4. Graph Algorithm

Design and analyze an algorithm to find the diameter of a binary tree. The diameter is defined as the longest path between any two nodes in the tree.

To find the diameter of a binary tree, we have to discuss three cases:

1. The longest path which includes the root of the binary tree.
2. The diameter of the right subtree
3. The diameter of the left subtree

The diameter is the maximum value of the three.
In case one, we can see that the longest path which includes the root of the binary tree is equal to : the height of right subtree + the height of left subtree + 1 (root).

Therefore, I can apply recursion to solve this question.
We first construct a helper function to calculate the height of a given binary tree:

```
def height(tree){
##  base case
 If tree == null {
        return 0
}
##  the height is the maximum height between left and right subtree + 1

height = 1 + max(height(tree.left),height(tree.right))
return height
}
```

Here is my main function to calculate the diameter of the binary tree using height function:

```
def diameter(tree){
## base case
If tree == null{
          Return 0
}
## calculate the left subtree height and the right subtree height.
Left_height = height(tree.left)
Right_height = height(tree.right)
## calculate the diameter of the tree including the root
root_L_R  = Left_height + Right_height + 1
## apply recursion here to find the maximum of the three
return max(root_L_R, diameter(tree.left),diameter(tree.right))
}
```

Time complexity:
It is obvious that this algorithm is O(n^2). The reason is that for every root node in the recursion, we are going to check every other node again. As a result, the time complexity is n*n = O(n^2)

Possible Improvements:
To improve this algorithm, we can keep recording the left_height and right_height during each recursion and passing down into the diameter function so that we can avoid calling the height function. This will reduce the time complexity to O(n) but will require extra space to store the values of the height.