1. Height of Binary Tree After Subtree Removal Queries

You are given the root of a binary tree with n nodes. Each node is assigned a unique valuefrom 1 to n. You are also given an array queries of size m.You have to performmindependent queries on the tree where in the ith query you do the following:

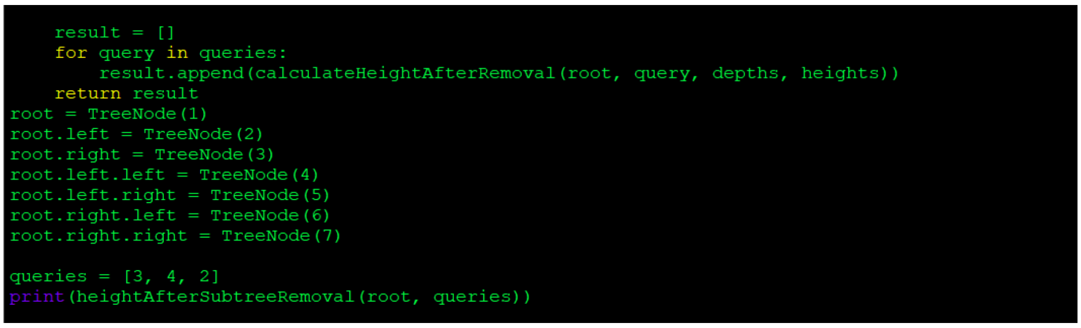
● Remove the subtree rooted at the node with the value queries[i] fromthe tree. It is

guaranteed that queries[i] will not be equal to the value of the root. Return an array answer of size m where answer[i] is the height of the tree after performingthe ith query. Note:

● The queries are independent, so the tree returns to its initial state after each query. ● The height of a tree is the number of edges in the longest simple path fromthe root tosome node in the tree.

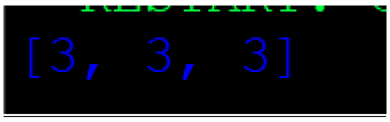
Code:

```
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None
def height(node):
    if not node:
        return 0
    left_height = height(node.left)
    right_height = height(node.right)
    return 1 + max(left_height, right_height)

def dfs(node, heights, depths, depth):
    if not node:
        return
    depths[node.val] = depth
    left_height = dfs(node.left, heights, depths, depth + 1)
    right_height = dfs(node.right, heights, depths, depth + 1)
    heights[node.val] = 1 + max(left_height, right_height)
    return heights[node.val]

def calculateHeightAfterRemoval(root, remove_val, depths, heights):
    if not root:
        return 0
    max_height = 0
    if root.val == remove_val:
        return 0

    def calculate(node, current_depth):
        nonlocal max_height
        if not node:
            return
        if node.val != remove_val:
            max_height = max(max_height, current_depth)
            calculate(node.left, current_depth + 1)
            calculate(node.right, current_depth + 1)

    calculate(root, 0)
    return max_height

def heightAfterSubtreeRemoval(root, queries):
    if not root:
        return []
    heights = {}
    depths = {}
    dfs(root, heights, depths, 0)
```



```python
    result = []
    for query in queries:
        result.append(calculateHeightAfterRemoval(root, query, depths, heights))
    return result
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 = [3, 4, 2]
print(heightAfterSubtreeRemoval(root, queries))
```

<u>Output:</u>



```
[3, 3, 3]
```

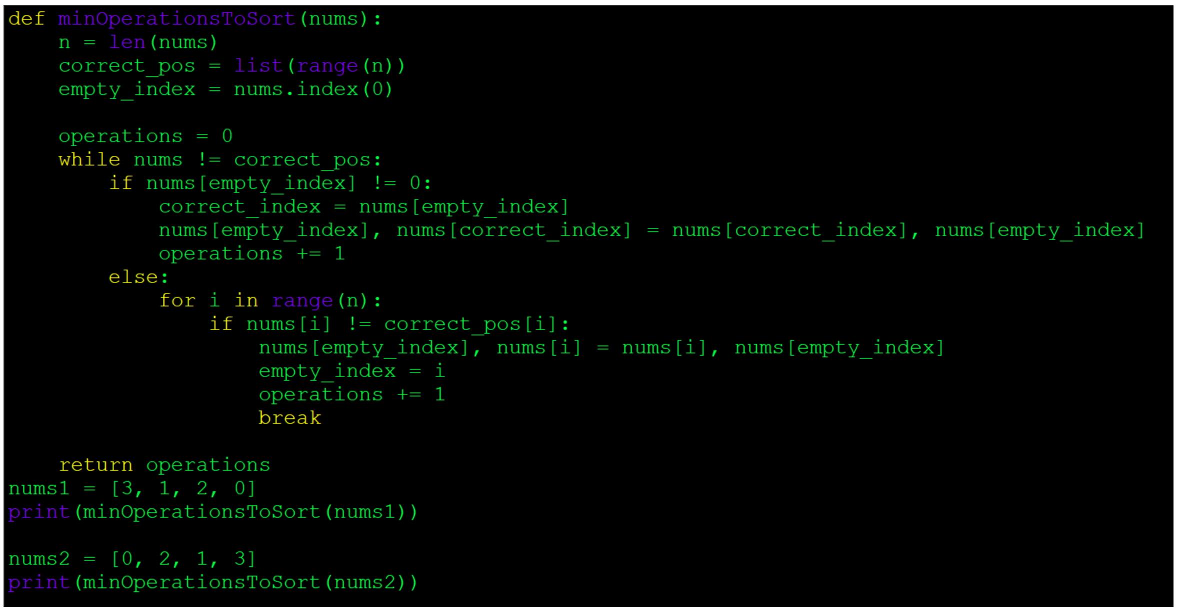
2. Sort Array by Moving Items to Empty Space

You are given an integer array nums of size n containing each element from0 to n - 1(inclusive). Each of the elements from 1 to n - 1 represents an item, and the element 0represents an empty space. In one operation, you can move any item to the empty space. nums is considered to be sortedif the numbers of all the items are in ascending order and the empty space is either at thebeginning or at the end of the array. For example, if n = 4, nums is sorted if:

- **nums = [0,1,2,3] or**
- **nums = [1,2,3,0]**

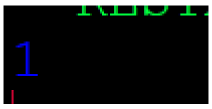
...and considered to be unsorted otherwise.Return the minimum number of operations neededto sort nums.

