Bottom View of Binary Tree - Stepwise Algorithm & Pseudocode

Stepwise Algorithm for Bottom View of a Binary Tree:

1. Define a helper function `width(root, pos, Lmin, Rmax)`:
• Traverse the tree recursively to calculate the minimum and maximum horizontal distance.
• Lmin stores the leftmost distance, Rmax stores the rightmost distance.

2. Define the main helper function `View(root, pos, result)`:
• Use a queue of pairs where int represents the horizontal distance.
• Push the root with its horizontal index (starting as offset of abs(Lmin)).
• While queue is not empty:
- Pop the front node and index.
- Update result[index] = current node's data (always overwrite to ensure bottom-most node remains).
- Push left child with index-1.
- Push right child with index+1.

3. In the main function `bottomView(root)`:
• Compute Lmin and Rmax using `width` function.
• Calculate total size = (Rmax - Lmin + 1).
• Initialize result vector with that size.
• Call `View(root, abs(Lmin), result)`.
• Return the result vector containing the bottom view from left to right.

Time Complexity:
• O(N), where N is the number of nodes (each visited once).
Space Complexity:
• O(W), where W is the width of the tree (queue + result storage).

```
// Pseudocode in given coding style

void View (Node *root , int pos , vector<int> &result ){
    queue<pair<Node*,int>> q;
    q.push({root, pos});

    while(!q.empty()){
        Node *Temp = q.front().first;
        int idx = q.front().second;
        q.pop();

        result[idx] = Temp->data;   // overwrite for bottom-most

        if(Temp->left) q.push({Temp->left , idx-1});
        if(Temp->right) q.push({Temp->right , idx+1});
    }
}

void width(Node * root , int pos , int &Lmin , int &Rmax){
    if(!root) return;
    width(root->left , pos-1 , Lmin , Rmax);
    Lmin = min(Lmin , pos);
    width(root->right , pos+1 , Lmin , Rmax);
    Rmax = max(Rmax , pos);
}

class Solution {
  public:
    vector<int> bottomView(Node *root) {
        int Lmin=0, Rmax=0;
        width(root , 0 , Lmin , Rmax);
        int size = (Rmax - Lmin) + 1;
```

```
        vector<int> result(size);
        View(root , abs(Lmin), result);
        return result;
    }
};
```