Bottom View of Binary Tree.

Refers to the nodes visible when the tree is viewed from the bottom. The bottom view consists of the last node at each horizontal distance when traversed from top to bottom.

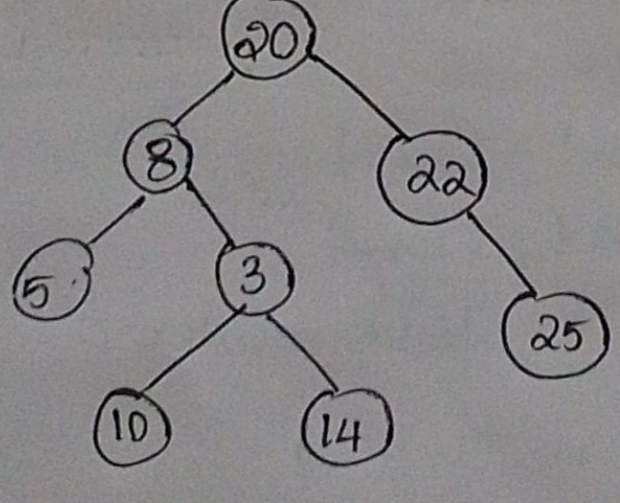
Algorithm:

① Using Breadth-First Search with Horizontal Distance (HD)

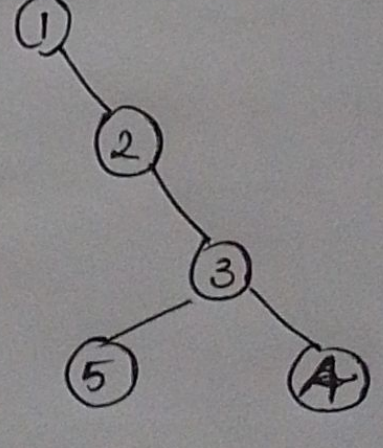
- Assigning a HD to each node
  - a) Root starts at HD = 0
  - b) Left child gets HD − 1
  - c) Right child gets HD + 2
- Store nodes in a queue as (node, HD).
- Use a map (dict in Python) to store the last node encountered at each HD. (highest node value).

② Processing Nodes:

- Traverse nodes level-wise using a queue
- for each node, update the map(dict) ← the latest node at that HD
- The bottom view consists of the values stored in the map, sorted by HD.



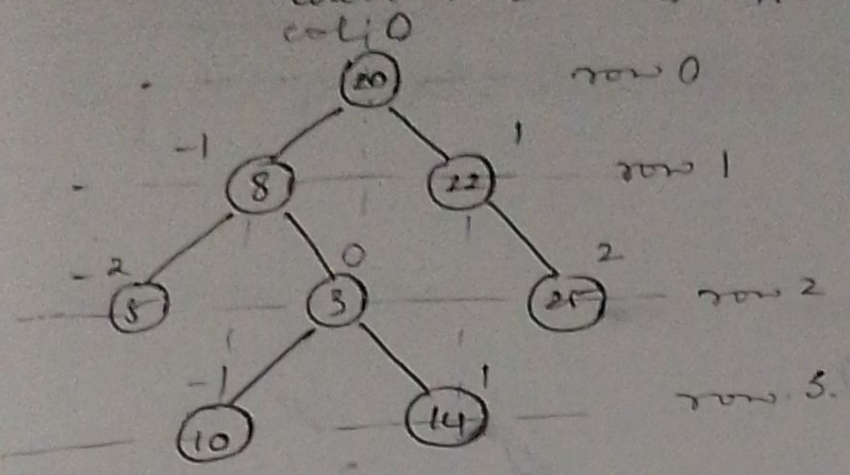
5, 10, 3, 14, 25 ✓



5, 4 ✗

4, 3, 5 & 1 —

Ccode Each tree has rows & columns.



from col 0
for left - subtract -1 if to
for right - add 1

Here bottom of the tree means. for each column find the node with the largest row value.

i.e for column '0' we have '20' (row 0) & '3' (row 2)

column '-1' ⇒ '8' (row 1), '10' (row 3)

