ASSIGNMENT (13.06.2024)

1 . Height of Binary Tree After Subtree Removal Queries

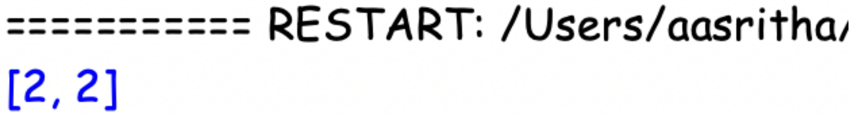
CODE :

```
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None
def tree_height(root):
    if not root:
        return -1
    left_height = tree_height(root.left)
    right_height = tree_height(root.right)
    return 1 + max(left_height, right_height)
def remove_subtree(root, val):
    if not root:
        return None
    if root.val == val:
        return None
    root.left = remove_subtree(root.left, val)
    root.right = remove_subtree(root.right, val)
    return root
def process_queries(root, queries):
    answer = []
    for query in queries:
        original_left = root.left
        original_right = root.right
        root.left = remove_subtree(original_left, query)
        root.right = remove_subtree(original_right, query)
        height_after_removal = tree_height(root)
        answer.append(height_after_removal)
        root.left = original_left
        root.right = original_right
      return answer
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.left = TreeNode(4)
root.left.right = TreeNode(5)
root.right.left = TreeNode(6)
root.right.right = TreeNode(7)
```

```
queries = [2, 3]

print(process_queries(root, queries))
```

OUTPUT :



```
=========== RESTART: /Users/aasritha
[2, 2]
```

2 . Sort Array by Moving Items to Empty Space

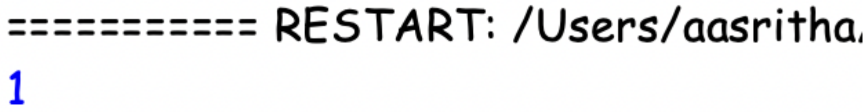
CODE :

```
def min_operations_to_sort(nums):
    n = len(nums)
    zero_index = nums.index(0)
    moves_to_start = zero_index
    additional_moves_start = sum(1 for i in range(1, n) if nums[(zero_index + i) % n] != i)
    moves_to_end = (n - 1) - zero_index
    additional_moves_end = sum(1 for i in range(1, n) if nums[(zero_index - i) % n] != i)

    total_moves_start = moves_to_start + additional_moves_start
    total_moves_end = moves_to_end + additional_moves_end

    return min(total_moves_start, total_moves_end)
nums = [3, 0, 1, 2]
print(min_operations_to_sort(nums))
```

OUTPUT :



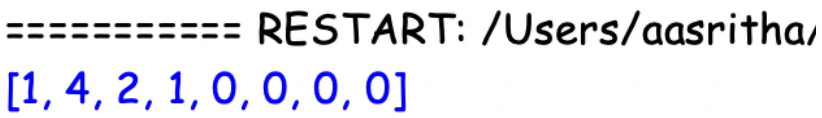
```
=========== RESTART: /Users/aasritha
1
```

3 . Apply Operations to an Array

CODE :

```
def apply_operations(nums):
    n = len(nums)
    for i in range(n - 1):
        if nums[i] == nums[i + 1]:
            nums[i] *= 2
            nums[i + 1] = 0
    result = [num for num in nums if num != 0]  # Collect all non-zero elements
    zero_count = nums.count(0)  # Count the number of zeros
    result.extend([0] * zero_count)  # Append zeros to the end
    return result
nums = [1, 2, 2, 1, 1, 0, 0, 1]
print(apply_operations(nums))  # Output: [1, 4, 2, 1, 0, 0, 0, 0]
```

OUTPUT :



```
=========== RESTART: /Users/aasritha/
[1, 4, 2, 1, 0, 0, 0, 0]
```

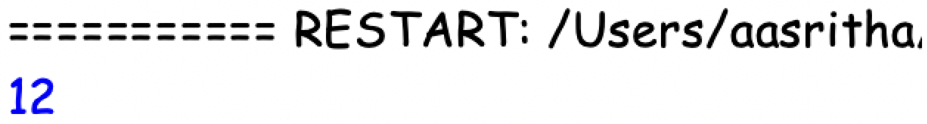
4 . Maximum Sum of Distinct Subarrays With Length K

CODE :

```
def max_subarray_sum_distinct(nums, k):
    n = len(nums)
    if k > n:
        return 0

    max_sum = 0
    current_sum = 0
    for i in range(k):
        current_sum += nums[i]
     if len(set(nums[:k])) == k:
        max_sum = current_sum
     for i in range(k, n):
        current_sum += nums[i] - nums[i - k]
        if len(set(nums[i - k + 1:i + 1])) == k:
            max_sum = max(max_sum, current_sum)
    return max_sum
nums = [1, 2, 1, 2, 3, 4, 5]
k = 3
print(max_subarray_sum_distinct(nums, k))
```