Code:



```
def minOperationsToSort(nums):
    n = len(nums)
    correct_pos = list(range(n))
    empty_index = nums.index(0)

    operations = 0
    while nums != correct_pos:
        if nums[empty_index] != 0:
            correct_index = nums[empty_index]
            nums[empty_index], nums[correct_index] = nums[correct_index], nums[empty_index]
            operations += 1
        else:
            for i in range(n):
                if nums[i] != correct_pos[i]:
                    nums[empty_index], nums[i] = nums[i], nums[empty_index]
                    empty_index = i
                    operations += 1
                    break

    return operations
nums1 = [3, 1, 2, 0]
print(minOperationsToSort(nums1))

nums2 = [0, 2, 1, 3]
print(minOperationsToSort(nums2))
```

Output:



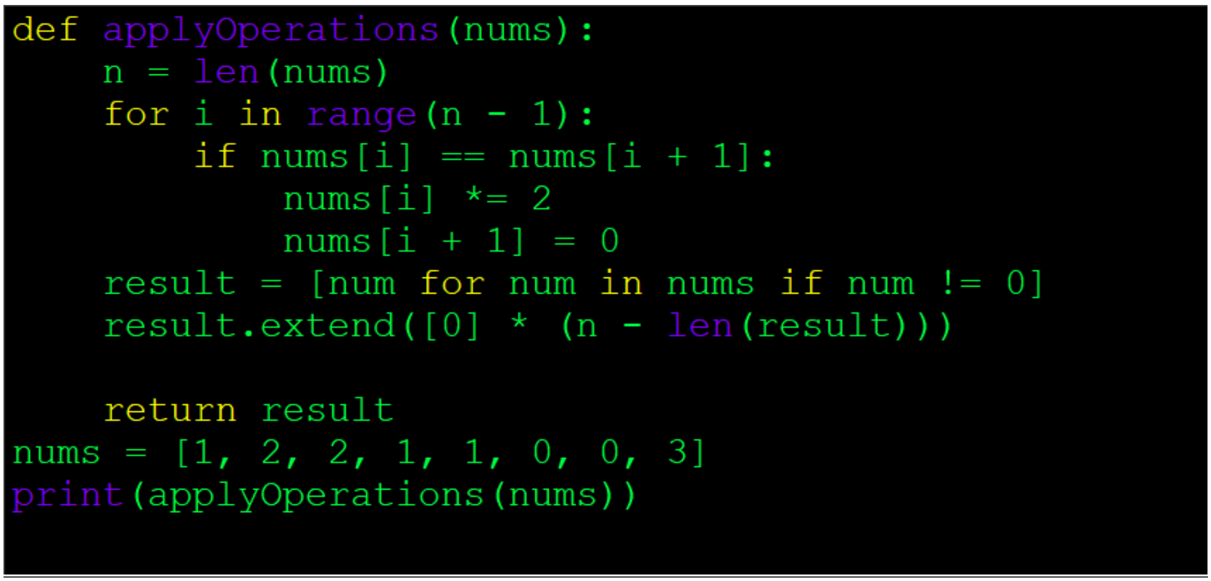
```
1
```

3. Apply Operations to an Array

You are given a 0-indexed array nums of size n consisting of non-negative integers.Youneedto apply n - 1 operations to this array where, in the ith operation (0-indexed), you will applythe following on the ith element of nums:

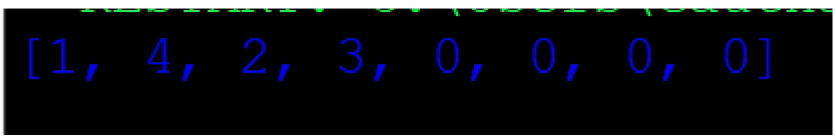
● If nums[i] == nums[i + 1], then multiply nums[i] by 2 and set nums[i + 1] to 0. Otherwise, you skip this operation. After performing all the operations, shift all the 0's to the end of the array. ● For example, the array [1,0,2,0,0,1] after shifting all its 0's to the end, is [1,2,1,0,0,0]. Return the resulting array.Note that the operations are applied sequentially, not all at once.

Code:



```
def applyOperations(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]
    result.extend([0] * (n - len(result)))

    return result
nums = [1, 2, 2, 1, 1, 0, 0, 3]
print(applyOperations(nums))
```

Output:



```
[1, 4, 2, 3, 0, 0, 0, 0]
```

4. **Maximum Sum of Distinct Subarrays With Length K**

You are given an integer array nums and an integer k. Find the maximumsubarray sumof all

