In [ ]:
```
6.Minimum Total Distance Traveled
```

In [1]:
```
def min_total_distance(positions):
    positions.sort()
    median = positions[len(positions)// 2]
    return sum(abs(pos - median) for pos in positions)
positions = [1, 2, 9, 5, 7]
result = min_total_distance(positions)
print(result)
```

```
13
```

In [ ]:
```
7.Minimum Subarrays in a Valid Split
```

In [2]:
```
def min_total_distance(robot, factory):
    robot.sort()
    factory.sort()
    total_distance = 0
    factory_index = 0
    factory_capacity = factory[factory_index][1]
    for r in robot:
        while factory_capacity== 0:
            factory_index+= 1
            factory_capacity = factory[factory_index][1]
        total_distance += abs(r - factory[factory_index][0])
        factory_capacity-= 1
    return total_distance
robot_positions =[1, 3, 6, 10]
factories = [[2, 2], [5, 2]]
result = min_total_distance(robot_positions, factories)
print(result)
```

```
8
```

In [ ]:
```
9.
```

In [4]:
```
def min_subarrays(nums):
    n = len(nums)
    if n == 0:
        return 0
    count = 1
    current_product = nums[0]
    for i in range(1, n):
        current_product *= nums[i]
        if current_product == 0:
            count+= 1
            if i< n - 1:
                current_product = nums[i + 1]
    return count
nums = [1,2,1]
print(min_subarrays(nums))
```

```
1
```

In [ ]:
```
8.. Number of Distinct Average
```

In [5]:
```
def distinct_averages(nums):
    n = len(nums)
    distinct_avg = set()
    for i in range(n):
        for j in range(i + 1, n):
            avg = (nums[i] + nums[j]) / 2
            distinct_avg.add(avg)
    return len(distinct_avg
nums = [1, 2, 3, 4]
print(distinct_averages(nums))
```

```
5
```

In [ ]:
```
9.. Count Ways To Build Good Strings
```

In [6]:
```
def count_good_strings(s, bad_chars):
    bad_set = set(bad_chars)
    count_good = 0
    for char in s:
        if char not in bad_set:
            count_good+= 1
    return count_good
s = "abacabadabacaba"
bad_chars = "abc"
print(count_good_strings(s, bad_chars))
```

```
1
```

In [ ]:
```
10.. Most Profitable Path in a Tree
```

In [7]:
```
class TreeNode:
    def __init__(self, value):
        self.value = value
        self.children = []
def max_profit_path(root):
    if not root:
        return 0
    def dfs(node):
        nonlocal max_profit
        if not node:
            return 0
        current_profit = node.value
        for child in node.children:
            current_profit += max(0, dfs(child))
        max_profit = max(max_profit, current_profit)
        return current_profit
    max_profit = float('-inf')
    dfs(root)
    return max_profit
root = TreeNode(1)
root.children = [TreeNode(2), TreeNode(3)]
root.children[0].children = [TreeNode(4), TreeNode(5)]
print(max_profit_path(root))  # Output: 10 (path: 1 -> 2 -> 5)
```

```
15
```

```
2.Sort Array by Moving Items to Empty Space
```

In [9]:
```
def min_operations_to_sort(nums):
    n = len(nums)
    empty_space = nums.index(0)
    operations = 0
    for i in range(n):
        if nums[i]!= i and nums[i] != 0:
            nums[empty_space], nums[i] = nums[i], nums[empty_space]
            empty_space = i
            operations += 1
    return operations
print(min_operations_to_sort([4, 2, 0, 3, 1]))
print(min_operations_to_sort([1, 2, 3, 4, 0]))
print(min_operations_to_sort([1, 0, 2, 4, 3]))
```

```
4
5
3
```

In [ ]:
```
3.Apply Operations
```

In [10]:
```
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
    nums.sort(key=lambda x: x == 0)
    return nums
result = apply_operations([2, 2, 0, 4, 4])
print(result)
```

```
[4, 8, 0, 0, 0]
```

In [ ]:
```
4.Maximum Sum of Distinct Subarrays With Length k
```

In [11]:
```
def max_sum_distinct_subarrays(nums, k):
    if len(set(nums)) < k:
        return 0
    max_sum = 0
    for i in range(len(nums) - k + 1):
        if len(set(nums[i:i+k])) == k:
            max_sum= max(max_sum, sum(nums[i:i+k]))
    return max_sum
nums = [1, 5, 4, 2, 9, 9, 9]
k = 3
print(max_sum_distinct_subarrays(nums, k))
nums = [4, 4, 4]
k = 3
print(max_sum_distinct_subarrays(nums, k))
```

```
15
0
```

In [ ]:
```
5. Total Cost to Hire K Workers
```

In [13]:
```
def total_cost_to_hire(costs, k, candidates):
    total_cost = 0
    for _ in range(k):
        min_cost = float('inf')
        min_index = -1
        for i in range(candidates):
            if costs[i] < min_cost:
                min_cost = costs[i]
                min_index = i
        for i in range(len(costs) - candidates, len(costs)):
            if costs[i]< min_cost:
                min_cost = costs[i]
                min_index = i
        total_cost += min_cost
        costs.pop(min_index)
    return total_cost
costs = [10, 20, 5, 30, 25, 15]
k = 3
candidates = 2
print(total_cost_to_hire(costs, k, candidates))
```

```
30
```

In [ ]: