Citadel城堡

1.Triplets

1. Triplets

Given an array of *n* distinct integers, *d = [d[0], d[1], ..., d[n−1]]*, and an integer threshold, *t*, how many *(a, b, c)* index triplets exist that satisfy both of the following conditions?

- *d[a] < d[b] < d[c]*
- *d[a] + d[b] + d[c] ≤ t*

Example
d = [1,2,3,4,5]
t = 8
The following 4 triplets satisfy the constraints:

```
(1,2,3) → 1 + 2 + 3 = 6 ≤ 8
(1,2,4) → 1 + 2 + 4 = 7 ≤ 8
(1,2,5) → 1 + 2 + 5 = 8 ≤ 8
(1,3,4) → 1 + 3 + 4 = 8 ≤ 8
```

Function Description
Complete the function *triplets* in the editor below.

triplets has the following parameter(s):
int t: an integer threshold
int d[n]: an array of integers

Returns:
long: a long integer that denotes the number of *(a, b, c)* triplets that satisfy the given conditions

Codes:

```
# brute force: O(n^3)
# Optimized: O(n*2)

def triplet(d, t):
    d.sort()
    n = len(d)
    res = 0
    for i in range(n-2):
        j, k = i+1, n-1
```

```
        while (j < k):
            summ = d[i] + d[j] + d[k]
            if summ <= t:
                # 因为是sort过的list，所以如果k位置符合条件，那么[j+1, k-1]位置上的值也都符合条件，一并加入答案
                res += (k-j)
                j += 1
            else:
                k -= 1
    return res

ans = triplet([1,2,3,4,5], t=8)
print(ans)
```

2.Ways to sum

2. Ways to Sum

An automated packaging system is responsible for packing boxes. A box is certified to hold a certain weight. Given an integer *total*, calculate the number of possible ways to achieve *total* as a sum of the weights of items weighing integer weights from *1* to *k*, inclusive.

Example
total = 8
k = 2

To reach a weight of 8, there are 5 different ways that items with weights between 1 and 2 can be combined:

- [1, 1, 1, 1, 1, 1, 1, 1]
- [1, 1, 1, 1, 1, 1, 2]
- [1, 1, 1, 1, 2, 2]
- [1, 1, 2, 2, 2]
- [2, 2, 2, 2]

Function Description
Complete the function *ways* in the editor below.

ways has the following parameter(s):
int total: the value to sum to
int k: the maximum of the range of integers to consider when summing to *total*
Returns
int: the number of ways to sum to the *total*; the number might be very large, so return the integer modulo *1000000007* (10^9+7)