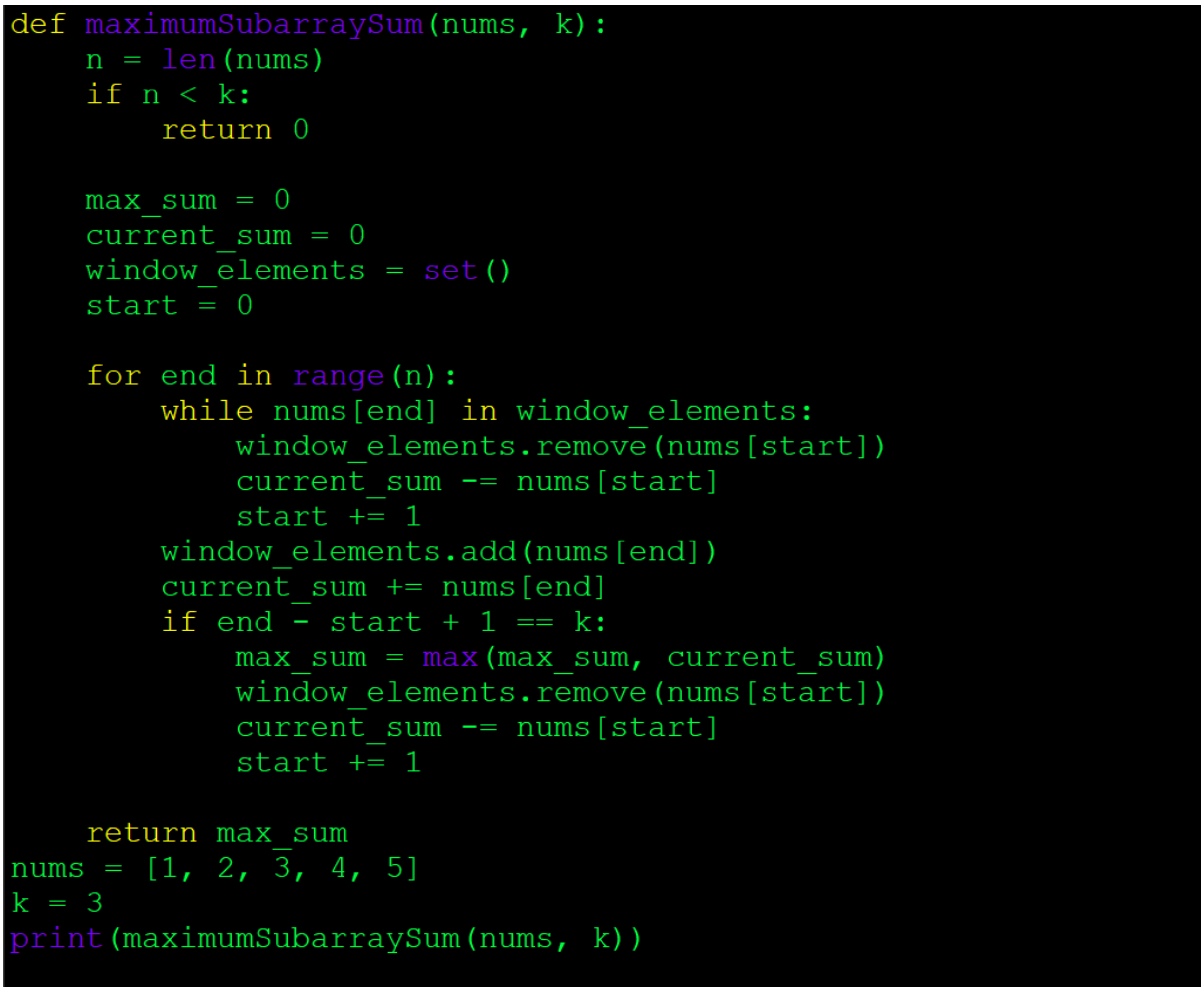
the subarrays of nums that meet the following conditions:

● The length of the subarray is k, and

● All the elements of the subarray are distinct. Return the maximum subarray sum of all the subarrays that meet the conditions. If nosubarray meets the conditions, return 0. A subarray is a contiguous non-empty sequenceof

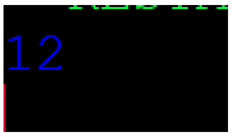
elements within an array

Code:



```
def maximumSubarraySum(nums, k):
    n = len(nums)
    if n < k:
        return 0

    max_sum = 0
    current_sum = 0
    window_elements = set()
    start = 0

    for end in range(n):
        while nums[end] in window_elements:
            window_elements.remove(nums[start])
            current_sum -= nums[start]
            start += 1
        window_elements.add(nums[end])
        current_sum += nums[end]
        if end - start + 1 == k:
            max_sum = max(max_sum, current_sum)
            window_elements.remove(nums[start])
            current_sum -= nums[start]
            start += 1

    return max_sum
nums = [1, 2, 3, 4, 5]
k = 3
print(maximumSubarraySum(nums, k))
```

Output:



```
12
```

5. . Total Cost to Hire K Workers

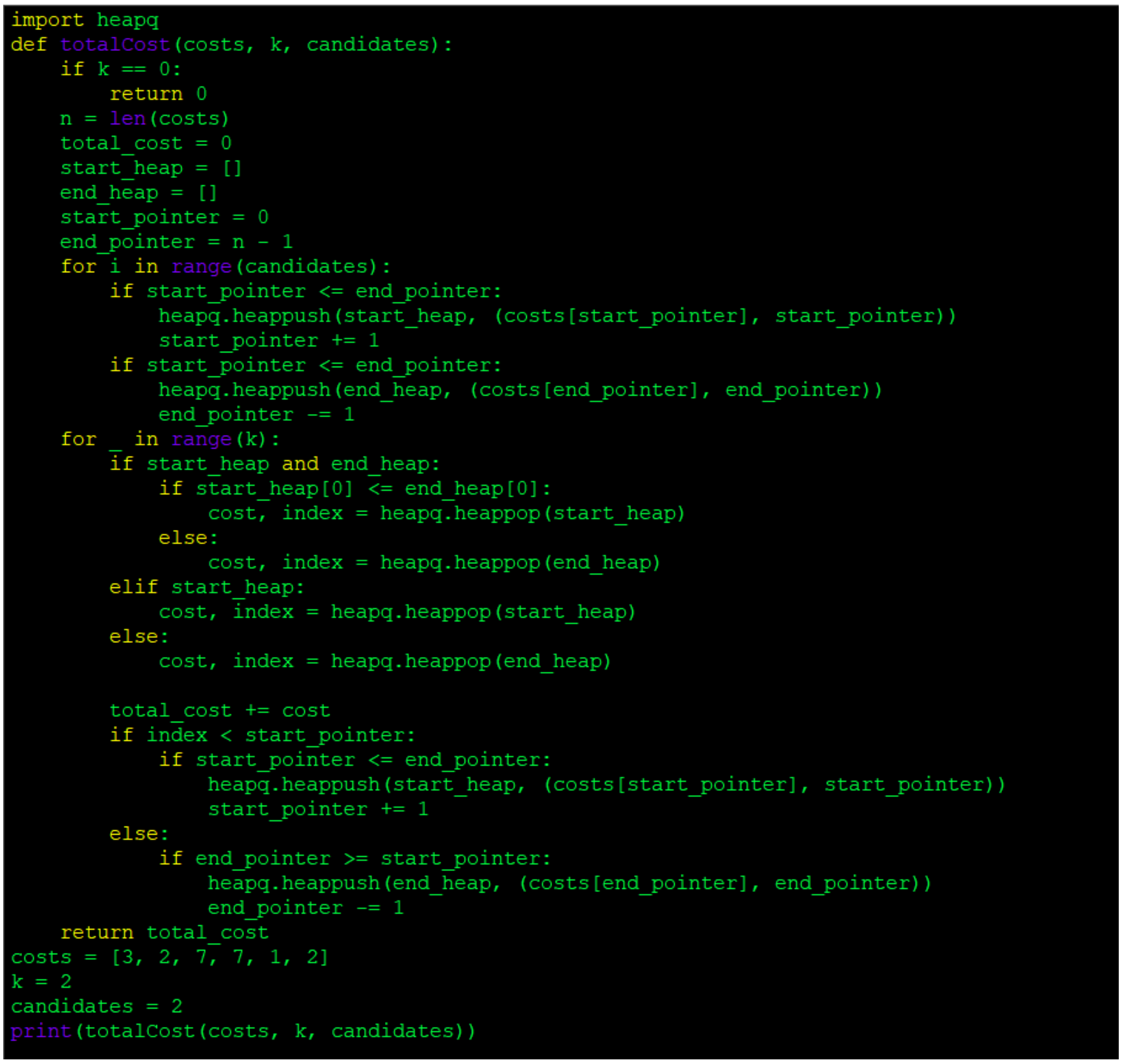
You are given a 0-indexed integer array costs where costs[i] is the cost of hiring the ithworker.You are also given two integers k and candidates. We want to hire exactly k workersaccording to the following rules:

● You will run k sessions and hire exactly one worker in each session. ● In each hiring session, choose the worker with the lowest cost fromeither the first

candidates workers or the last candidates workers. Break the tie by the smallest index. ○ For example, if costs = [3,2,7,7,1,2] and candidates = 2, then in the first hiringsession, we will choose the 4th worker because they have the lowest cost

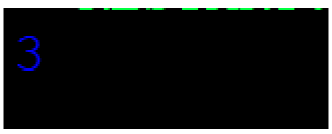
[3,2,7,7,1,2]. ○ In the second hiring session, we will choose 1st worker because they havethesame lowest cost as 4th worker but they have the smallest index [3,2,7,7,2]. Please note that the indexing may be changed in the process. ● If there are fewer than candidates workers remaining, choose the worker with thelowest cost among them. Break the tie by the smallest index. ● A worker can only be chosen once. Return the total cost to hire exactly k workers

Code:



```python
import heapq
def totalCost(costs, k, candidates):
    if k == 0:
        return 0
    n = len(costs)
    total_cost = 0
    start_heap = []
    end_heap = []
    start_pointer = 0
    end_pointer = n - 1
    for i in range(candidates):
        if start_pointer <= end_pointer:
            heapq.heappush(start_heap, (costs[start_pointer], start_pointer))
            start_pointer += 1
        if start_pointer <= end_pointer:
            heapq.heappush(end_heap, (costs[end_pointer], end_pointer))
            end_pointer -= 1
    for _ in range(k):
        if start_heap and end_heap:
            if start_heap[0] <= end_heap[0]:
                cost, index = heapq.heappop(start_heap)
            else:
                cost, index = heapq.heappop(end_heap)
        elif start_heap:
            cost, index = heapq.heappop(start_heap)
        else:
            cost, index = heapq.heappop(end_heap)

        total_cost += cost
        if index < start_pointer:
            if start_pointer <= end_pointer:
                heapq.heappush(start_heap, (costs[start_pointer], start_pointer))
                start_pointer += 1
        else:
            if end_pointer >= start_pointer:
                heapq.heappush(end_heap, (costs[end_pointer], end_pointer))
                end_pointer -= 1
    return total_cost
costs = [3, 2, 7, 7, 1, 2]
k = 2
candidates = 2
print(totalCost(costs, k, candidates))
```

