



# Solution Review: Problem Challenge 2

We'll cover the following



- Path with Maximum Sum (hard)
  - Solution
  - Code
    - Time complexity
    - Space complexity

## Path with Maximum Sum (hard)#

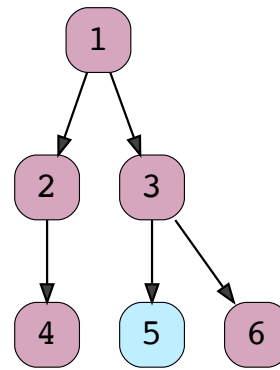
Find the path with the maximum sum in a given binary tree. Write a function that returns the maximum sum.

A path can be defined as a **sequence of nodes between any two nodes** and doesn't necessarily pass through the root. The path must contain at least one node.

**Example 1:**

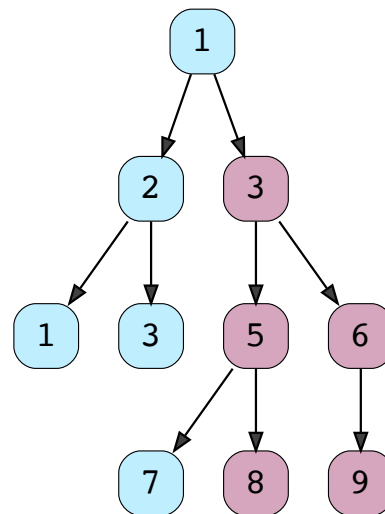
Output: 16

Explanation: The path with maximum sum is: [4, 2, 1, 3, 6]

**Example 2:**

Output: 31

Explanation: The path with maximum sum is: [8, 5, 3, 6, 9]



## Solution#

This problem follows the Binary Tree Path Sum


(<https://www.educative.io/collection/page/5668639101419520/5671464854355968/5642684278505472/>) pattern and shares the algorithmic logic with Tree Diameter


(<https://www.educative.io/collection/page/5668639101419520/5671464854355968/5691878833913856/>). We can follow the same **DFS** approach. The


only difference will be to ignore the paths with negative sums. Since we need to find the overall maximum sum, we should ignore any path which has an overall negative sum.


## Code#

Here is what our algorithm will look like, the most important changes are in the highlighted lines:

 Java

 Python3

 C++

 JS

```
class TreeNode {
  constructor(val, left = null, right = null) {
    this.val = val;
    this.left = left;
    this.right = right;
  }
}

class MaximumPathSum {
  find_maximum_path_sum(root) {
    this.globalMaximumSum = -Infinity;
    this.find_maximum_path_sum_recursive(root);
    return this.globalMaximumSum;
  }

  find_maximum_path_sum_recursive(currentNode) {
    if (currentNode === null) {
      return 0;
    }

    let maxPathSumFromLeft = this.find_maximum_path_sum_recursive(currentNode.left);
    let maxPathSumFromRight = this.find_maximum_path_sum_recursive(currentNode.right);

    // ignore paths with negative sums, since we need to find the maximum sum we
    // ignore any path which has an overall negative sum.
    maxPathSumFromLeft = Math.max(maxPathSumFromLeft, 0);
    maxPathSumFromRight = Math.max(maxPathSumFromRight, 0);
```

```
// maximum path sum at the current node will be equal to the sum from the left subtree + val of current node
// the sum from right subtree + val of current node
const localMaximumSum = maxPathSumFromLeft + maxPathSumFromRight + currentNode.val;

// update the global maximum sum
this.globalMaximumSum = Math.max(this.globalMaximumSum, localMaximumSum);

// maximum sum of any path from the current node will be equal to the maximum of the sum from the left subtree, the sum from the right subtree, or the sum from the left subtree + the sum from the right subtree + the value of the current node
// the sums from left or right subtrees plus the value of the current node
return Math.max(maxPathSumFromLeft, maxPathSumFromRight) + currentNode.val;
}
}
```

```
const maximumPathSum = new MaximumPathSum();
let root = new TreeNode(1);
root.left = new TreeNode(2);
root.right = new TreeNode(3);

console.log(`Maximum Path Sum: ${maximumPathSum.find_maximum_path_sum(root)}`);
root.left.left = new TreeNode(1);
root.left.right = new TreeNode(3);
root.right.left = new TreeNode(5);
root.right.right = new TreeNode(6);
root.right.left.left = new TreeNode(7);
root.right.left.right = new TreeNode(8);
root.right.right.left = new TreeNode(9);
console.log(`Maximum Path Sum: ${maximumPathSum.find_maximum_path_sum(root)}`);

root = new TreeNode(-1);
root.left = new TreeNode(-3);
console.log(`Maximum Path Sum: ${maximumPathSum.find_maximum_path_sum(root)}`);
```

RunSaveReset

## Time complexity#

The time complexity of the above algorithm is  $O(N)$ , where 'N' is the total number of nodes in the tree. This is due to the fact that we traverse each node once.

## Space complexity#

The space complexity of the above algorithm will be  $O(N)$  in the worst case. This space will be used to store the recursion stack. The worst case will happen when the given tree is a linked list (i.e., every node has only one child).

Interviewing soon? We've partnered with Hired so that companies apply to [utm\\_source=educative&utm\\_medium=lesson&utm\\_location=US&utm\\_campaign=educative](https://www.hired.com/?utm_source=educative&utm_medium=lesson&utm_location=US&utm_campaign=educative)

[← Back](#)[Problem Challenge 2](#)[Next →](#)[Introduction](#)[Mark as Completed](#)[Report an Issue](#)