Constraints

- $1 \le total \le 1000$
- $1 \le k \le 100$

▸ **Input Format For Custom Testing**

▾ **Sample Case 0**

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
5       →   total = 5
3       →   k = 3
```

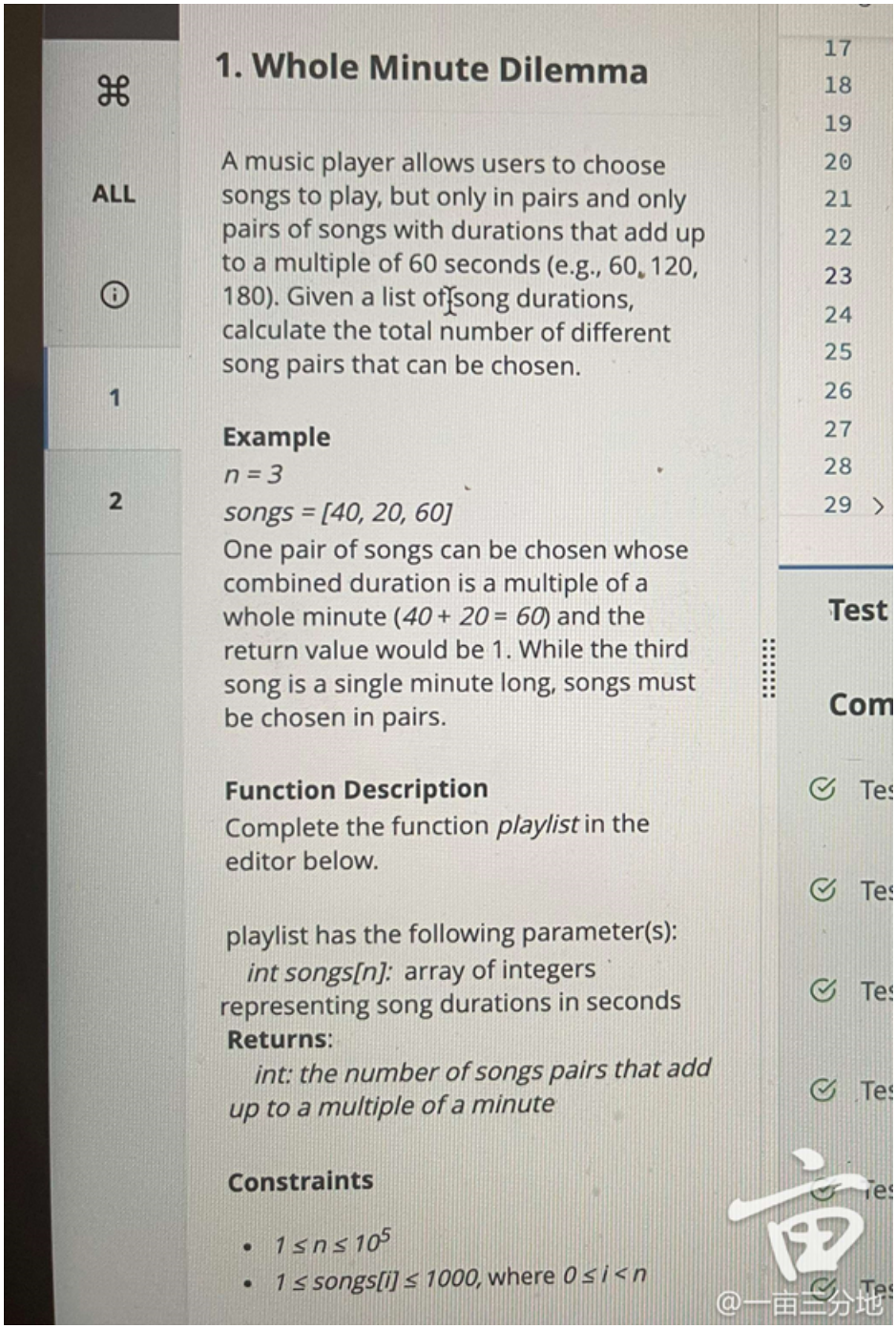
Sample Output

```
5
```

Explanation
The sum required is *5*. *k = 3* so the integers that can be considered to reach the sum are *[1, 2, 3]*.

