AMAZON OA 整理

1. getkConsistency score

删掉最多K个元素得到元素相同的subarray，这个subarray最长的可能长度。思路：
hashmap<int,vector> key是stockprice，value是出现的index。然后对于每个元素做sliding window，找到最长的可能。

https://www.1point3acres.com/bbs/thread-917161-1-1.html

6m left

ALL

1

2

2. Code Question 2

A team of financial analysts at Amazon has designed a stock indicator to determine the consistency of Amazon's stock in delivering returns daily. More formally, the percentage return (rounded off to nearest integer) delivered by the stock each day over the last *n* days is considered. This is given as an array of integers, *stock_prices.* The stock's *k-consistency score* is calculated using the following operations:

- In a single operation, omit a particular day's return from the *stock_prices* array to get have one less element, then rejoin the parts of the array. This can be done at most *k* times.
- The maximum number of contiguous days during which the daily return was the same is defined as the *k-consistency score* for Amazon's stock. Note that the return may be positive or negative.

As part of the team, you have been assigned to determine the *k*-consistency score for the stock. You are given an array *stock_prices* of size *n* representing the daily percentage return delivered by Amazon stock and a parameter *k*.

Determine the *k*-consistency score.

Example:
Consider the percentage return delivered by Amazon's Stock in the last 8 days is represented as *stock_prices = [1, -2, 1, 1, 3, 2, 1, -2]* and *k = 3*.

Function Description

Complete the function *getkConsistency* in the editor below.

getkConsistency has the following parameters:

int stock_prices[n]: the percentage return for Amazon stock over the last *n* days

int k: the maximum number of days that can be removed

Returns

int: the maximum possible *k-consistency score*

Constraints

- $1 \le n \le 3 \cdot 10^5$
- $-10^9 \le stock\_prices[i] \le 10^9$
- $0 \le k \le 3 \cdot 10^5$

▶ Input Format For Custom Testing

▼ **Sample Case 0**

Sample Input For Custom Testing

```
STDIN       FUNCTION
-----       --------
6       →   n = 6
1       →   stock_prices = [1, 1, 2, 1,
2, 1]
1
2
1
2
1
```


Function Description

Complete the function *getkConsistency* in the editor below.

getkConsistency has the following parameters:

int stock_prices[n]: the percentage return for Amazon stock over the last *n* days

int k: the maximum number of days that can be removed

Returns

int: the maximum possible *k-consistency score*

Constraints

- $1 \leq n \leq 3 \cdot 10^5$
- $-10^9 \leq stock\_prices[i] \leq 10^9$
- $0 \leq k \leq 3 \cdot 10^5$

▶ **Input Format For Custom Testing**

▼ **Sample Case 0**

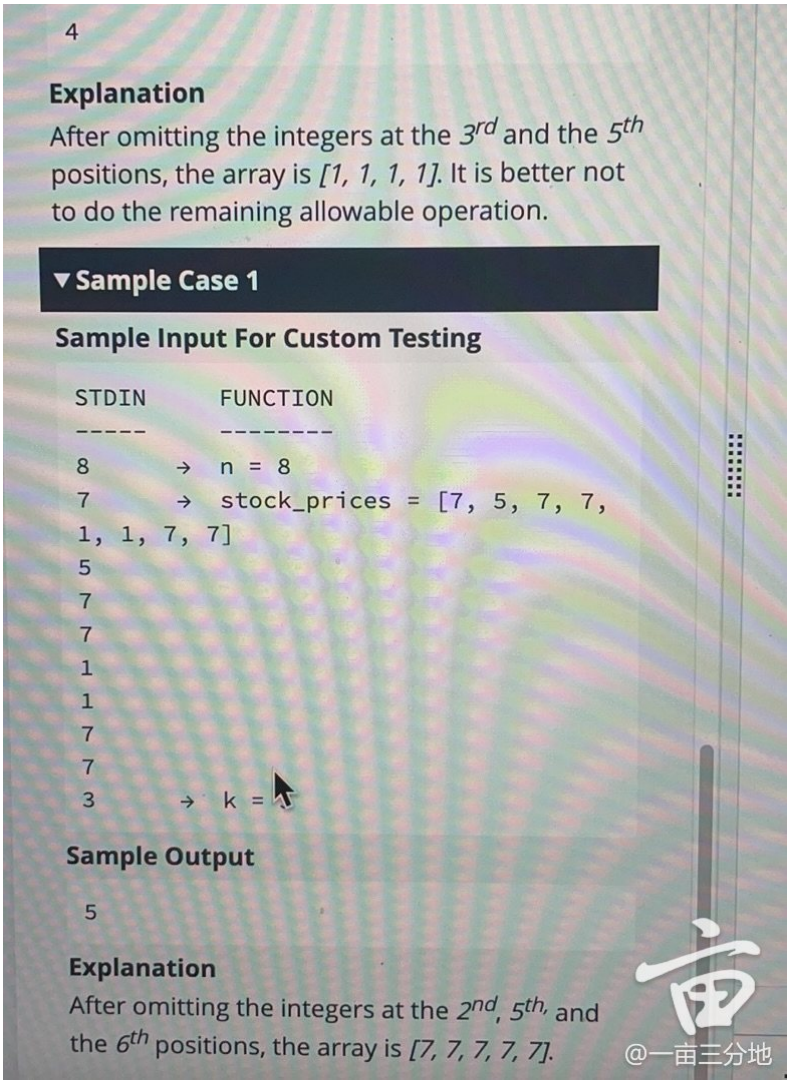
Sample Input For Custom Testing

```
STDIN       FUNCTION
-----       --------
6       →   n = 6
1       →   stock_prices = [1, 1, 2, 1,
2, 1]
1
2
1
2
1
```



```
2
1
2
1
3       →   k = 3
```

