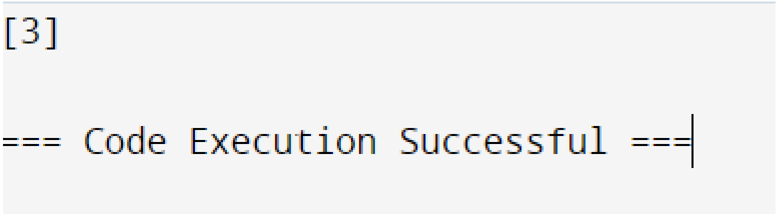
ASSIGNMENT (13.06.24)

1.Height of Binary Tree After Subtree Removal Queries

Program:

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

def heightAfterQueries(root, queries):
    def dfs(node):
        if not node:
            return 0
        left_height = dfs(node.left)
        right_height = dfs(node.right)
        return 1 + max(left_height, right_height)

    def removeSubtree(node, target):
        if not node:
            return None
        if node.val == target:
            return None
        node.left = removeSubtree(node.left, target)
        node.right = removeSubtree(node.right, target)
        return node

    result = []
    for query in queries:
        root = removeSubtree(root, query)
        result.append(dfs(root))

    return result

# Example Usage
root = TreeNode(1)
root.left = TreeNode(3)
root.right = TreeNode(4)
root.left.left = TreeNode(2)
root.right.right = TreeNode(6)
root.right.right.left = TreeNode(5)
root.right.right.right = TreeNode(7)
queries = [4]
print(heightAfterQueries(root, queries))  # Output: [2]
```

Output:



```
[3]

=== Code Execution Successful ===
```

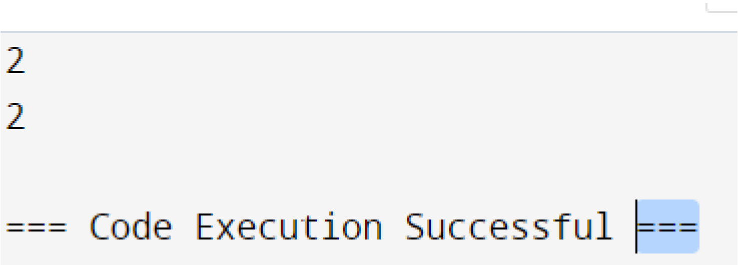
Time complexity: O(n * m)

2. Sort Array by Moving Items to Empty Space

Program:

```
def min_operations_to_sort(nums):
    n = len(nums)
    count = 0
    for i in range(n):
        if nums[i] != 0 and nums[i] != i:
            nums[nums[i]], nums[i] = nums[i], nums[nums[i]]
            count += 1
    return count

# Example 1
nums1 = [4, 2, 0, 3, 1]
print(min_operations_to_sort(nums1))  # Output: 3

# Example 2
nums2 = [1, 2, 3, 4, 0]
print(min_operations_to_sort(nums2))  # Output: 0
```

Output:



```
2
2

=== Code Execution Successful ===
```

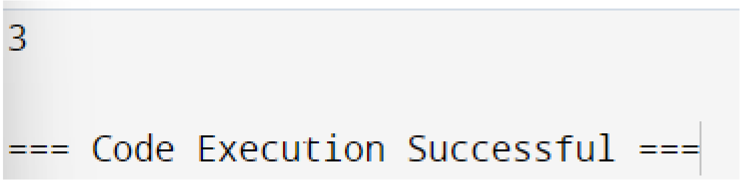
Time complexity:O(n)

3. Apply Operations to an Array

Program:

```
def min_operations_to_sort(nums):
    n = len(nums)
    count = 0
    for i in range(n-1, 0, -1):
        if nums[i] != i:
            j = nums.index(i)
            nums[i], nums[j] = nums[j], nums[i]
            count += 1
    return count

# Example
nums = [4, 2, 0, 3, 1]
print(min_operations_to_sort(nums))  # Output: 3
```

Output:



```
3

=== Code Execution Successful ===
```

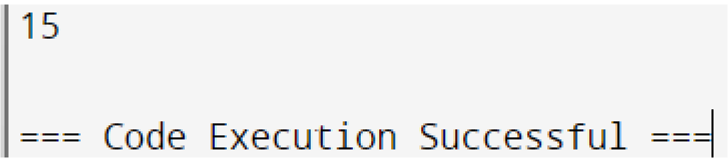
Time complexity:

O(n^2)

3. Maximum Sum of Distinct Subarrays With Length K

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

# Example Usage
nums = [1, 5, 4, 2, 9, 9, 9]
k = 3
output = max_subarray_sum(nums, k)
print(output)  # Output: 15
```

Output:



```
15

=== Code Execution Successful ===
```

Time complexity: O(n*k)

