**Name:\_\_\_\_\_\_\_\_\_\_\_\_Shuqing Ye\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_UCI NET ID:\_\_\_\_\_\_\_shuqiny2\_\_\_\_\_\_\_\_**

|  |  |
| --- | --- |
| Test cases (including the edge cases):  Input: [10, 5, 15, 3, 7, 13, 18, 1, null, 6, null] Output: 7  Input: [10, 5, 15, 3, 7, 13, 18, 1, 2, 6, 4, 8, 9, 11, 14], output: 55  Input: [1, 2, null, 3, null, null, null, 4, null], output: 4  Input: [null], Output: 0  Input: [1], output: 1 | time complexity: O(n) (traverse the tree)  space complexity: O(1) + call stack |
| int maxSum = 0; // global variable, stores the sum of deepest leaves  int maxHeight = 0; // global variable, stores the height of tree that has been traversed  // Implementation of the desired method  public int sumDeepestLeaves(Node root) {  recursiveSum(root, 0);  return maxSum;  }  // helper method  // recursively traverse the tree and get the current sum of deepest leaves  private void recursiveSum(Node root, int height) {  if (root == null)  return;  if (height > maxHeight) {  maxHeight = height;  maxSum = root.key;  }  else if (height == maxHeight) {  maxSum += root.key;  }  recursiveSum(root.left, height + 1);  recursiveSum(root.right, height + 1);  } |  |