```cpp
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

using namespace std;

class BinaryTree {
public:
  int value;
  BinaryTree *left;
  BinaryTree *right;

  BinaryTree(int value) {
    this->value = value;
    left = NULL;
    right = NULL;
  }
};

int sumAllNodeDepths(BinaryTree *node,
                     unordered_map<BinaryTree *, int> &nodeDepths);
void addNodeDepths(BinaryTree *node,
                   unordered_map<BinaryTree *, int> &nodeDepths,
                   unordered_map<BinaryTree *, int> &nodeCounts);
void addNodeCounts(BinaryTree *node,
                   unordered_map<BinaryTree *, int> &nodeCounts);

// Average case: when the tree is balanced
// O(n) time | O(n) space - where n is the number of nodes in the Binary Tree
int allKindsOfNodeDepths(BinaryTree *root) {
  unordered_map<BinaryTree *, int> nodeCounts = {};
  addNodeCounts(root, nodeCounts);
  unordered_map<BinaryTree *, int> nodeDepths = {};
  addNodeDepths(root, nodeDepths, nodeCounts);
  return sumAllNodeDepths(root, nodeDepths);
}

int sumAllNodeDepths(BinaryTree *node,
                     unordered_map<BinaryTree *, int> &nodeDepths) {
  if (node == NULL)
    return 0;
  return sumAllNodeDepths(node->left, nodeDepths) +
         sumAllNodeDepths(node->right, nodeDepths) + nodeDepths[node];
}

void addNodeDepths(BinaryTree *node,
                   unordered_map<BinaryTree *, int> &nodeDepths,
                   unordered_map<BinaryTree *, int> &nodeCounts) {
  nodeDepths.insert({node, 0});
  if (node->left != NULL) {
    addNodeDepths(node->left, nodeDepths, nodeCounts);
    nodeDepths[node] += nodeDepths[node->left] + nodeCounts[node->left];
  }
  if (node->right != NULL) {
    addNodeDepths(node->right, nodeDepths, nodeCounts);
    nodeDepths[node] += nodeDepths[node->right] + nodeCounts[node->right];
  }
}

void addNodeCounts(BinaryTree *node,
                   unordered_map<BinaryTree *, int> &nodeCounts) {
  nodeCounts.insert({node, 1});
  if (node->left != NULL) {
    addNodeCounts(node->left, nodeCounts);
    nodeCounts[node] += nodeCounts[node->left];
  }
  if (node->right != NULL) {
    addNodeCounts(node->right, nodeCounts);
    nodeCounts[node] += nodeCounts[node->right];
  }
}
```