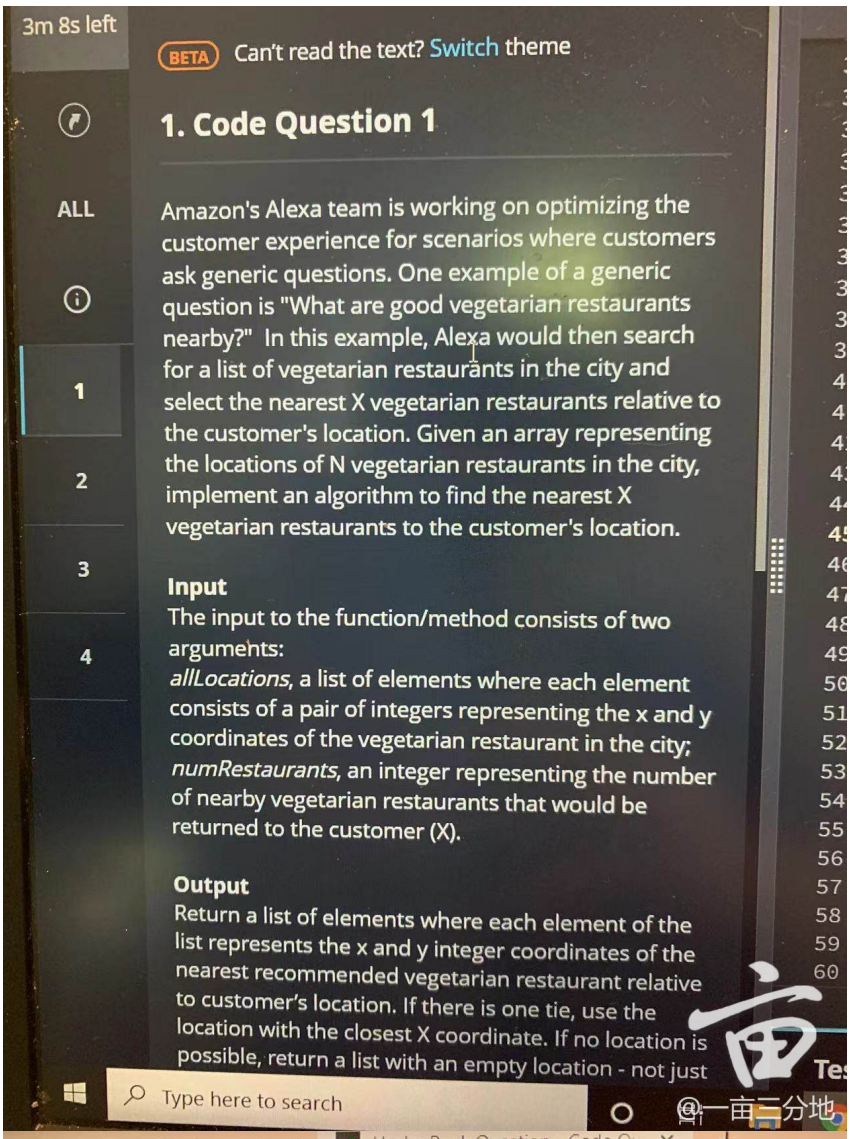
Sample Output

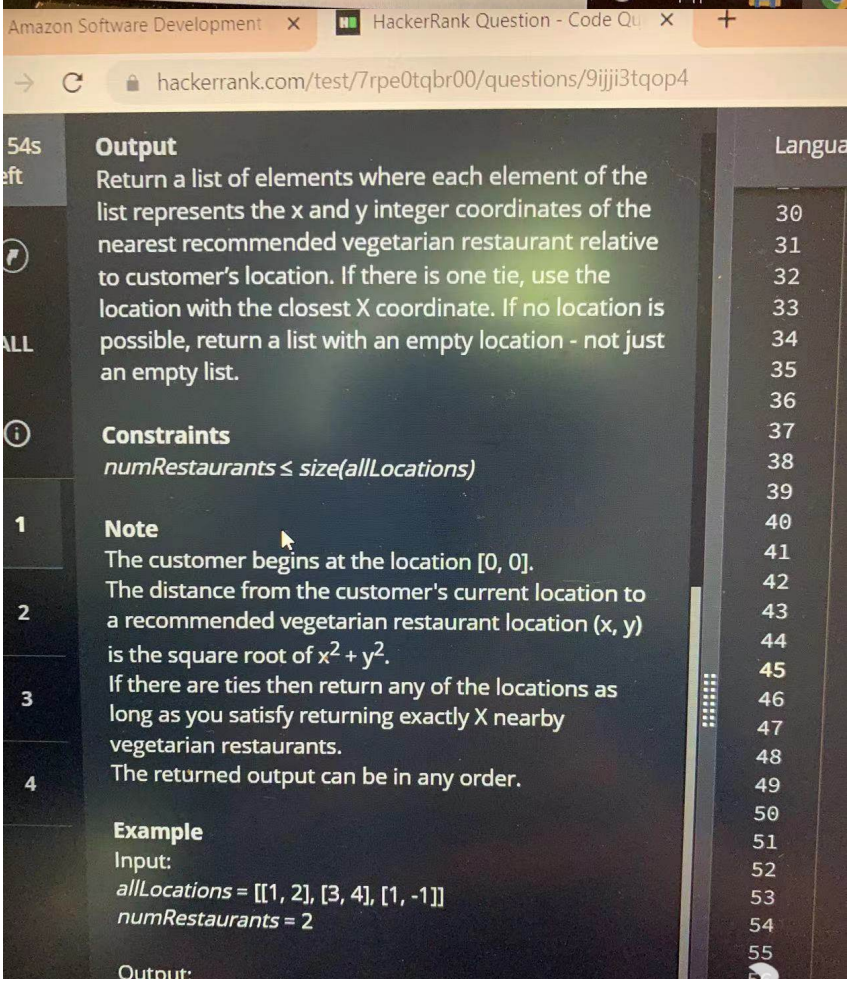


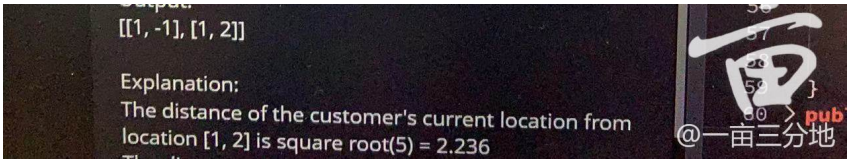

```cpp
int main() {
    //vector<int> stockPrice={1,1,2,1,2,1};
    vector<int> stockPrice={7,5,7,7,1,1,7,7};
    int k=3;
    int result=0;
    unordered_map<int,vector<int>> m;
    for(int i=0;i<stockPrice.size();i++)
        m[stockPrice[i]].push_back(i);
    for(auto tmp:m){
        int currentLength = 0;
        int n=tmp.second.size();
        if(n==1) continue;
        int deleted=tmp.second[1]-tmp.second[0]-1;
        int left=0;int right=1;
        while(left<n && right<n){
            currentLength = max(right-left+1,currentLength);
            while(deleted>k){
                left++;
                deleted -= tmp.second[left]-tmp.second[left-1]-1;
            }
            right++;
            if(right<n) deleted += tmp.second[right]-tmp.second[right-1]-1;
        }
        result = max(result,currentLength);
    }
```

```
    cout<<result;
    return 0;
}
```

2. LC 973: 找最近的餐馆


3m 8s left
BETA Can't read the text? Switch theme
1. Code Question 1
ALL
1
2
3
4
Amazon's Alexa team is working on optimizing the customer experience for scenarios where customers ask generic questions. One example of a generic question is "What are good vegetarian restaurants nearby?" In this example, Alexa would then search for a list of vegetarian restaurants in the city and select the nearest X vegetarian restaurants relative to the customer's location. Given an array representing the locations of N vegetarian restaurants in the city, implement an algorithm to find the nearest X vegetarian restaurants to the customer's location.
Input
The input to the function/method consists of two arguments:
allLocations, a list of elements where each element consists of a pair of integers representing the x and y coordinates of the vegetarian restaurant in the city;
numRestaurants, an integer representing the number of nearby vegetarian restaurants that would be returned to the customer (X).
Output
Return a list of elements where each element of the list represents the x and y integer coordinates of the nearest recommended vegetarian restaurant relative to customer's location. If there is one tie, use the location with the closest X coordinate. If no location is possible, return a list with an empty location - not just
Type here to search
@一亩三分地


Amazon Software Development
HackerRank Question - Code Qu
hackerrank.com/test/7rpe0tqbr00/questions/9ijji3tqop4
54s left
Output
Return a list of elements where each element of the list represents the x and y integer coordinates of the nearest recommended vegetarian restaurant relative to customer's location. If there is one tie, use the location with the closest X coordinate. If no location is possible, return a list with an empty location - not just an empty list.
Constraints
numRestaurants ≤ size(allLocations)
Note
The customer begins at the location [0, 0].
The distance from the customer's current location to a recommended vegetarian restaurant location (x, y) is the square root of x2 + y2.
If there are ties then return any of the locations as long as you satisfy returning exactly X nearby vegetarian restaurants.
The returned output can be in any order.
Example
Input:
allLocations = [[1, 2], [3, 4], [1, -1]]
numRestaurants = 2
Output:



Brute Force:

```
class Solution {
public:
    vector<vector<int>> kClosest(vector<vector<int>>& points, int k) {
        sort(points.begin(),points.end(),[](const auto & p1,const auto &p2)
{return p1[0]*p1[0]<p2[0]*p2[0]+p2[1]*p2[1]-p1[1]*p1[1];});
         vector<vector<int>> result;
        for(int i=0;i<k;i++)
            result.push_back(points[i]);
        return result;
    }
};
```

快排:

```
class Solution {
public:
    vector<vector<int>> kClosest(vector<vector<int>>& points, int k) {
        quickSort(points,0,points.size()-1);
        return vector<vector<int>>(points.begin(),points.begin()+k);
    }
private:
    void quickSort(vector<vector<int>>& points, int low,int high){
        if(low >= high) return;
        int pivotIndex = partition(points,low,high);
        quickSort(points,low, pivotIndex-1);
        quickSort(points,pivotIndex+1,high);
    }
    int partition(vector<vector<int>>& points, int low, int high){
        auto pivot = distance(points[low]);
        int i=low+1;
        int j=high;
        while(true){
            while(i<high && distance(points[i])<pivot) i++;
            while(j>low && distance(points[j]) > pivot) j--;
            if(i>=j)
                break;
            swap(points[i],points[j]);
            i++;
            j--;
        }
        swap(points[low],points[j]);
        return j;
    }
    long distance(vector<int>& p){
        return p[0]*p[0]+p[1]*p[1];
    }
};
```

3. combo 问题

先排序。

把所有整数相加得到最大值。

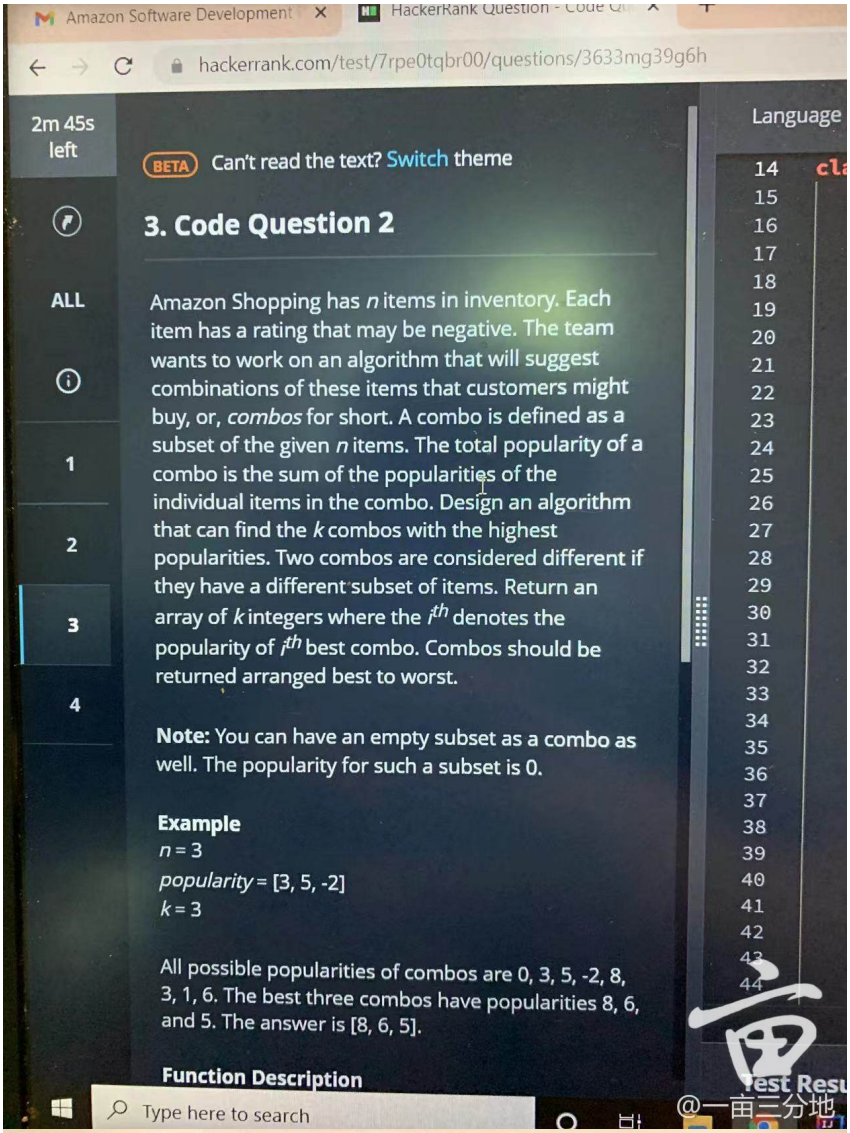
那接下来就是 减去正数 或者加上负数

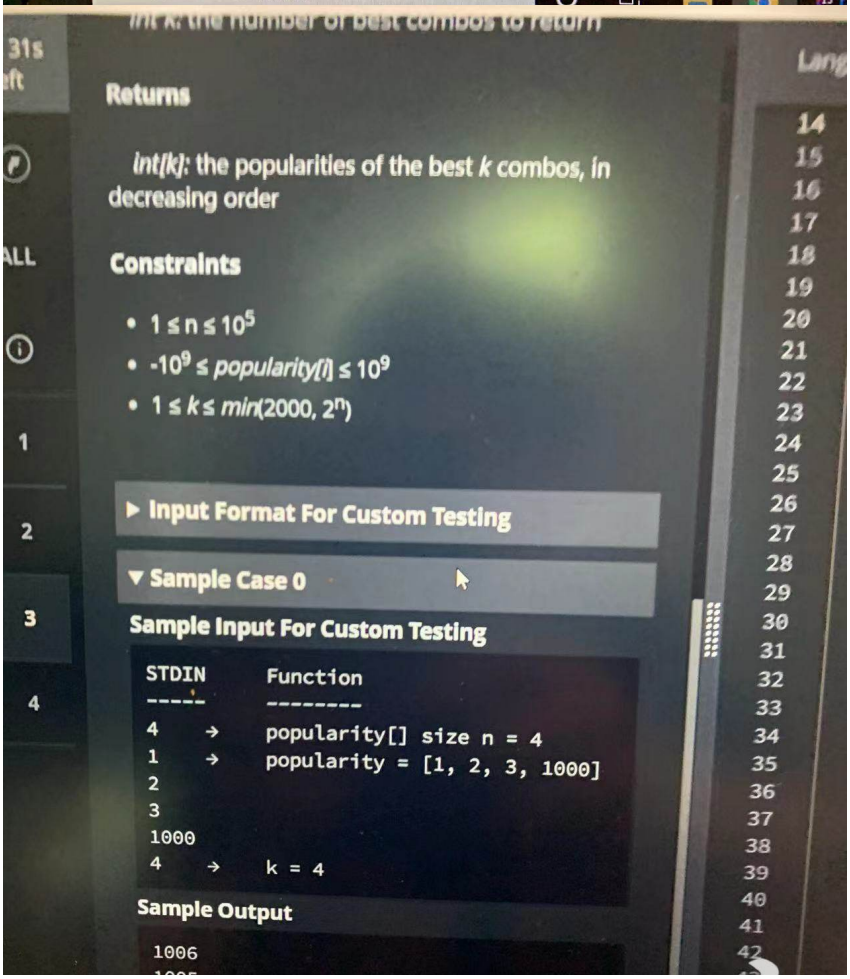
如果我们把负数都变成整数，那就统一变成了减去正数。

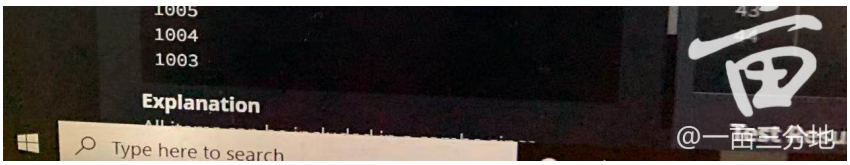
然后问题就变成了有一个array，求前k个最小的subarray的和，这题用priority_queue做，里面装pair<long,int>前者为subarray的和，后者为当前subarray的最后一个元素。

subarray move到下一个元素时要考虑两种情况：包含当前元素和不包含当前元素。

https://leetcode.com/discuss/interview-question/1895396/amazon-sde-new-grad-oa-k-most-popular-combos