```python
def ways(total, k):
    dp = [[0] * (total+1) for _ in range(k+1)]
    for i in range(1, k+1):
        for j in range(1, total+1):
            if i == j:
                dp[i][j] = dp[i-1][j] + 1
            elif i > j:
                dp[i][j] = dp[i-1][j]
            else:
                dp[i][j] = dp[i-1][j] + dp[i][j-i]
    return dp[k][total]
```

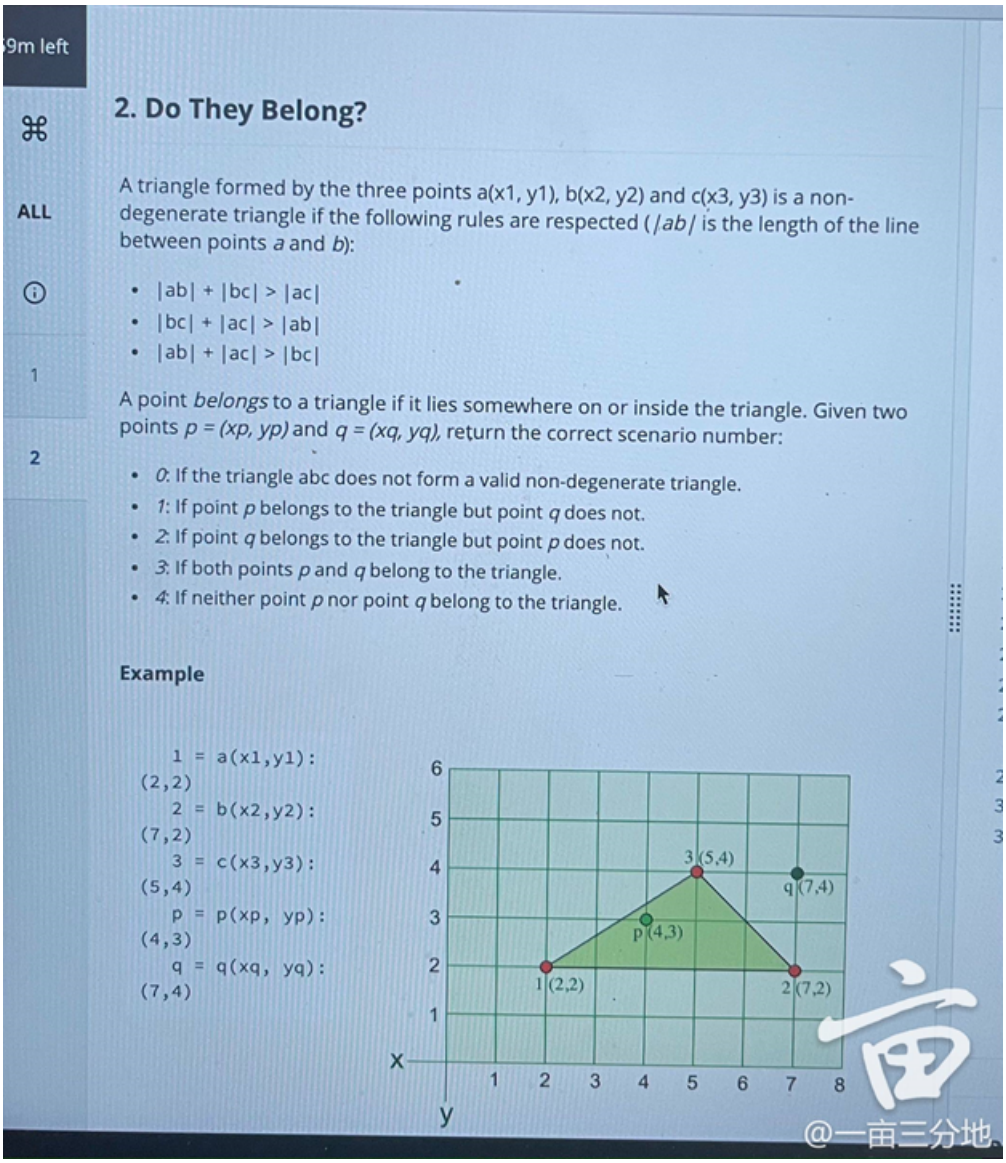
3.Whole Minute Dilemma



```python
from collections import defaultdict
def getSongPairs(songs):
    if len(songs) < 2:
        return 0
    c = defaultdict(int)
    for song in songs:
        c[song%60] += 1
    res = c[0] * (c[0]-1) // 2 + c[30] * (c[30]-1) // 2
    for i in range(1, 30):
        res += c[i] * c[60-i]
    return res

num = getSongPairs([20,40,60])
```

4.Do They Belong?




```python
import math

def check_degenerate(ax, ay, bx, by, cx, cy):
    AB = math.sqrt((ax - bx) ** 1 + (ay - by) ** 2)
    AC = math.sqrt((ax - cx) ** 1 + (ay - cy) ** 2)
    BC = math.sqrt((bx - cx) ** 1 + (by - cy) ** 2)
```

```
    if (AB + AC > BC) and (AB + BC > AC) and (BC + AC > AB):
        return True
    return False


def getArea(x1, y1, x2, y2, x3, y3):
    return 0.5 * (x1*y2 + x2*y3 + x3*y1 - x1*y3 - x2*y1 - x3*y2)


def check_inside(ax, ay, bx, by, cx, cy, x, y):
    area1 = getArea(ax, ay, bx, by, x, y)
    area2 = getArea(bx, by, cx, cy, x, y)
    area3 = getArea(ax, ay, cx, cy, x, y)
    total_area = getArea(ax, ay, bx, by, cx, cy)
    if area1 + area2 + area3 == total_area:
        return True
    return False


def decide_scenario(ax, ay, bx, by, cx, cy, px, py, qx, qy):
    if not check_degenerate(ax, ay, bx, by, cx, cy):
        return 0
    flag1 = check_inside(ax, ay, bx, by, cx, cy, px, py)
    flag2 = check_inside(ax, ay, bx, by, cx, cy, qx, qy)
    if flag1 and not flag2:
        return 1
    elif not flag1 and flag2:
        return 2
    elif flag1 and flag2:
        return 3
    else:
        return 4
```

5.Disk space analysis

1. Disk Space Analysis

A company is performing an analysis on the computers at its main office. The computers are spaced along a single row. For each group of contiguous computers of a certain length, that is, for each segment, determine the minimum amount of disk space available on a computer. Return the maximum of these values as your answer.

Example
n = 4, the number of computers
space = [8, 2, 4, 6]
x = 2, the segment length

The free disk space of computers in each of the segments is *[8, 2], [2, 4], [4, 6].* The minima of the three segments are *[2, 2, 4].* The maximum of these is *4.*

Function Description
Complete the function *segment* in the editor below.

segment has the following parameter(s):
int x: the segment length to analyze
int space[n]: the available hard disk space on each of the computers

Returns:
int: the maximum of the minimum values of available hard disk space found while analyzing the computers in segments of length *x*

Constraints

- $1 \le n \le 10^6$
- $1 \le x \le n$
- $1 \le space[i] \le 10^9$


```python
# 单调队列
from collections import deque


def add_to_dq(dq, nums, idx):
    # 维护单调队列的性质，从左到右严格递增
    while dq and nums[dq[-1]] >= nums[idx]:
        dq.pop()
    dq.append(idx)
    return


def segment(x, space):
    ans = 0
    dq = deque()
    # 初始化单调队列
```

```
    for i in range(x):
        add_to_dq(dq, space, i)
    left, right = 0, x-1
    while right < len(space):
        while True:
            if dq[0] >= left:
                # dp[0]对应的元素是当前范围内的最小值
                if space[dq[0]] > ans:  # 最小值大于ans，则更新ans
                    ans = space[dq[0]]
                break
            else:
                # 如果单调队列的开头已经超过了当前考虑的范围，循环执行popleft，直至dq[0]
>= left
                dq.popleft()
        left, right = left+1, right+1
        if right < len(space):
            add_to_dq(dq, space, right)  # 把新元素添加到单调队列中
    return ans


res = segment(2, [8, 2, 4, 6])
print(res)
```

6.Portfolio Balance

1. Portfolio Balances

An investor opens a new account and wants to invest in a number of assets. Each asset begins with a balance of 0, and its value is stored in an array using *1-based* indexing. Periodically, a contribution is received and equal investments are made in a subset of the portfolio. Each contribution will be given by *investment amount, start index, end index*. Each investment in that range will receive the contribution amount. Determine the maximum amount invested in any one investment after all contributions.

For example, start with an array of *5* elements: *investments = [0, 0, 0, 0, 0]**. The variables *left* and *right* represent the starting and ending indices, inclusive. Another variable, *contribution*, is the new funds to invest per asset. The first investment is at index *1*.

```
    left right contri-      investments
               bution
                      [  0,  0,  0,  0,  0]
    1    2     10     [ 10, 10,  0,  0,  0]
    2    4     5      [ 10, 15,  5,  5,  0]
    3    5     12     [ 10, 15, 17, 17, 12]
```

In the first round, a contribution of *10* is made to investments *1* and *2*. In the second round, a contribution of *5* is made to assets *2*, *3* and *4*. Finally, in the third round, *a contribution of 12* is added to investments *3*, *4* and *5*. The maximum invested in any one asset is *17*.

***Note:** The *investments* array is not provided in the function. It is to be created after the number of assets available is known.

Function description

Complete the *maxValue* function in the editor below.

maxValue has the following parameters:
int n: the number of investments available
int rounds[o][3]: each *rounds[i]* contains *3* integers, *[left, right, contribution]*
Returns:
int: the maximum invested in any one asset

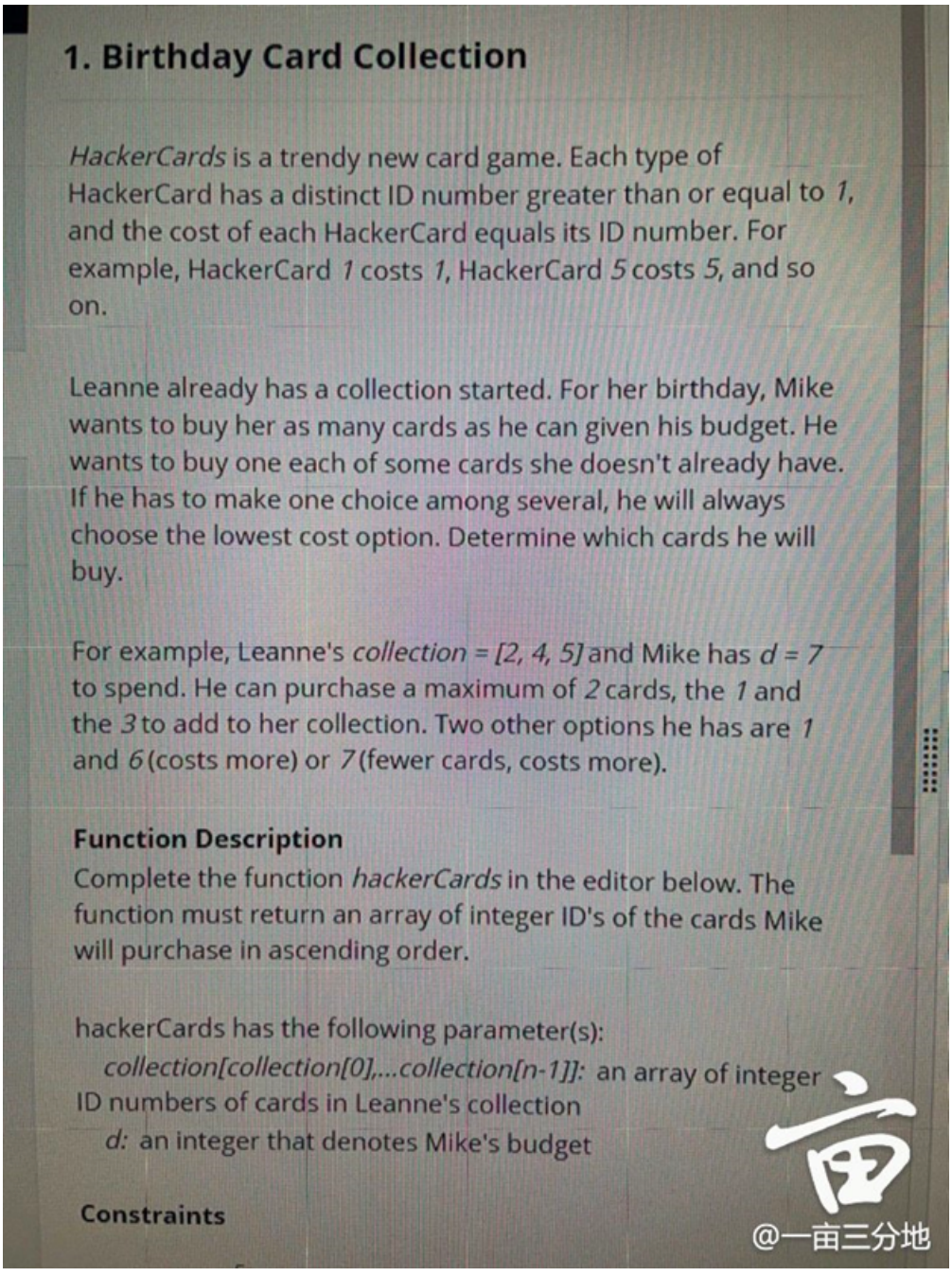
Constraints

- $3 \le n \le 10^7$
- $1 \le o \le 2 \times 10^5$
- $1 \le left \le right \le n$
- $0 \le contribution \le 10^9$

```python
# 没太搞懂逻辑，抄别人的
def maxValue(n, rounds):
    value = [0] * (n+1)
    for left, right, inv in rounds:
        value[left] += inv
        if right < n:
            value[right+1] -= inv
    ans = value[0]
    for i in range(2, n+1):
        value[i] += value[i-1]
        ans = max(ans,value[i])
    return ans

res = maxValue(5, [[1,2,10], [2,4,5], [3,5,12]])
print(res)
```

7.Birthday Card Collection



1. Birthday Card Collection

HackerCards is a trendy new card game. Each type of HackerCard has a distinct ID number greater than or equal to *1*, and the cost of each HackerCard equals its ID number. For example, HackerCard *1* costs *1*, HackerCard *5* costs *5*, and so on.

Leanne already has a collection started. For her birthday, Mike wants to buy her as many cards as he can given his budget. He wants to buy one each of some cards she doesn't already have. If he has to make one choice among several, he will always choose the lowest cost option. Determine which cards he will buy.

For example, Leanne's *collection = [2, 4, 5]* and Mike has *d = 7* to spend. He can purchase a maximum of *2* cards, the *1* and the *3* to add to her collection. Two other options he has are *1* and *6* (costs more) or *7* (fewer cards, costs more).

Function Description

Complete the function *hackerCards* in the editor below. The function must return an array of integer ID's of the cards Mike will purchase in ascending order.

hackerCards has the following parameter(s):

collection[collection[0],...collection[n-1]]: an array of integer ID numbers of cards in Leanne's collection

d: an integer that denotes Mike's budget

Constraints

```python
def hacker_cards(collections, d):
    max_val = max(collections)
    max_val = max(max_val, d)
    c_set = set(collections)
    res = set()
    for i in range(1, max_val+1):
        if i <= d and i not in c_set and i not in res:
            res.add(i)
            d = d - i
    return sorted(list(res))

c = hacker_cards([2,3,4,5], 7)
print(c)
```

8.Prime Factor Visitation

2. Prime Factor Visitation

Alex entered a room with some fancy lights arranged in a row which were either on or off. Alex had a list of numbers and worked through each number on that list in order. For each number, Alex visited the bulbs at positions which were a multiple of the prime factors of the chosen number. Whenever a bulb was visited, its state was flipped. Determine the final state of each bulb after the numbers on the list were processed.

For example, given *n=10* bulbs, initially *states = [1 1 0 0 1 1 0 1 1 1]*, where *0* means *off* and *1* means *on*, and a list of *m = 3* numbers, *numbers = [3, 4, 15]*. The states of the bulbs after processing each number are as follows, where a bolded state means this bulb's state was flipped.

Initial state
[1 1 0 0 1 1 0 1 1 1]
numbers[0] = 3, there is one prime factor: *{3}*. After the states are changed, affected bulbs in bold:
*[1 1 **1** 0 1 **0** 0 1 **0** 1]*
numbers[1] = 4, there is one prime factor: *{2}*. The states of the bulbs and the affected bulbs are
*[1 **0** 1 **1** 1 **1** 0 **0** 0 **0**]*
numbers[2] = 15, the prime factors are *{3, 5}*. The states of the bulbs and the affected bulbs are
*[1 0 **0** 1 1 **0** 0 0 **1** 0]*
*[1 0 0 1 **0** 0 0 0 1 **1**]*

The final states are *1 0 0 1 0 0 0 0 1 1*.

Function Description

Complete the function *lightBulbs* in the editor below. The function must return an array of integers that denote the final states of each bulb.

lightBulbs has the following parameter(s):
states[states[1],...states[n]]: an array of *n* integers that denote the initial states of each bulb
numbers[numbers[0],...numbers[m-1]]: an array of *m* integers, the elements in her list

Constraints

- $1 \le n \le 10^5$
- $1 \le m \le 10^5$
- $0 \le states[i] \le 1$
- $1 \le numbers[i] \le 2 \cdot 10^5$

```
def find_primes(n):
    primes = set()
    factor = 2
    while factor <= n:
        if n % factor == 0:
            primes.add(factor)
            while n % factor == 0:
                n = n // factor
        factor += 1
    return list(primes)


def light_bulbs(states, numbers):
    d = dict()
    final_states = states[:]
    for num in numbers:
        prime_list = find_primes(num)
        for prime in prime_list:
            if prime in d:
                d[prime] += 1
            else:
                d[prime] = 1

    for pos, cnt in d.items():
        if cnt % 2 == 0:
            continue
        else:
            for i in range(pos-1, len(final_states), pos):
                final_states[i] = final_states[i] ^ 1

    return final_states


init_states = [1, 1, 0, 0, 1, 1, 0, 1, 1, 1]
print(light_bulbs(init_states, [3, 4, 15]))
```

9.Profit Targets

2. Profit Targets

A financial analyst is responsible for a portfolio of profitable stocks represented in an array. Each item in the array represents the yearly profit of a corresponding stock. The analyst gathers all distinct pairs of stocks that reached the target profit. Distinct pairs are pairs that differ in at least one element. Given the array of profits, find the number of distinct pairs of stocks where the sum of each pair's profits is exactly equal to the target profit.

Example

stocksProfit = [5, 7, 9, 13, 11, 6, 6, 3, 3]
target = 12 profit's target

- There are *4* pairs of stocks that have the sum of their profits equals to the target 12 . Note that because there are two instances of *3* in *stocksProfit* there are two pairs matching (9, 3): *stocksProfits* indices 2 and 7, and indices 2 and 8, but only one can be included.
- There are *3* distinct pairs of stocks: *(5, 7)*, *(3, 9)*, and *(6, 6)* and the return value is *3*.

Function Description

Complete the function *stockPairs* in the editor below.

stockPairs has the following parameter(s):
int stocksProfit[n]: an array of integers representing the stocks profits
target: an integer representing the yearly target profit
Returns:
int: the total number of pairs determined

Constraints

- $1 \le n \le 5 \times 10^5$
- $0 \le stocksProfit[i] \le 10^9$
- $0 \le target \le 5 \times 10^9$


```python
from collections import Counter

def stockPairs(profits, target):
    counts = Counter(profits)
    used = set()
    ans = 0
    for key, cnt in counts.items():
        if target - key in counts:
            if key not in used and target-key not in used:
                if key != target-key:
                    ans += 1
                else:
                    if counts[key] >= 2:
                        ans += 1
                used.add(key)
                used.add(target-key)
    return ans
```

```
num = stockPairs([5,7,9,13,11,6,6,3,3], 12)
print(num)
```

10.Sprint Training

Sprint Training

Imagine there are n points along a straight trail, while a runner run sprints of intervals between those point.
The training plan is an array a[], which implies the runner should run from point a[i] to point a[i+1].

```
For example, given n = 10, a = [2, 4, 1, 2].
The runner should run from point 2 to point 4,
then turn back from point 4 to point 1,
and then from point 1 to point 2.
```

Find the point that visited the most by runner after he finished training, i.e. in above example, point 2 is the most visited.
If more than one point are visited the most, find the point with minimum index.

```
def find_most_visited(train):
    n = len(train)
    count = dict()
    for i in range(n-1):
        if train[i] <= train[i+1]:
            curr = train[i]
            while curr <= train[i+1]:
                if curr in count:
                    count[curr] += 1
                else:
                    count[curr] = 1
                curr += 1
        else:
            curr = train[i]
            while curr >= train[i+1]:
                if curr in count:
                    count[curr] += 1
                else:
                    count[curr] = 1
                curr -= 1

    ans, max_cnt = 0, 0
    for key, cnt in count.items():
        if cnt > max_cnt:
            ans = key
            max_cnt = cnt
    return ans


idx = find_most_visited([2,4,1,2])
print(idx)
```

```java
// 别人写的， 优化过的方法
public int GetMostVisited(int markerCount, int[] sprints) {
    int[] incremental = new int[markerCount + 2];
    for (int i = 0; i < sprints.length; i++) {
        incremental[Math.min(sprints[0], sprints[1])]++;
        incremental[Math.max(sprints[0], sprints[1]) + 1]--;
    }

    int[] scores = new int[markerCount + 1];
    int score = 0;

    for (int i = 1; i < markerCount + 1; i++) {
        score += incremental[i];
        scores[i] = score;
    }

    int ans = 0;
    for (int i = 1; i < markerCount + 1; i++) {
        if (scores[i] > scores[ans]) {
            ans = i;
        }
    }
    return ans;
}



}
```