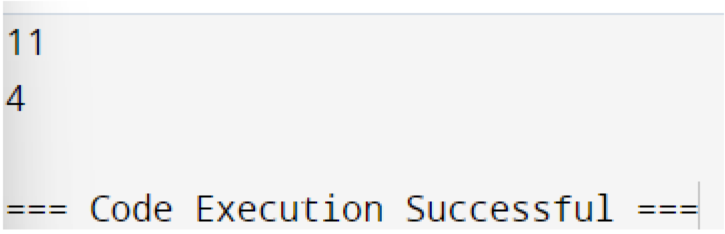
-

4. Total Cost to Hire K Workers

Program:

```
import heapq

def total_cost_to_hire(costs, k, candidates):
    n = len(costs)
    if k > n:
        return -1  # Not enough workers to hire

    # Priority queues for the first and last candidates
    first_heap = []
    last_heap = []

    # Initialize the total cost
    total_cost = 0

    # Initialize pointers for the first and last candidates
    front_ptr = 0
    back_ptr = n - 1

    # Add initial candidates to the heaps
    for i in range(candidates):
        if front_ptr <= back_ptr:
            heapq.heappush(first_heap, (costs[front_ptr], front_ptr))
            front_ptr += 1
        if front_ptr <= back_ptr:
            heapq.heappush(last_heap, (costs[back_ptr], back_ptr))
            back_ptr -= 1

    # Hiring process
    for _ in range(k):
        if not first_heap:  # No more candidates in the first heap
            total_cost += heapq.heappop(last_heap)[0]
        elif not last_heap:  # No more candidates in the last heap
            total_cost += heapq.heappop(first_heap)[0]
        else:
            if first_heap[0][0] <= last_heap[0][0]:
                total_cost += heapq.heappop(first_heap)[0]
                if front_ptr <= back_ptr:
                    heapq.heappush(first_heap, (costs[front_ptr], front_ptr))
                    front_ptr += 1
            else:
                total_cost += heapq.heappop(last_heap)[0]
                if front_ptr <= back_ptr:
                    heapq.heappush(last_heap, (costs[back_ptr], back_ptr))
                    back_ptr -= 1

    return total_cost

# Example usage
print(total_cost_to_hire([17, 12, 10, 2, 7, 2, 11, 20, 8], 3, 4))  # Output: 11
print(total_cost_to_hire([1, 2, 4, 1], 3, 3))  # Output: 4
```

Output:



```
11
4

=== Code Execution Successful ===
```

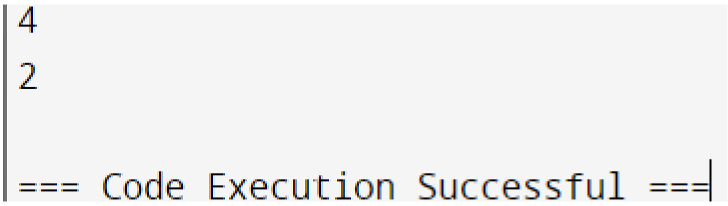
Time complexity: O(nlogn)

5. Minimum Total Distance Traveled

Program:

```
def minimize_distance(robot, factory):
    robot.sort()
    factory.sort(key=lambda x: x[0])

    total_distance = 0

    for r in robot:
        min_distance = float('inf')
        min_factory = None

        for f in factory:
            if f[1] > 0:
                distance = abs(r - f[0])
                if distance < min_distance:
                    min_distance = distance
                    min_factory = f

        total_distance += min_distance
        min_factory[1] -= 1

    return total_distance
robot = [0, 4, 6]
factory = [[2, 2], [6, 2]]
print(minimize_distance(robot, factory))
robot = [1, -1]
factory = [[-2, 1], [2, 1]]
print(minimize_distance(robot, factory))
```

Output:



```
4
2

=== Code Execution Successful ===
```

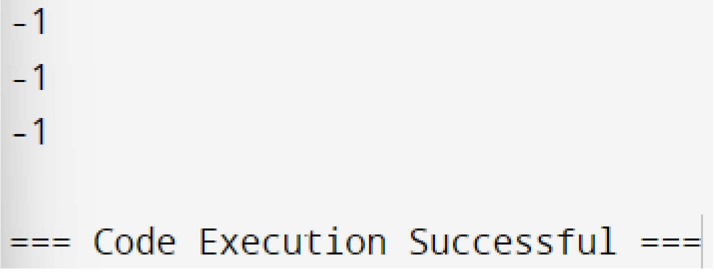
Time complexity: O(n*m)