Amazon Software Development
HackerRank Question - Code
hackerrank.com/test/7rpe0tqbr00/questions/3633mg39g6h
2m 45s left
Language
BETA Can't read the text? Switch theme
3. Code Question 2
ALL
1
2
3
4
Amazon Shopping has n items in inventory. Each item has a rating that may be negative. The team wants to work on an algorithm that will suggest combinations of these items that customers might buy, or, combos for short. A combo is defined as a subset of the given n items. The total popularity of a combo is the sum of the popularities of the individual items in the combo. Design an algorithm that can find the k combos with the highest popularities. Two combos are considered different if they have a different subset of items. Return an array of k integers where the i^th denotes the popularity of i^th best combo. Combos should be returned arranged best to worst.
Note: You can have an empty subset as a combo as well. The popularity for such a subset is 0.
Example
n = 3
popularity = [3, 5, -2]
k = 3
All possible popularities of combos are 0, 3, 5, -2, 8, 3, 1, 6. The best three combos have popularities 8, 6, and 5. The answer is [8, 6, 5].
Function Description
Type here to search
@一亩三分地


int k: the number of best combos to return
Returns
int[k]: the popularities of the best k combos, in decreasing order
Constraints
1 ≤ n ≤ 10^5
-10^9 ≤ popularity[i] ≤ 10^9
1 ≤ k ≤ min(2000, 2^n)
Input Format For Custom Testing
Sample Case 0
Sample Input For Custom Testing
STDIN Function
4 → popularity[] size n = 4
1 → popularity = [1, 2, 3, 1000]
2
3
1000
4 → k = 4
Sample Output
1006




```
int main() {
    //string str="aabcdea";
    //string str="alabama";
    vector<int> popularity={3,5,-2};
    int k=3;
    long maxPopularity=0; //add all positive popularity
    for(auto& p:popularity){
        if(p>0)
            maxPopularity += p;
        else
            p=abs(p);   //convert negative to positive
    }
    sort(popularity.begin(),popularity.end());
    priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>>
pq;
    //min heap storing the sum of a subarray, top is the minimum;
    //for each pair, the 1st element is the sum of a subarry, the 2nd is the
index of the last element in current subarry
    int n=popularity.size();
    pq.push(pair<int,int>(popularity[0],0));
    vector<long> subarraySum;
    while(!pq.empty() && subarraySum.size()<k-1) {
        auto tmp = pq.top();
        pq.pop();
        int currentMinSum=tmp.first;
        int currentEndIndex=tmp.second;
        subarraySum.push_back(currentMinSum);
        if(currentEndIndex+1<n) {  //add the subarry end with next element
            pq.push(pair<int,int>
(currentMinSum+popularity[currentEndIndex+1],currentEndIndex+1)); //next
subarray including current element
             pq.push(pair<int,int>(currentMinSum-
popularity[currentEndIndex]+popularity[currentEndIndex+1],currentEndIndex+1));
//next subarray withour current element
        }
    }
    vector<long> result;
    result.push_back(maxPopularity);
    for(int i:subarraySum)
        result.push_back(maxPopularity-i);
    for(auto i:result)
        cout<<i<<",";
    return 0;
}
```

4 ceil 问题

https://www.1point3acres.com/bbs/thread-918351-1-1.html

Amazon Web Services (AWS) has several processors for executing proces scheduled on its servers. There are n processes to be executed, where the ith process takes execution [i] amount of time to execute.

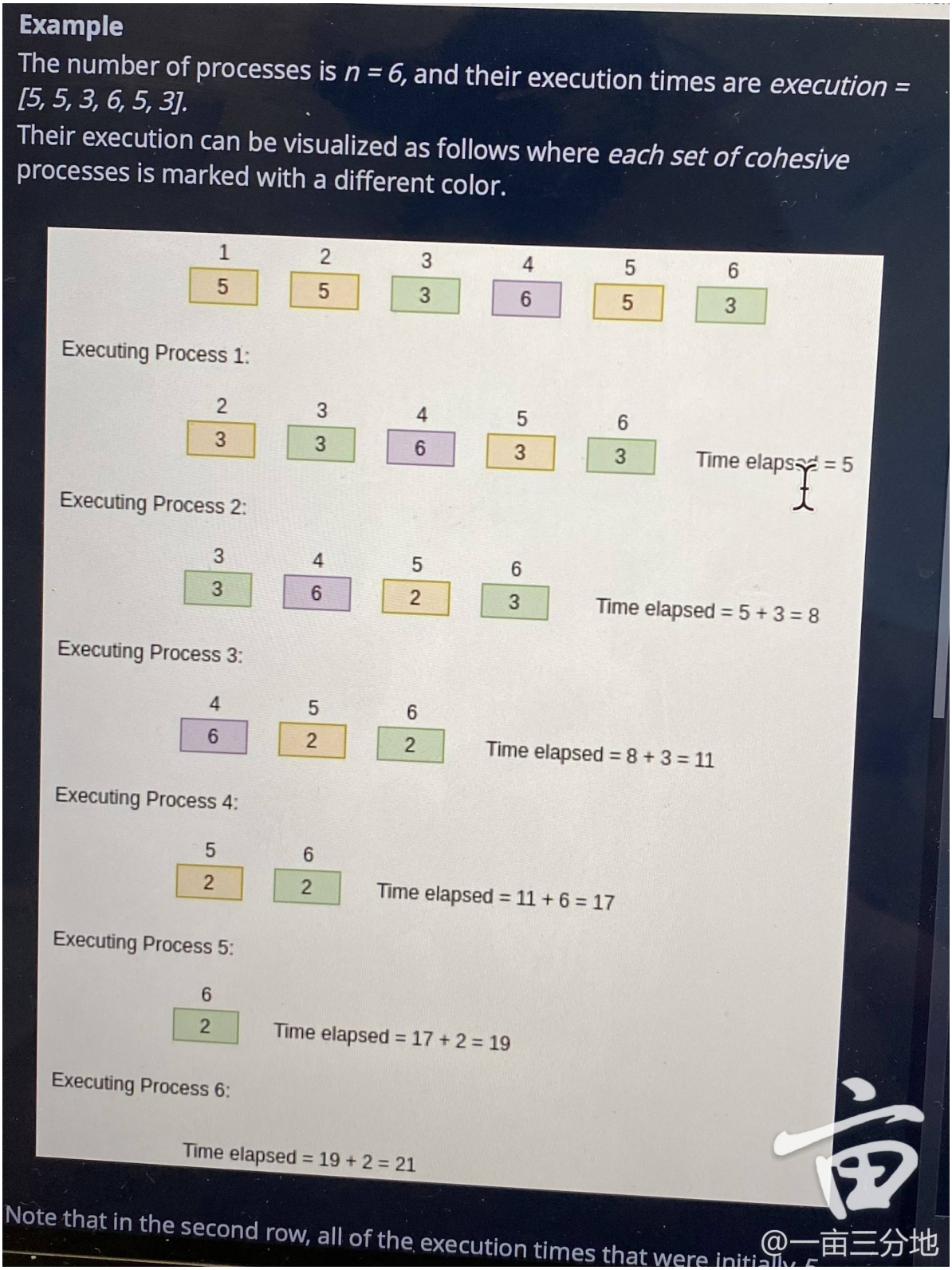
Two processes are cohesive if and only if their original execution times are equal.