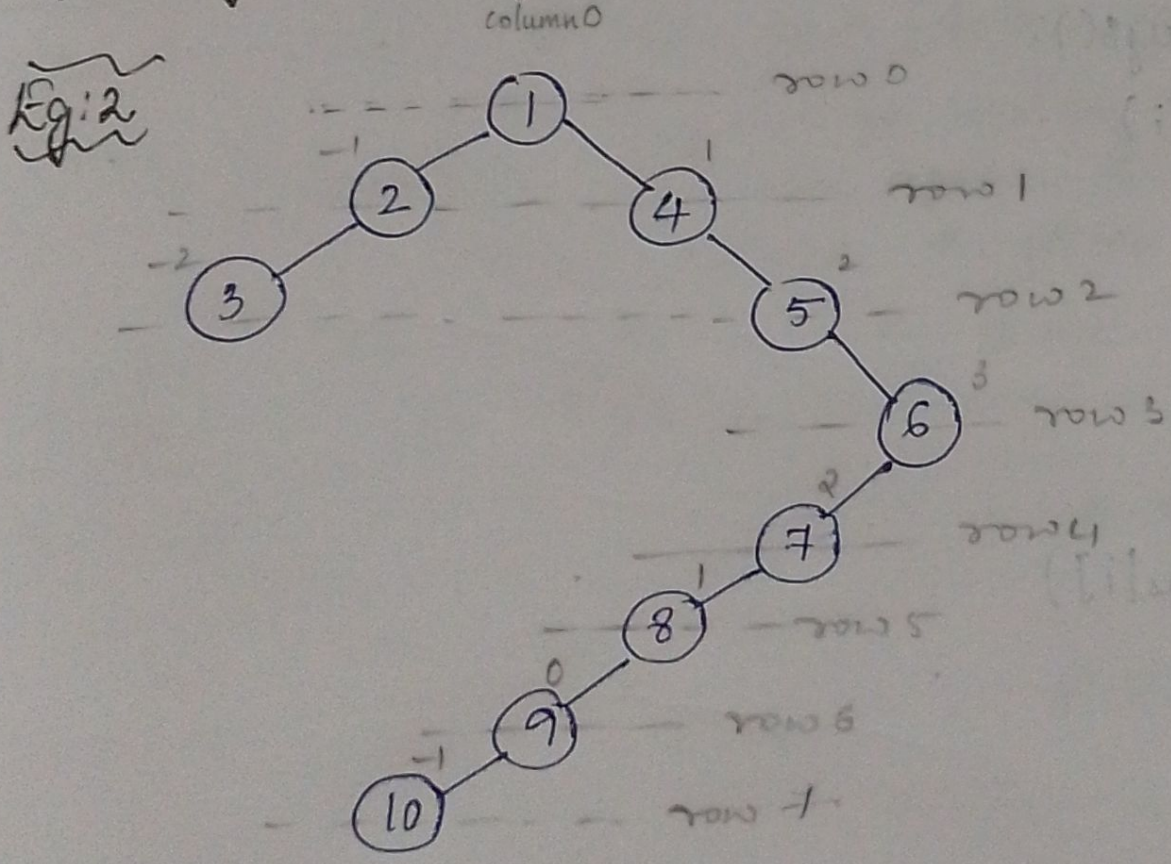
column '-2' ⇒ '5' (row 2)

column '1' ⇒ '22' (row 1), '14' (row 3)

column '2' ⇒ '25' (row 2)

printing from left to right of the tree hence 5, 10, 3, 14, 25.

