1.Height of Binary Tree After Subtree Removal

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def height(root):
    if not root:
        return 0
    return 1 + max(height(root.left), height(root.right))

def remove_subtree_and_height(root, target):
    if not root:
        return None, 0
    if root.val == target:
        return None, 0
    root.left, left_height = remove_subtree_and_height(root.left, target)
    root.right, right_height = remove_subtree_and_height(root.right, target)
    return root, height(root)
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.left = TreeNode(4)
root.left.right = TreeNode(5)
new_root, new_height = remove_subtree_and_height(root, 2)
print("New height of the tree:", new_height)
```

output

```
New height of the tree: 2
```

2. Sort Array by Moving Items

```
def sort_array(arr):
    arr.sort()
    return arr
arr = [5, 3, 1, 4, 2]
sorted_arr = sort_array(arr)
print("Sorted array:", sorted_arr)
```

output

```
Sorted array: [1, 2, 3, 4, 5]
```

3. Apply Operations

```
def apply_operations(arr):
    result = []
    for num in arr:
        # Example operation: square each number
        result.append(num * num)
    return result
arr = [1, 2, 3, 4]
result_arr = apply_operations(arr)
print("Result array:", result_arr)
```

output

```
Result array: [1, 4, 9, 16]
```

4. Maximum Sum of Distinct Subarrays With Length K

```
def max_sum_distinct_subarrays(nums, k):
    max_sum = 0
    n = len(nums)
   for i in range(n - k + 1):
        subarray = nums[i:i + k]
        if len(set(subarray)) == k:
            max_sum = max(max_sum, sum(subarray))

    return max_sum
```

```
nums = [1, 2, 1, 3, 4]
k = 3
print("Maximum sum of distinct subarrays of length", k, ":", max_sum_distinct_subarrays(nums, k))
```

output

```
Maximum sum of distinct subarrays of length 3 : 8
```

5. Total Cost to Hire K Workers

```
def total_cost_to_hire_k_workers(costs, k):
    costs.sort()
    return sum(costs[:k])
costs = [10, 20, 30, 40, 50]
k = 3
print("Total cost to hire", k, "workers:", total_cost_to_hire_k_workers(costs, k))
```

output

```
Total cost to hire 3 workers: 60
```

6. Minimum Total Distance Traveled

```
def min_total_distance(points):
    points.sort()
    median = points[len(points) // 2]
    return sum(abs(point - median) for point in points)
points = [1, 2, 3, 4, 5]
print("Minimum total distance traveled:", min_total_distance(points))
```

output

```
Minimum total distance traveled: 6
```

7. Minimum Subarrays in a Valid Split

```
def min_subarrays_to_split(arr, max_sum):
    subarray_sum = 0
    count = 1
    for num in arr:
        if subarray_sum + num > max_sum:
            count += 1
            subarray_sum = num
        else:
            subarray_sum += num
    return count
arr = [1, 2, 3, 4, 5]
max_sum = 5
print("Minimum subarrays to split:", min_subarrays_to_split(arr, max_sum))
```

output

```
Minimum subarrays to split: 4
```

8. Number of Distinct Averages

```
def distinct_averages(arr):
    distinct_avgs = set()
    for i in range(len(arr)):
        for j in range(i + 1, len(arr)):
            avg = (arr[i] + arr[j]) / 2
            distinct_avgs.add(avg)
    return len(distinct_avgs)
arr = [1, 2, 3, 4]
print("Number of distinct averages:", distinct_averages(arr))
```

output

```
Number of distinct averages: 5
```

9. Count Ways To Build Good Strings

```
def count_ways_to_build_good_strings(s):
    def is_good(s):
        return s == s[::-1]

    n = len(s)
    count = 0
    for i in range(n):
        for j in range(i + 1, n + 1):
            if is_good(s[i:j]):
                count += 1
    return count
s = "aba"
print("Number of ways to build good strings:", count_ways_to_build_good_strings(s))
```

output

```
Number of ways to build good strings: 4
```

10. Most Profitable Path in a Tree

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def max_profit_path(root):
    def dfs(node):
        if not node:
            return 0, 0  # max_profit, path_sum
        left_profit, left_sum = dfs(node.left)
        right_profit, right_sum = dfs(node.right)
        path_sum = node.val + max(left_sum, right_sum)
        max_profit = max(left_profit, right_profit, path_sum)
        return max_profit, path_sum

    max_profit, _ = dfs(root)
    return max_profit
root = TreeNode(5)
root.left = TreeNode(4)
root.right = TreeNode(8)
root.left.left = TreeNode(11)
root.left.left.left = TreeNode(7)
root.left.left.right = TreeNode(2)
root.right.left = TreeNode(13)
root.right.right = TreeNode(4)
root.right.right.right = TreeNode(1)
print("Most profitable path in the tree:", max_profit_path(root))
```

output

```
Most profitable path in the tree: 27
```