When a process with execution time and simultaneously reduces the execution time of all its cohesive processe
to ceil(executionli] / 2).

Given the execution time of n processes, find the total amount of time the processor takes to execute all the processes if you execute the processes in the given order, i.e. from left to right.

Notes
The ceill() function returns the smallest integer that is bigger or equal to its argument. For example, ceil(1.1) = 2, ceil(2.5) = 3, ceil(5) = 5, etc.

If the execution time of some process is reduced and becomes equal to the execution time of any other process ;, then the two
processes /and jare not considered cohesive.
Example
The number of processes is n=6, and their execution times are execution = [5, 5, 3, 6, 5, 3].
Their execution can be visualized as follows where each set of cohesive processes is marked with a different color.
Note that in the second row, all of the execution times that were initially 5 were reduced to ceil(5/2).= 3 because they matched the amount of time to execute the first process. They are cohesive. Even though their values now equal those at original indices 3 and 6, they did not match originally. They still are not cohesive. The answer is 21.



Hash map<int,int>, key是值，value是当前的值。遍历数组，每当key再出现一次，先加一次当前value，然后把当前value变为ceil(key)/2。

```
int main() {
    int res = 0;
    vector<int> execution={5,5,3,6,5,3};
    unordered_map<int,int> m;
    for(auto i:execution){
        if(m.find(i)==m.end())
            m[i]=i;
        else
            m[i] = m[i]/2 + (m[i]/2 *2 !=m [i]);
        res += m[i];
```

```
    }
    cout<<res;
    return res;
}
//21
//{4,3,3,3}: 4+3+2+1=10
//{5, 8, 4, 4 ,8, 2} :5+8+4+2+4+2=25
```

5. 求Minimum Total Lag

给两个List center和destination， 求Minimum Total Lag。
分别用两个Priority Queue存储position，total += Math.abs(destination - center)即可.

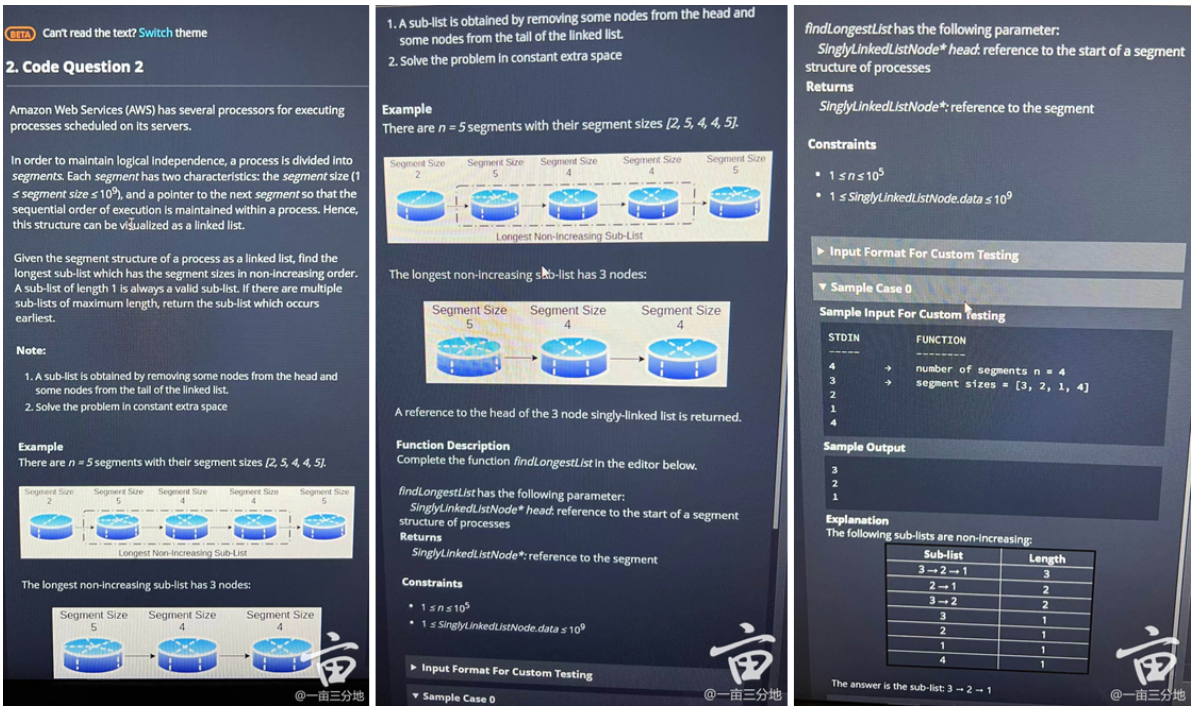
6. 找出后一个数是前一个数的平方数的序列

https://www.1point3acres.com/bbs/thread-920909-1-1.html

2，无序的整型数组，求出平方数序列的最大长度，比如对于[2, 8, 9, 16, 4, 3]应该输出3，对应的平方数序列是2，4，16。

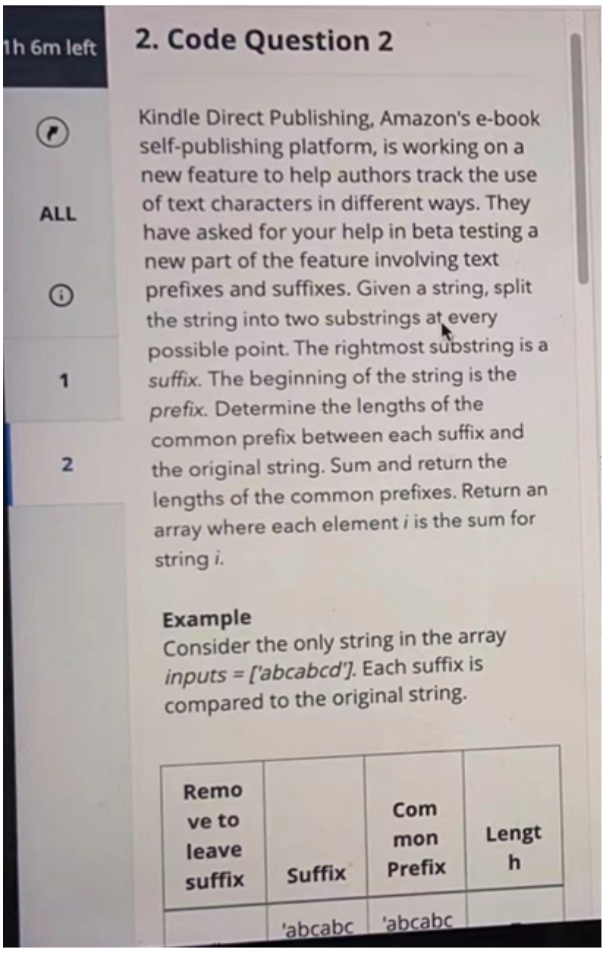
2是起始值，这样的序列对应长度是3，如果序列是2，3，5这样的就是长度0，找出符合条件的序列的最大长度

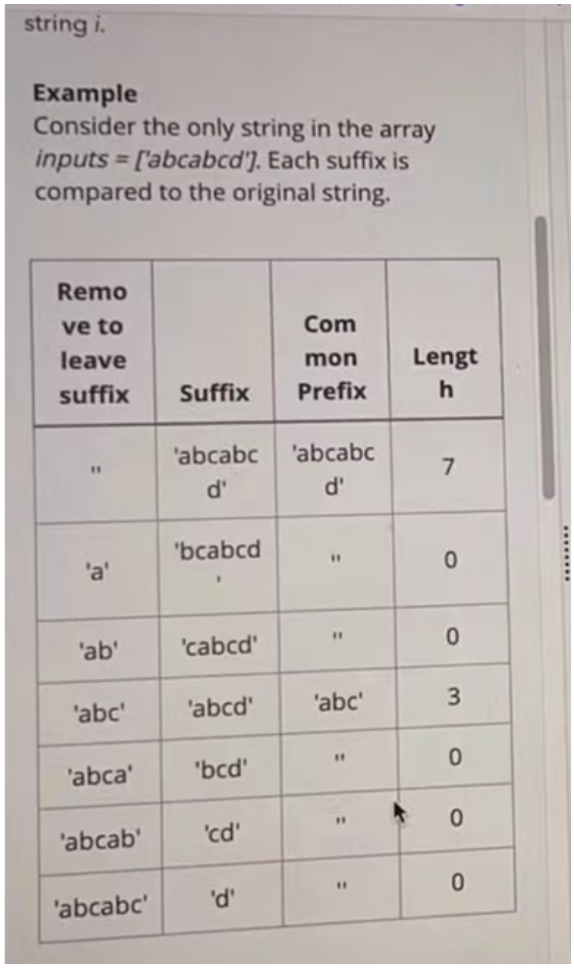
7. findLongestList



sliding window

8. Common Prefix






```
int main() {
    string s = "abcabcd";
    vector<int> res;
    res.push_back(s.length());
    for(int i=0;i<s.length()-1;i++){
        int currentL=0;
        for(int j=i+1;j<s.length();j++){
            if(s[j-(i+1)]==s[j])
                currentL++;
            else
                break;
        }
        res.push_back(currentL);
    }
    for(auto l:res)
        cout<<l<<",";
    return 0;
}
```

9. 学生排队 (findimbalance) LC 2104

给size为n的数组。数组里的数为n的permutation，比如[4, 1, 3, 2]。假设有个子数组s，定义该子数组imbalance计算方法子数组排序后s[i]-s[i-1] > 1的总个数。要求输出所有子数组imbalance的总个数

比如[4, 1, 3, 2]
子数组 4 => 排序后4，imbalance = 0
子数组 1 => 排序后1，imbalance = 0
子数组 3 => 排序后3，imbalance = 0

子数组 2 => 排序后2，imbalance = 0
子数组 4,1 => 排序后1,4，imbalance = 1 因为 4-1=3>1
子数组 1,3 => 排序后1,3，imbalance = 1 因为 3-1=2>1
子数组 3,2 => 排序后2,3，imbalance = 0 因为 3-2=1=1
子数组 4,1,3 => 排序后1,3,4，imbalance = 1 因为 3-1=2>1,4-3=1=1
子数组 1,3,2 => 排序后1,2,3，imbalance = 0 因为 2-1=1=1,3-2=1=1
子数组 4,1,3,2 => 排序后1,2,3,4，imbalance = 0 因为 2-1=1=1,3-2=1=1,4-3=1=1
最后输出结果为3

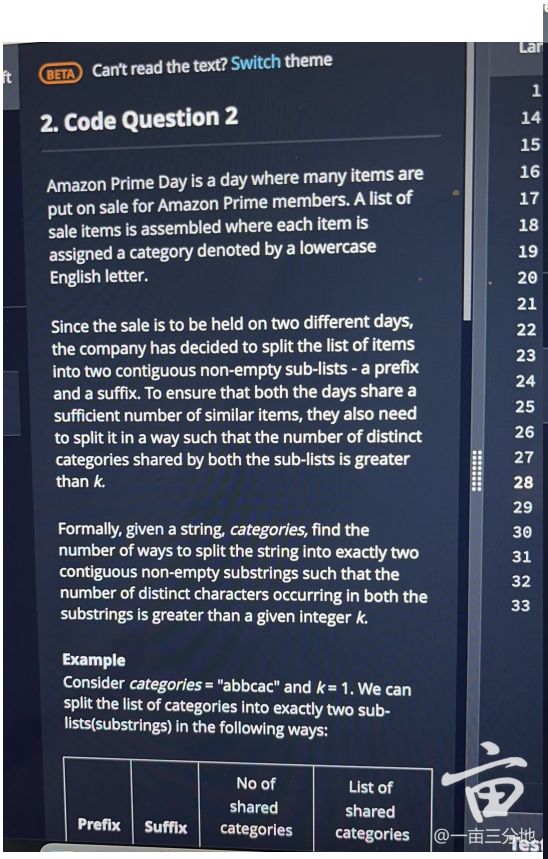
O(N) monotonic-stack

```
    public int imbalanceCount(int[] nums) {
        int n = nums.length;
        int[] indexArr = new int[n + 1];
        for (int i = 0; i < n; i++) {
            indexArr[nums[i]] = i;
        }
        int[] left = new int[n], right = new int[n];
        int[] maxStack = new int[n]; // monotonic-stack
        int index = -1;
        for (int i = 0; i < n; i++) {
            while (index >= 0 && nums[maxStack[index]] < nums[i]) {
                index--;
            }
            left[i] = index < 0 ? -1 : maxStack[index];
            maxStack[++index] = i;
        }
        index = -1;
        for (int i = n - 1; i >= 0; i--) {
            while (index >= 0 && nums[maxStack[index]] < nums[i]) {
                index--;
            }
            right[i] = index < 0 ? n : maxStack[index];
            maxStack[++index] = i;
        }
        int count = 0;
        for (int i = 0; i < n; i++) {
            if (nums[i] < n - 1) {
                int addOneIdx = indexArr[nums[i] + 1];
                if (addOneIdx > i) {
                    count += (i + 1) * (addOneIdx - right[i]);
                    if (left[i] >= 0) {
                        count += (left[i] + 1) * (right[i] - i);
                    }
                } else {
                    count += (left[i] - addOneIdx) * (n - i);
                    if (right[i] < n) {
                        count += (i - left[i]) * (n - right[i]);
                    }
                }
            }
        }
        return count;
    }
```

O(n^2):

```
# 学生排队
def find_imbalances(rank):
    count = 0
    s = set()
    for i in range(len(rank)):
        s.clear()
        max = rank[i]
        min = rank[i]
        s.add(rank[i])
        for j in range(i+1, len(rank)):
            if rank[j] - 1 in s and rank[j] + 1 in s:
                pass
            elif min > rank[j] and rank[j] + 1 in s:
                min = rank[j]
            elif max < rank[j] and rank[j] - 1 in s:
                max = rank[j]
            elif min > rank[j]:
                min = rank[j]
                count += 1
            elif max < rank[j]:
                max = rank[j]
                count += 1
            elif rank[j] - 1 in s or rank[j] + 1 in s:
                count += 1
            else:
                count += 2
            s.add(rank[j])
            print(count, s)
    return count
```

10. 分割成两个字符串使得两个substr都有的distinct char的数 > k， 有多少种方式



BETA Can't read the text? Switch theme

2. Code Question 2

Amazon Prime Day is a day where many items are put on sale for Amazon Prime members. A list of sale items is assembled where each item is assigned a category denoted by a lowercase English letter.

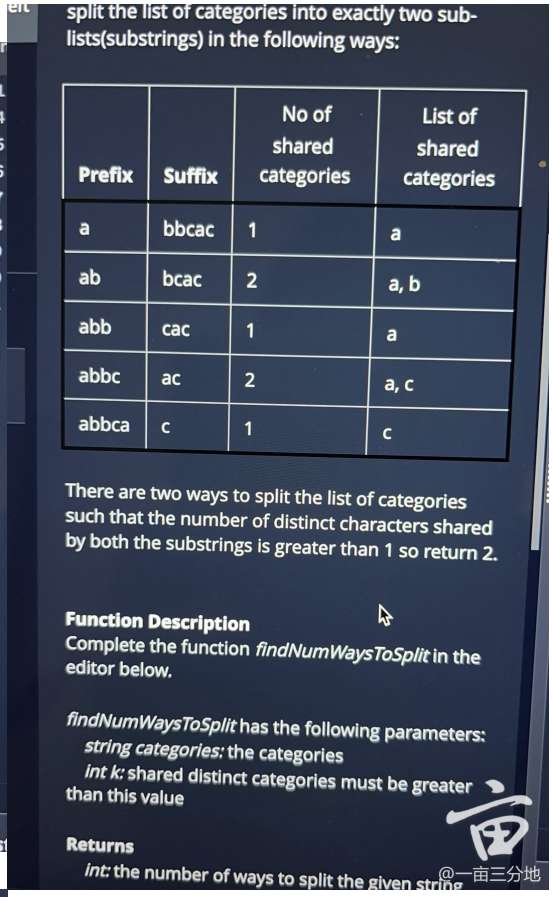
Since the sale is to be held on two different days, the company has decided to split the list of items into two contiguous non-empty sub-lists - a prefix and a suffix. To ensure that both the days share a sufficient number of similar items, they also need to split it in a way such that the number of distinct categories shared by both the sub-lists is greater than *k*.

Formally, given a string, *categories*, find the number of ways to split the string into exactly two contiguous non-empty substrings such that the number of distinct characters occurring in both the substrings is greater than a given integer *k*.

Example

Consider *categories* = "abbcac" and *k* = 1. We can split the list of categories into exactly two sub-lists(substrings) in the following ways:

Prefix	Suffix	No of shared categories	List of shared categories



split the list of categories into exactly two sub-lists(substrings) in the following ways:

Prefix	Suffix	No of shared categories	List of shared categories
a	bbcac	1	a
ab	bcac	2	a, b
abb	cac	1	a
abbc	ac	2	a, c
abbca	c	1	c

There are two ways to split the list of categories such that the number of distinct characters shared by both the substrings is greater than 1 so return 2.

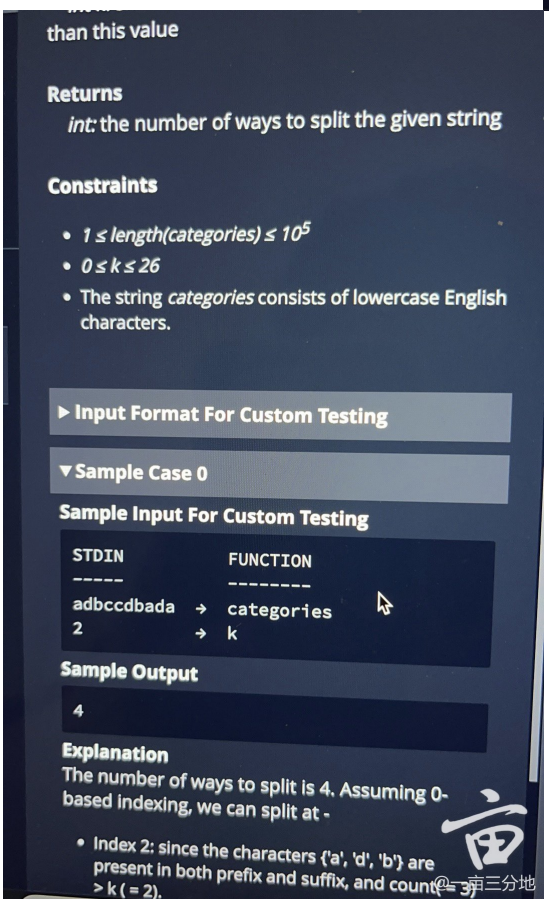
Function Description

Complete the function *findNumWaysToSplit* in the editor below.

findNumWaysToSplit has the following parameters:
string categories: the categories
int k: shared distinct categories must be greater than this value

Returns

int: the number of ways to split the given string



than this value

Returns

int: the number of ways to split the given string

Constraints

- $1 \le length(categories) \le 10^5$
- $0 \le k \le 26$
- The string *categories* consists of lowercase English characters.

▶ Input Format For Custom Testing

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN           FUNCTION
-----           --------
adbccdbada   →  categories
2            →  k
```

Sample Output

```
4
```

Explanation

The number of ways to split is 4. Assuming 0-based indexing, we can split at -

- Index 2: since the characters {'a', 'd', 'b'} are present in both prefix and suffix, and count > k (= 2).


```cpp
bool valid(unordered_map<char,vector<int>> &m,int k){
    int n=0;
    for(auto tmp:m)
        if(tmp.second[0]>0 && tmp.second[1]>0)
            n++;
    return n>k;
}
```