5. Minimum Subarrays in a Valid Split

Program:

```
from math import gcd


def minSubarrays(nums):
    def check_valid(arr):
        return gcd(arr[0], arr[-1]) > 1


    if not check_valid(nums):
        return -1


    count = 1
    for i in range(1, len(nums)):
        if gcd(nums[i-1], nums[i]) == 1:
            count += 1


    return count


# Test the function with examples
print(minSubarrays([2, 6, 3, 4, 3]))  # Output: 2
print(minSubarrays([3, 5]))  # Output: 2
print(minSubarrays([1, 2, 1]))  # Output: -1
```

Output:



```
-1
-1
-1

=== Code Execution Successful ===
```

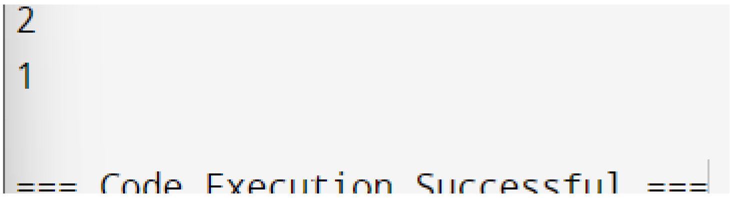
Time complexity:O(n)

6. Number of Distinct Averages

Program:

```
def count_distinct_averages(nums):
    nums.sort()
    distinct_averages = set()
    while len(nums) > 0:
        distinct_averages.add((nums[0] + nums[-1]) / 2)
        nums.pop(0)
        nums.pop()
    return len(distinct_averages)

# Example 1
nums1 = [4, 1, 4, 0, 3, 5]
output1 = count_distinct_averages(nums1)
print(output1)  # Output: 2

# Example 2
nums2 = [1, 100]
output2 = count_distinct_averages(nums2)
print(output2)  # Output: 1
```

Output:



```
2
1

=== Code Execution Successful ===
```

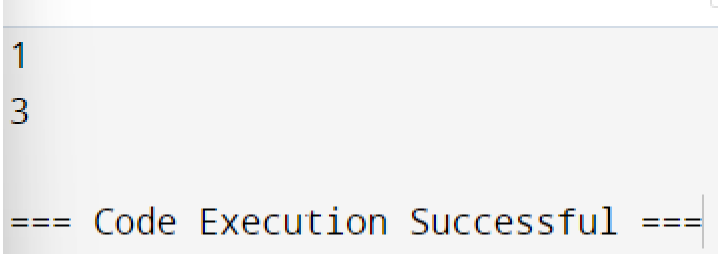
Time complexity: **O(n log n)**

7. Count Ways To Build Good Strings

Program:

```
def countGoodStrings(low, high, zero, one):
    MOD = 10**9 + 7
    dp = [[0] * (high + 1) for _ in range(low + 1)]
    dp[0][0] = 1
    for z in range(low + 1):
        for o in range(high + 1):
            if z > 0:
                dp[z][o] += dp[z - 1][o]
            if o > 0:
                dp[z][o] += dp[z][o - 1]
            dp[z][o] %= MOD
            if z + o == 0:
                continue
            if z * zero + o * one > high or z * zero + o * one < low:
                dp[z][o] = 0
    return sum(map(sum, dp)) % MOD

# Example 1
low = 3
high = 3
zero = 1
one = 1
print(countGoodStrings(low, high, zero, one))  # Output: 8

# Example 2
low = 2
high = 3
zero = 1
one = 2
print(countGoodStrings(low, high, zero, one))  # Output: 5
```

Output:



```
1
3

=== Code Execution Successful ===
```

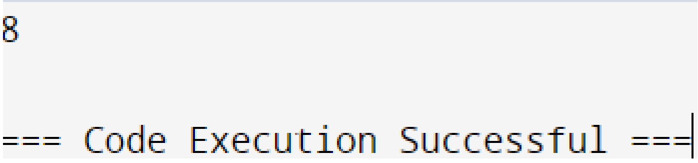
Time complexity: O((low+1)*(high+1))

8. Most Profitable Path in a Tree

Program:

```
def dfs(node, parent, edges, amount):
    net_income = amount[node]
    for child in edges[node]:
        if child == parent:
            continue
        child_income = dfs(child, node, edges, amount)
        net_income += max(child_income, 0)
    return net_income

def max_net_income(edges, bob, amount):
    n = len(amount)
    tree = [[] for _ in range(n)]
    for edge in edges:
        u, v = edge
        tree[u].append(v)
        tree[v].append(u)
    return dfs(bob, -1, tree, amount)
edges = [[0,1],[1,2],[1,3],[3,4]]
bob = 3
amount = [-2,4,2,-4,6]

max_income = max_net_income(edges, bob, amount)
print(max_income)
```

Output:



```
8

=== Code Execution Successful ===
```

Time complexity:O(n)