Output:



```
3
```

6. Minimum Total Distance Traveled

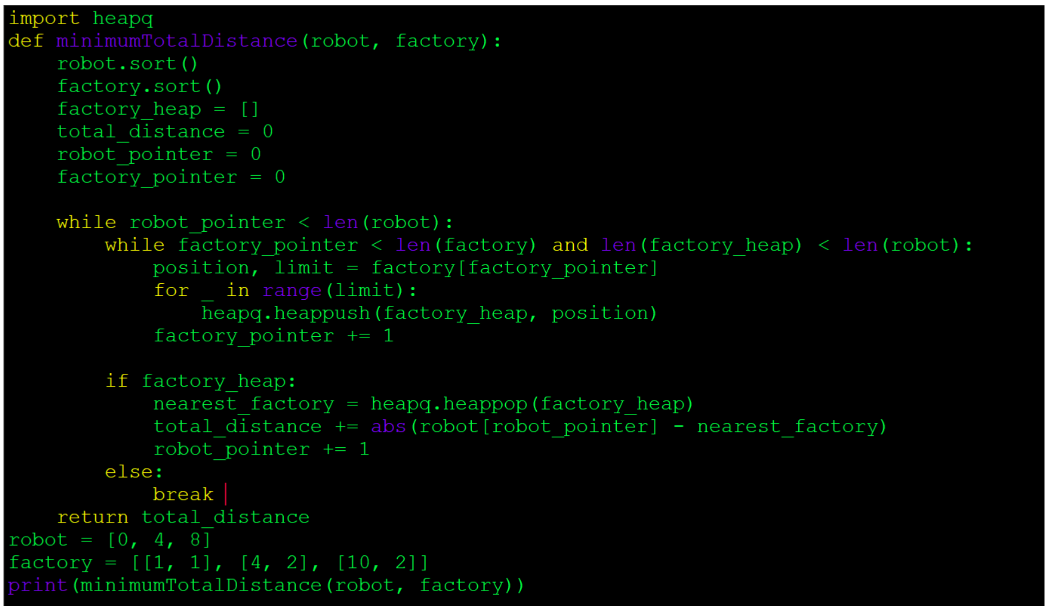
There are some robots and factories on the X-axis. You are given an integer array robot

where robot[i] is the position of the ith robot. You are also given a 2D integer array factorywhere factory[j] = [positionj, limitj] indicates that positionj is the position of the jth factoryand that the jth factory can repair at most limitj robots. The positions of each robot are unique. The positions of each factory are also unique. Notethat a robot can be in the same position as a factory initially. All the robots are initially broken; they keep moving in one direction. The direction couldbethe negative or the positive direction of the X-axis. When a robot reaches a factory that didnot reach its limit, the factory repairs the robot, and it stops moving. At any moment, you can set the initial direction of moving for some robot. Your target is tominimize the total distance traveled by all the robots. Return the minimum total distance traveled by all the robots. The test cases are generatedsuch that all the robots can be repaired. Note that

● All robots move at the same speed. ● If two robots move in the same direction, they will never collide. ● If two robots move in opposite directions and they meet at some point, they donot

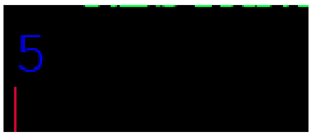
collide. They cross each other. ● If a robot passes by a factory that reached its limits, it crosses it as if it does not exist. ● If the robot moved from a position x to a position y, the distance it moved is |y- x|.

<u>Code:</u>



```python
import heapq
def minimumTotalDistance(robot, factory):
    robot.sort()
    factory.sort()
    factory_heap = []
    total_distance = 0
    robot_pointer = 0
    factory_pointer = 0

    while robot_pointer < len(robot):
        while factory_pointer < len(factory) and len(factory_heap) < len(robot):
            position, limit = factory[factory_pointer]
            for _ in range(limit):
                heapq.heappush(factory_heap, position)
            factory_pointer += 1

        if factory_heap:
            nearest_factory = heapq.heappop(factory_heap)
            total_distance += abs(robot[robot_pointer] - nearest_factory)
            robot_pointer += 1
        else:
            break
    return total_distance
robot = [0, 4, 8]
factory = [[1, 1], [4, 2], [10, 2]]
print(minimumTotalDistance(robot, factory))
```

Output:


```
5
```

7. Minimum Subarrays in a Valid Split

You are given an integer array nums.Splitting of an integer array nums into subarrays is validif:

● the greatest common divisor of the first and last elements of each subarray is greater

