Vertical Order Traversal (when order is not important in note)

Stepwise Algorithm Explanation: 1. **Problem Understanding:**
Vertical Order Traversal means grouping the nodes of a binary tree based on their horizontal distance from the root. Here, we do not care about the top-to-bottom ordering of nodes in the same vertical line. 2. **Idea:**
- Each node has a "horizontal position" (pos). - Root node starts with pos = 0. - For left child, pos decreases by 1. - For right child, pos increases by 1. - We group nodes based on this pos value. 3. **Width Calculation:**
- We need to know the leftmost (Lmin) and rightmost (Rmax) positions in the tree. - Use a recursive function `width` to calculate Lmin and Rmax. 4. **Vertical Traversal:**
- Create a result vector of size (Rmax - Lmin + 1). - Use another recursive function `vertical` to traverse the tree. - Push each node's data into the correct vertical index. 5. **Offset Handling:**
- Since pos can be negative, we shift it using offset = abs(Lmin). - Example: if Lmin = -2, we add +2 to all positions so they map correctly to 0-based array indices. 6. **Return:**
- After traversal, return the result vector of vectors containing vertical groupings.

Code Implementation:

```
void vertical(Node * root , vector<vector<int>> &ans , int pos){
    if(!root) return ;
    ans[pos].push_back(root->data);
    vertical(root->left , ans , pos-1) ;
    vertical(root->right , ans , pos+1);
}

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

class Solution {
  public:
    vector<vector<int>> verticalOrder(Node *root) {
        int Lmin = 0 ,Rmax =0;
        width(root , Lmin , Rmax , 0);
        int size = (Rmax - Lmin) + 1;
        vector<vector<int>> result(size);
        vertical(root , result , abs(Lmin));
        return result;
    }
};
```