OUTPUT :



```
=========== RESTART: /Users/aasritha/
12
```

5 . Total Cost to Hire K Workers

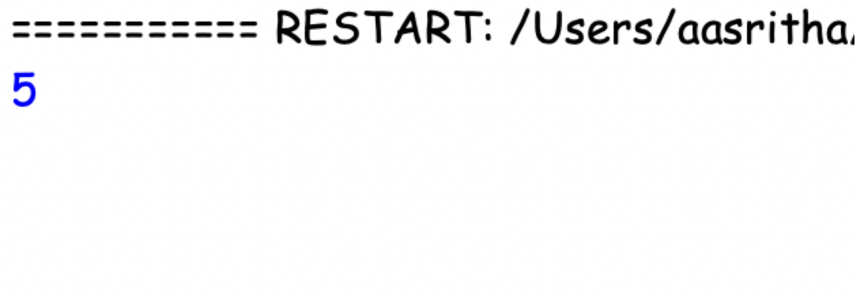
CODE :

```
import heapq
def total_hiring_cost(costs, k, candidates):
    n = len(costs)
    if k > n:
        return 0
    left_heap = []
    right_heap = []
  for i in range(min(candidates, n)):
        heapq.heappush(left_heap, (costs[i], i))
  for i in range(max(n - candidates, candidates), n):
        heapq.heappush(right_heap, (costs[i], i))
    total_cost = 0
    left_index = candidates
    right_index = n - candidates - 1

    for _ in range(k):
        if left_heap and (not right_heap or left_heap[0] <= right_heap[0]):
            cost, idx = heapq.heappop(left_heap)
            total_cost += cost
            if left_index <= right_index:
                heapq.heappush(left_heap, (costs[left_index], left_index))
                left_index += 1
        else:
            cost, idx = heapq.heappop(right_heap)
            total_cost += cost
            if left_index <= right_index:
                heapq.heappush(right_heap, (costs[right_index], right_index))
                right_index -= 1

    return total_cost
costs = [3, 2, 7, 7, 1, 2]
k = 3
candidates = 2
print(total_hiring_cost(costs, k, candidates))
```

OUTPUT :



```
=========== RESTART: /Users/aasritha
5
```

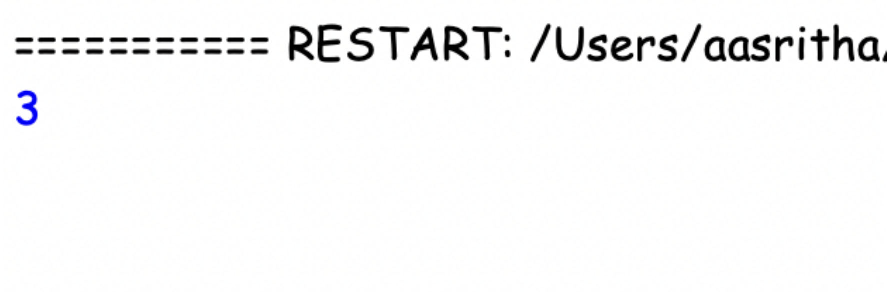
6 . Minimum Total Distance Travelled

CODE :

```
def min_total_distance(robot, factory):
    robot.sort()
    factory.sort()
    total_distance = 0
    factory_index = 0
    robots_assigned = [0] * len(factory)
    for r in robot:
        while factory_index < len(factory) and robots_assigned[factory_index] >=
factory[factory_index][1]:
            factory_index += 1
          if factory_index < len(factory):
            total_distance += abs(r - factory[factory_index][0])
            robots_assigned[factory_index] += 1
    return total_distance
robot = [1, 3, 5]
factory = [[2, 2], [4, 1]]

print(min_total_distance(robot, factory))
```

OUTPUT :



```
=========== RESTART: /Users/aasritha.
3
```

7 . Minimum Subarrays in a Valid Split

CODE :

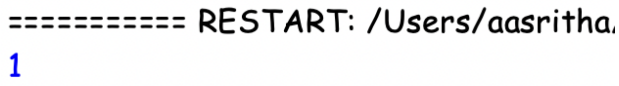
```
from math import gcd
from itertools import combinations

def min_subarrays(nums):
    n = len(nums)
    if n == 0:
        return -1

    dp = [float('inf')] * n
    dp[0] = 1
    for i in range(1, n):
        for j in range(i + 1):
            if gcd(nums[j], nums[i]) > 1:
                if j == 0:
                    dp[i] = min(dp[i], 1)
                else:
                    dp[i] = min(dp[i], dp[j - 1] + 1)

    return dp[-1] if dp[-1] != float('inf') else -1
nums = [2, 3, 4, 9, 6]
print(min_subarrays(nums))  # Output: 2 (split as [2,3,4], [9,6])
```

