>> hm;
 queue <    Node, int  > q;

      q.enque({root, 0});

      while(          )
     {
      {curr, level} = q.front();  q.deque();
           // insert curr in HM[level]
           if(curr.left != null) { q.en({curr.left, level-1});
           if(curr.right != null) { q.en( {curr.right, level+1});
              minl = min(minl, level);
              maxl = max(maxl, level);
      }
```

```
for( i = minL → maxL )
{
        get the list from HM[i]
          // traverse the list
}
```

Top-view :- first node of every vertical level

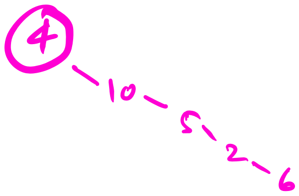
Down-view :- last node of every vertical level

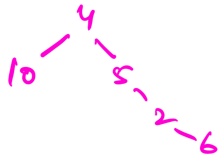
Bottm

• preorder Root left Right

4 10 1 5 2 6

root





postor

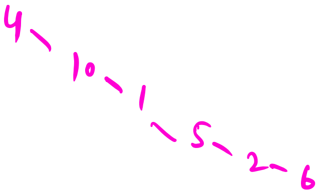
4 10 1 5 2 6

root

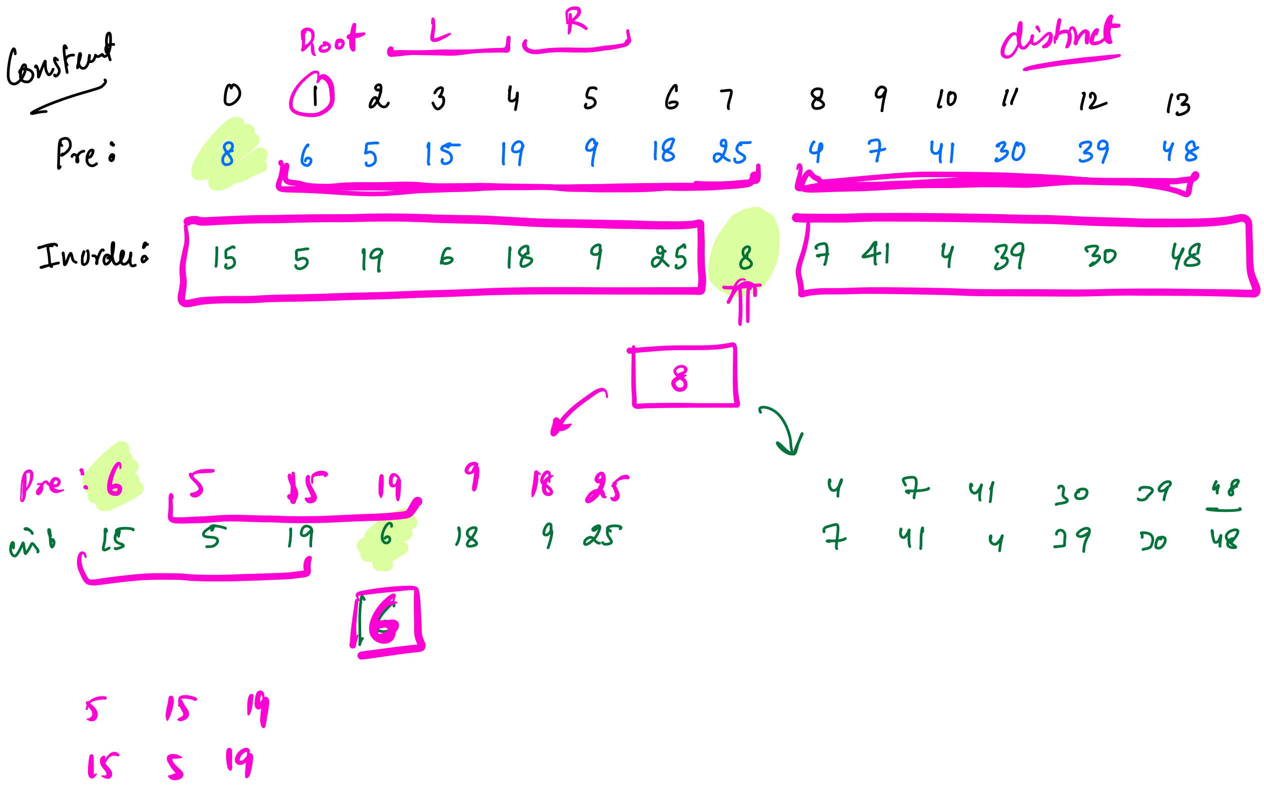
inorder

root

4 10 1 5 2 6



5



Construct

	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

8

Pre :	6	5	15	19	9	18	25
in	15	5	19	6	18	9	25

4	7	41	30	39	48
7	41	4	39	30	48

6

5	15	19
15	5	19

identify your root from preorder

⇓

search root in inorder

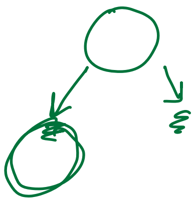
⇓

distribute your LST & RST

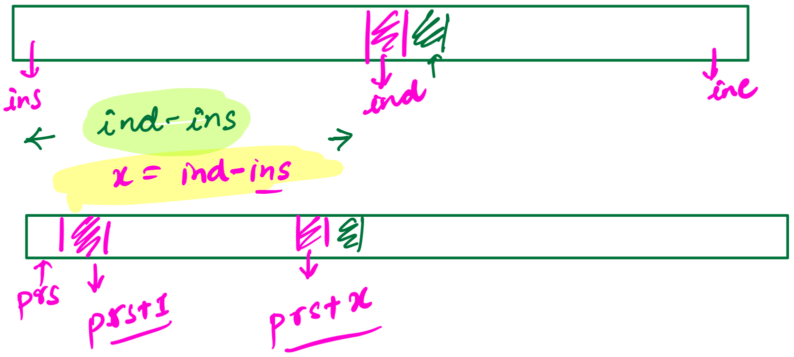
```
Node construct ( int pre[], int prs, int pre, int in[], ins, ine)
{
    if( prs > pre) return NULL;

    Node temp = new Node(pre[prs]);

    int ind;
    for( i = ins → ine)
    {
        if( in[i] == pre[prs])
        {       ind = i;
                break;
        }
    }
```



O(1)
↑
HM
↓
<element, index>
|
inorder



T.C: $O(n^2)$
↓
$O(n)$

S.C: $O(n)$

```
    temp.left  = construct(pre, prs+1, prs+x, in, ins, ind-1)
    temp.right = construct(pre, prs+x+1, pre, in, ind+1, ine)

    return temp;
}
```

pre
post

LST ~~RST~~