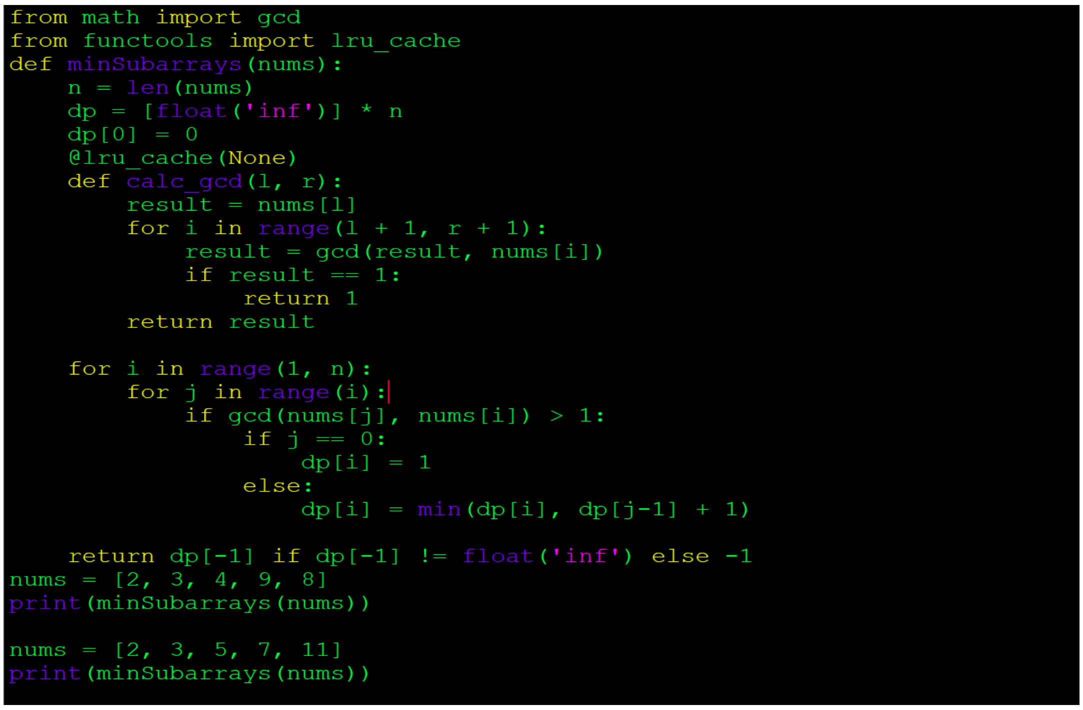
than 1, and

● each element of nums belongs to exactly one subarray. Return the minimum number of subarrays in a valid subarray splitting of nums. If a validsubarray splitting is not possible, return -1. Note that:

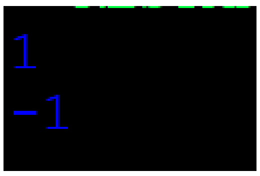
● The greatest common divisor of two numbers is the largest positive integer that

evenly divides both numbers. ● A subarray is a contiguous non-empty part of an array.

Code:


```python
from math import gcd
from functools import lru_cache
def minSubarrays(nums):
    n = len(nums)
    dp = [float('inf')] * n
    dp[0] = 0
    @lru_cache(None)
    def calc_gcd(l, r):
        result = nums[l]
        for i in range(l + 1, r + 1):
            result = gcd(result, nums[i])
            if result == 1:
                return 1
        return result

    for i in range(1, n):
        for j in range(i):
            if gcd(nums[j], nums[i]) > 1:
                if j == 0:
                    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, 8]
print(minSubarrays(nums))

nums = [2, 3, 5, 7, 11]
print(minSubarrays(nums))
```

Output:



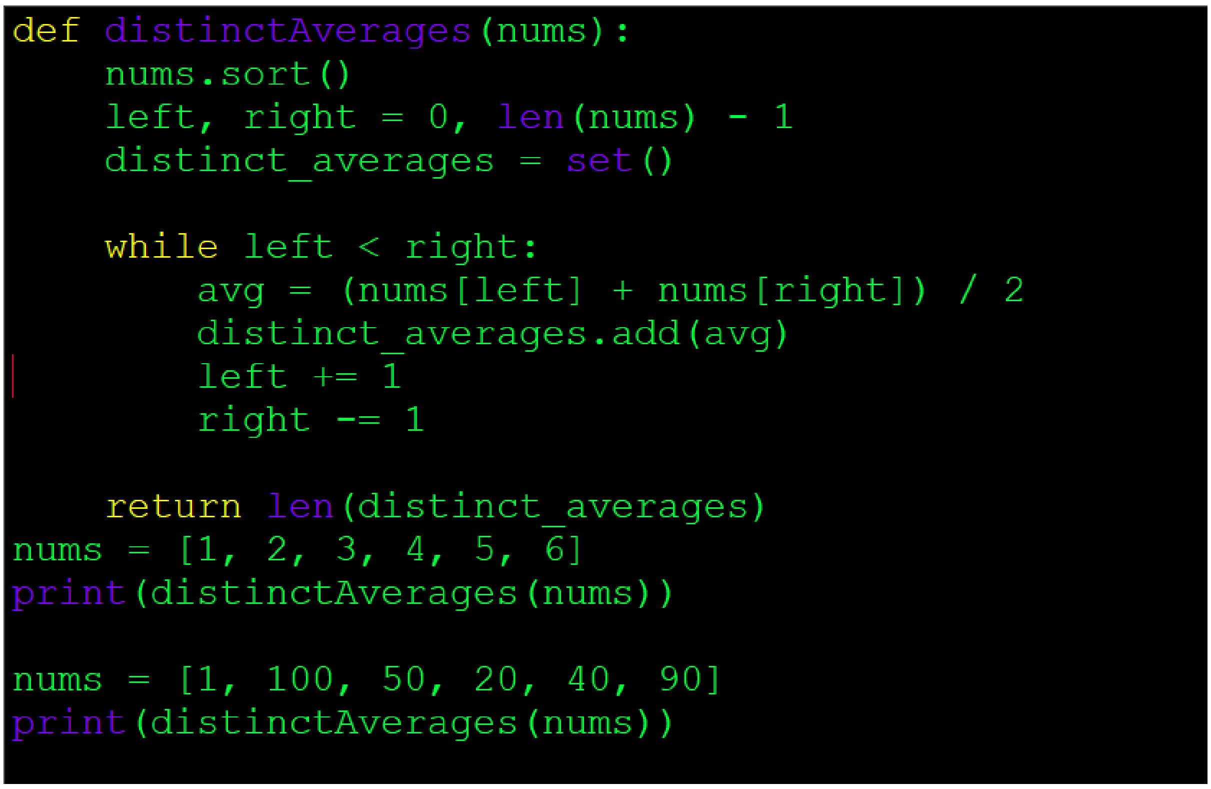
```
1
-1
```

8 . Number of Distinct Averages

You are given a 0-indexed integer array nums of even length. As long as nums is not empty, you must repetitively:

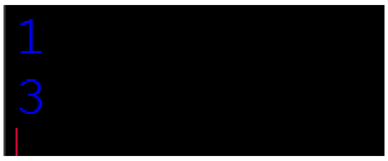
● Find the minimum number in nums and remove it. ● Find the maximum number in nums and remove it. ● Calculate the average of the two removed numbers. The average of two numbers a and b is (a + b) / 2. ● For example, the average of 2 and 3 is (2 + 3) / 2 = 2.5. Return the number of distinct averages calculated using the above process.Note that whenthere is a tie for a minimum or maximum number, any can be removed.