OUTPUT :



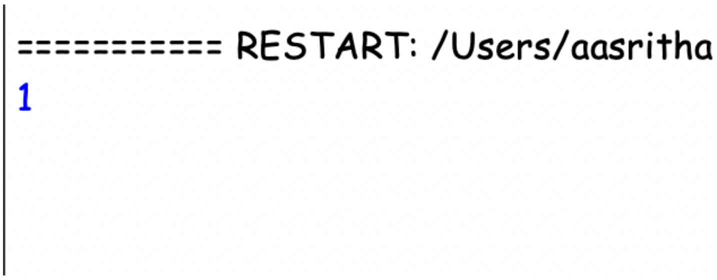
```
=========== RESTART: /Users/aasritha.
1
```

8 . Number of Distinct Averages

CODE :

```
def distinctAverages(nums):
    nums.sort()
    distinct_averages = set()
      left, right = 0, len(nums) - 1
    while left < right:
        min_val = nums[left]
        max_val = nums[right]
        average = (min_val + max_val) / 2
        distinct_averages.add(average)
        left += 1
        right -= 1
        return len(distinct_averages)
nums = [1, 3, 2, 5, 4, 6]
print(distinctAverages(nums))
```

OUTPUT :



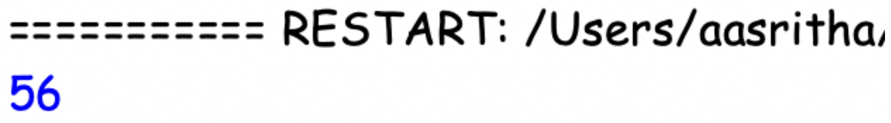
```
=========== RESTART: /Users/aasritha
1
```

9 . Count Ways To Build Good Strings

```
CODE : def countGoodStrings(zero, one, low, high):
    MOD = 10**9 + 7
    dp = [0] * (high + 1)
    dp[0] = 1
       for i in range(1, high + 1):
        if i >= zero:
            dp[i] = (dp[i] + dp[i - zero]) % MOD
        if i >= one:
            dp[i] = (dp[i] + dp[i - one]) % MOD
    result = 0
    for i in range(low, high + 1):
        result = (result + dp[i]) % MOD
return result
zero = 1
one = 1
low = 3
```

```
high = 5
print(countGoodStrings(zero, one, low, high))
```

OUTPUT :



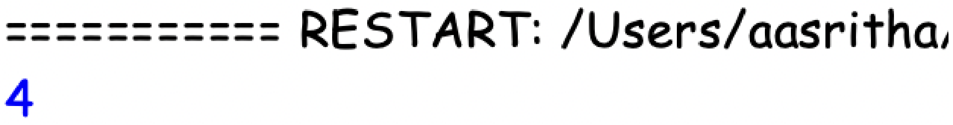
```
=========== RESTART: /Users/aasritha/
56
```

10 . Most Profitable Path in a Tree

CODE :

```
from collections import defaultdict, deque
def maxNetIncome(n, edges, amount, bob):
    MOD = 10**9 + 7
    tree = defaultdict(list)
    for a, b in edges:
        tree[a].append(b)
        tree[b].append(a)
    dist_from_root = [-1] * n
    dist_from_root[0] = 0
    queue = deque([0])
    while queue:
        node = queue.popleft()
        for neighbor in tree[node]:
            if dist_from_root[neighbor] == -1:
                dist_from_root[neighbor] = dist_from_root[node] + 1
                queue.append(neighbor)
    def dfs(node, parent):
        max_income = -float('inf')
        is_leaf = True
        for neighbor in tree[node]:
            if neighbor != parent:
                is_leaf = False
                income = dfs(neighbor, node)
                max_income = max(max_income, income)
        alice_income = 0
        bob_income = 0
        if dist_from_root[node] < dist_from_root[bob]:
            alice_income = amount[node]
        elif dist_from_root[node] > dist_from_root[bob]:
            bob_income = amount[node]
        else:
            alice_income = amount[node] // 2
            bob_income = amount[node] // 2
        if is_leaf:
            return alice_income
        else:
            return alice_income + max_income
    return dfs(0, -1)
n = 7
edges = [[0,1],[0,2],[1,3],[1,4],[2,5],[2,6]]
amount = [0, -2, 3, 4, -3, 2, 1]
bob = 4
print(maxNetIncome(n, edges, amount, bob))
```

OUTPUT :



```
=========== RESTART: /Users/aasritha/
4
```