Eg:2



col '0' ⇒ '1' (row 0)
'9' (row 6)

col '-1' ⇒ '2' (row 1)
'10' (row 7)

col '-2' ⇒ '3' (row 2)

col '1' ⇒ '4' (row 1)
'8' (row 5)

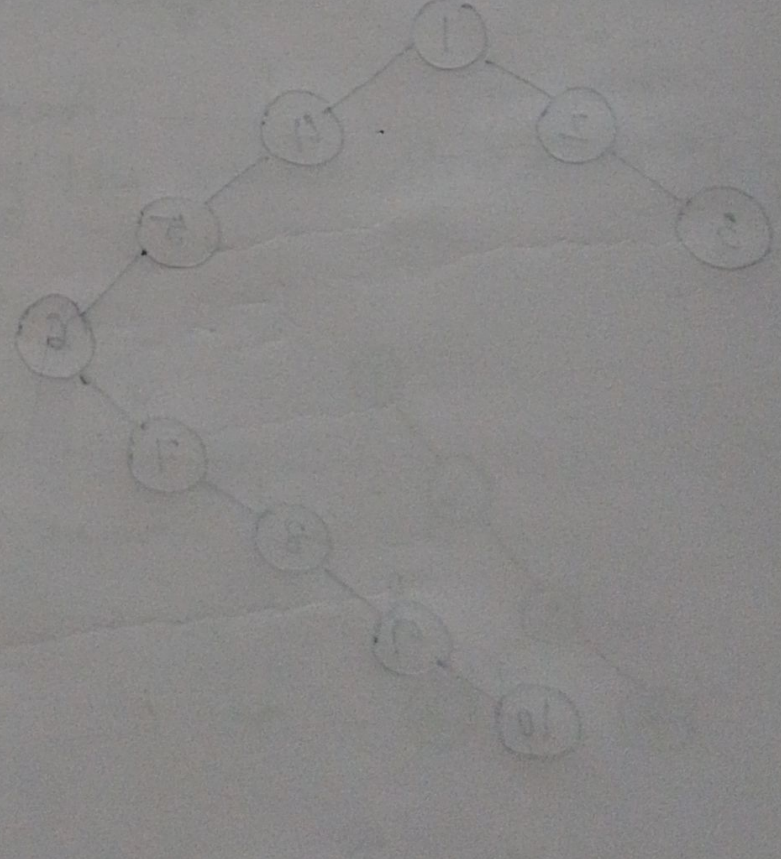
col '2' ⇒ '5' (row 2)
'7' (row 4)

col '3' ⇒ '6' (row 3).

So we print left to right

[3, 10, 9, 8, -1, 6]

```
class Solution:
    def bottomView(self, root):
        queue = [[root, 0]]
        ans = {}
        while queue:
            node = queue.pop(0)
            r = node[0]
            val = node[0].data
            col = node[1]
            ans[col] = val
            if r.left:
                queue.append([r.left, col - 1])
            if r.right:
                queue.append([r.right, col + 1])
        keys = []
        for i in ans.keys():
            keys.append(i)
        keys.sort()
        a = []
        for i in keys:
            a.append(ans[i])
        return a
```



Code breakdown

FIFO.

Key components: Data structures used. Queue for BFS

stores [node, horizontal distance]

Dictionary (Map) : ans. stores the last (bottom most) node at each HD.

1) Initialize Queue & dictionary.

queue = [[root, 0]]

ans = { } ie { HD : node_value (element) }

2) BFS Traversal (level Order)

removing an element from the front of the queue

while queue:

node = queue.pop(0) # Dequeue the front node

r = node [0] # Extract the node

val = node[0].data # Get the node's value

col = node [1]. # Get the HD ie col index.

3) Update Bottom View

ans [col] = val # Store last (bottom-most) node @ each column.

* why every time? . cuz we go level by level

. if a node appears @ later time & same HD, it replaces the previous one (since it's lower in the tree)

4) Enqueue Left & Right Children

adding element to the end of the queue.

if r.left

Move left → decrease col

move right → ↑ col

5) Extract & Sort Bottom View

Here we sort the column indices since we need the output from left to right.

⑥ Build the final Output

Now that keys are sorted, we add their corresponding values to the final answer.

⑦ Return the Result

return a

TID

Step: 1 Calculating the Horizontal Distance

Node	Column (col)
20	0
8	-1
22	+1
5	-2
3	0
25	+2
10	-1
14	+1

Step: 2 Last seen @ each column

Column	Last seen node
-2	5
-1	10
0	3
1	14
2	25

Time complexity = O(N)
Space complexity = O(N).

final Output [5, 10, 3, 14, 25]