Code:



```python
def distinctAverages(nums):
    nums.sort()
    left, right = 0, len(nums) - 1
    distinct_averages = set()

    while left < right:
        avg = (nums[left] + nums[right]) / 2
        distinct_averages.add(avg)
        left += 1
        right -= 1

    return len(distinct_averages)
nums = [1, 2, 3, 4, 5, 6]
print(distinctAverages(nums))

nums = [1, 100, 50, 20, 40, 90]
print(distinctAverages(nums))
```

Output:


```
1
3
```

9. Count Ways To Build Good Strings

Given the integers zero, one, low, and high, we can construct a string by starting withanempty string, and then at each step perform either of the following:

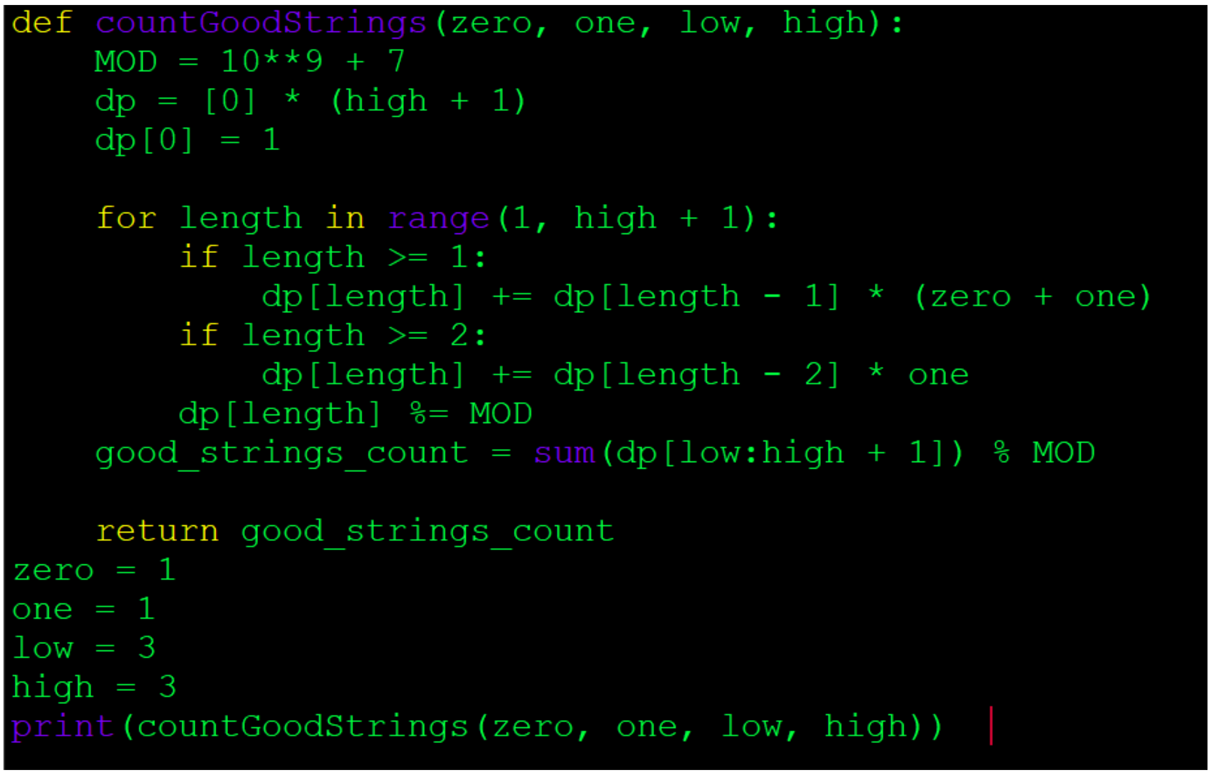
● Append the character '0' zero times. ● Append the character '1' one times.

This can be performed any number of times.A good string is a string constructed by the

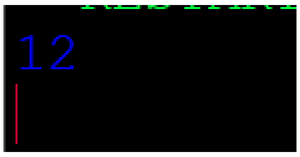
above process having a length between low and high (inclusive). Return the number of dif erent good strings that can be constructed satisfying these

properties. Since the answer can be large, return it modulo 109 + 7.

Code:


```python
def countGoodStrings(zero, one, low, high):
    MOD = 10**9 + 7
    dp = [0] * (high + 1)
    dp[0] = 1

    for length in range(1, high + 1):
        if length >= 1:
            dp[length] += dp[length - 1] * (zero + one)
        if length >= 2:
            dp[length] += dp[length - 2] * one
        dp[length] %= MOD
    good_strings_count = sum(dp[low:high + 1]) % MOD

    return good_strings_count
zero = 1
one = 1
low = 3
high = 3
print(countGoodStrings(zero, one, low, high))
```

Output:



```
12
```

10. . Most Profitable Path in a Tree

There is an undirected tree with n nodes labeled from 0 to n - 1, rooted at node 0. Youaregiven a 2D integer array edges of length n - 1 where edges[i] = [ai, bi] indicates that thereisan edge between nodes ai and bi in the tree. At every node i, there is a gate. You are also given an array of even integers amount, whereamount[i] represents:

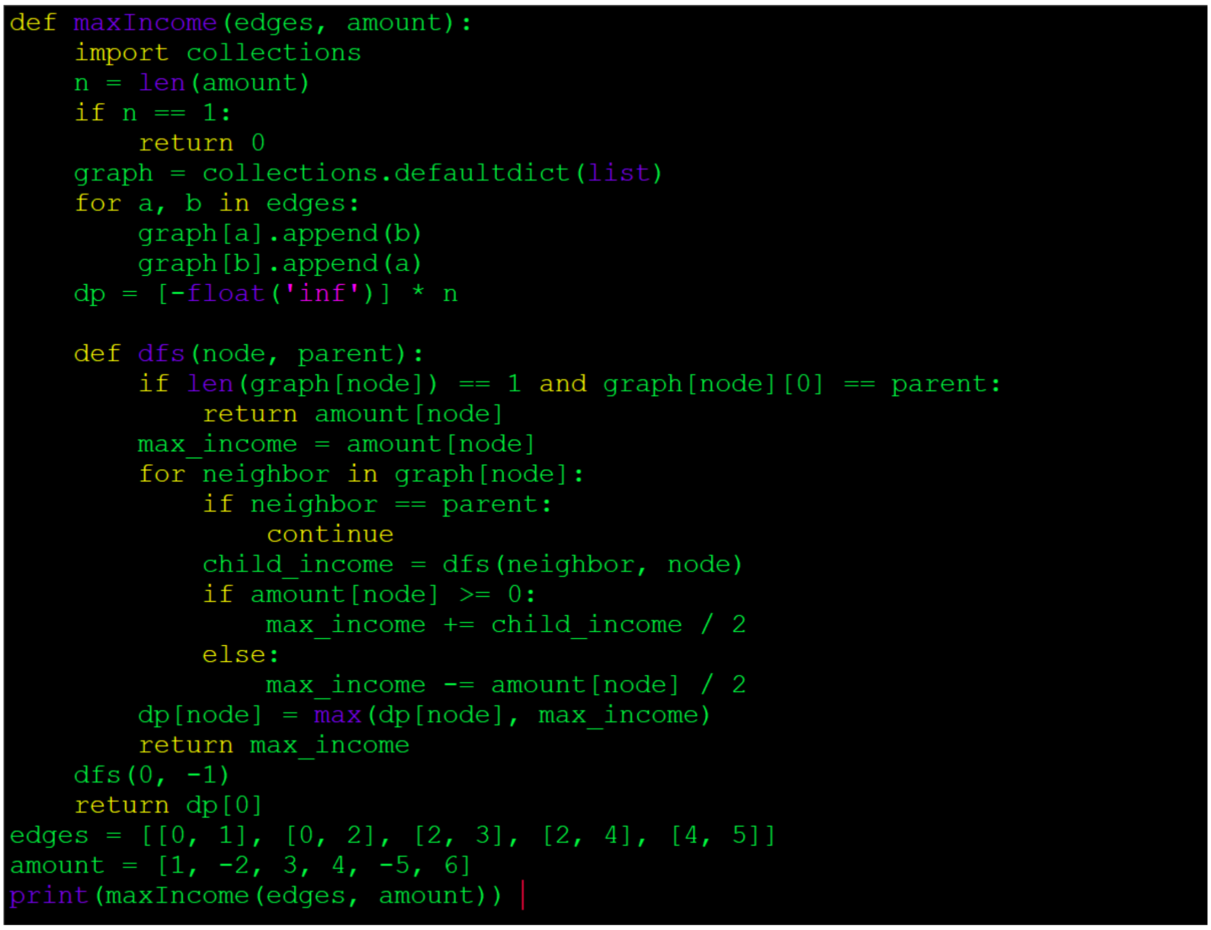
● the price needed to open the gate at node i, if amount[i] is negative, or, ● the cash reward obtained on opening the gate at node i, otherwise. The game goes on as follows:

● Initially, Alice is at node 0 and Bob is at node bob. ● At every second, Alice and Bob each move to an adjacent node. Alice moves towardssome leaf node, while Bob moves towards node 0. ● For every node along their path, Alice and Bob either spend money to open the gateat

that node, or accept the reward. Note that:

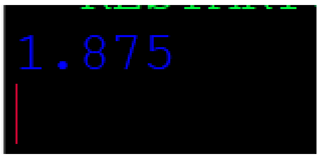
○ If the gate is already open, no price will be required, nor will there be anycashreward. ○ If Alice and Bob reach the node simultaneously, they share the price/rewardfor opening the gate there. In other words, if the price to open the gate is c, then both Alice and Bob pay c / 2 each. Similarly, if the reward at the gateisc, both of them receive c / 2 each. ● If Alice reaches a leaf node, she stops moving. Similarly, if Bob reaches node 0, hestops moving. Note that these events are independent of each other. Return the maximum net income Alice can have if she travels towards the optimal leaf node.

Code:



```
def maxIncome(edges, amount):
    import collections
    n = len(amount)
    if n == 1:
        return 0
    graph = collections.defaultdict(list)
    for a, b in edges:
        graph[a].append(b)
        graph[b].append(a)
    dp = [-float('inf')] * n

    def dfs(node, parent):
        if len(graph[node]) == 1 and graph[node][0] == parent:
            return amount[node]
        max_income = amount[node]
        for neighbor in graph[node]:
            if neighbor == parent:
                continue
            child_income = dfs(neighbor, node)
            if amount[node] >= 0:
                max_income += child_income / 2
            else:
                max_income -= amount[node] / 2
        dp[node] = max(dp[node], max_income)
        return max_income
    dfs(0, -1)
    return dp[0]
edges = [[0, 1], [0, 2], [2, 3], [2, 4], [4, 5]]
amount = [1, -2, 3, 4, -5, 6]
print(maxIncome(edges, amount))
```

Output:



```
1.875
```