```
int main(){
    int res = 0;
    string categories="abbcac";
    int k=1;
    unordered_map<char,vector<int>> m;
    //key:char value: vector[0] is the freq in left substr, vector[1] is the freq
in 2nd substr
    for(auto c:categories){
        if(m.find(c)==m.end()){
            m[c].push_back(0);
            m[c].push_back(1);
        }
        else
            m[c][1]++;
    }
    for(int i=0;i<categories.size();i++){
        m[categories[i]][0]++;
        m[categories[i]][1]--;
        if(m[categories[i]][1]==0)
            m.erase(categories[i]);
        res += valid(m,k);
    }
    cout<<res;
    return 0;
}
// 2
/*
    string categories="abdccdbada";
    int k=2;
    return:4
*/
```

11. number of unique average values among the combined experience

先排序再求第一个和最后一个平均值存在set里，最后看set的size

1h 9m left

ALL

1

2

1. Code Question 1

There are *n* developers working at Amazon where the i^{th} developer has the experience points *experience[i]*. The company decided to pair the developers by iteratively pairing the developers with the highest and lowest remaining experience points for a hackathon. The *combined experience* of a pair is the average of the experience points of the two developers. Find the number of unique values among the combined experience of the pairs formed.

Example

experience = [1, 4, 1, 3, 5, 6]

There are n = 6 developers. The pairs formed are (1, 6), (1, 5), and (4, 3) making their experience points 3.5, 3, and 3.5 respectively. There are 2 distinct values, 3 and 3.5, so return 2 as the answer.

Function Description

Complete the function *findUniqueValues* in the editor below.

findUniqueValues has the following parameter:

int experience[n]: the experience points for each developer

1h 9m left

ALL

1

2

findUniqueValues has the following parameter:

int experience[n]: the experience points for each developer

Returns

int: the number of unique values among the combined experience points of the pairs formed

Constraints

- $2 \leq n \leq 10^5$
- ***n* is an even number**
- $1 \leq experience[i] \leq 10^9$

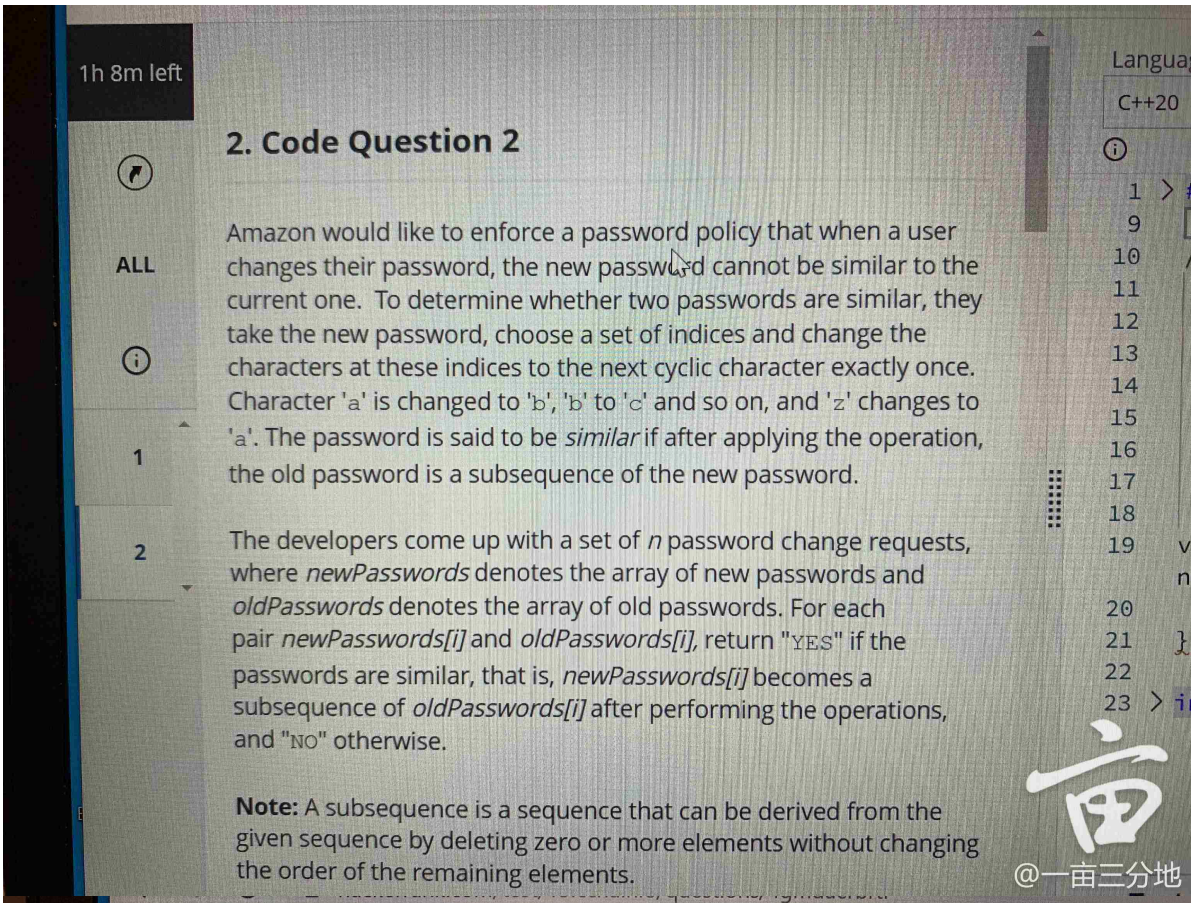
▶ **Input Format For Custom Testing**

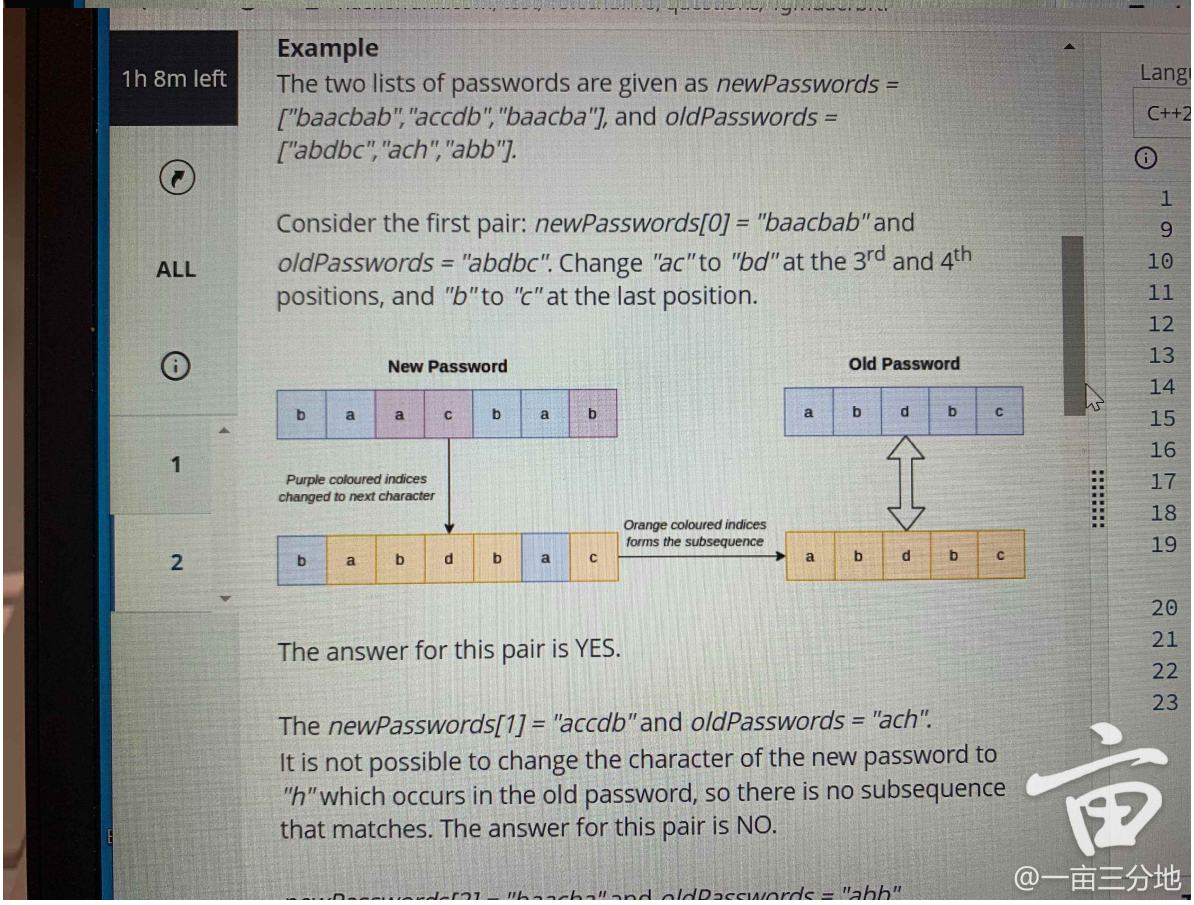
▼ **Sample Case 0**

Sample Input For Custom Testing

```
STDIN    FUNCTION
-----    --------
6     →  number of developers n = 6
1     →  experience = [1, 1, 1, 1, 1, 1]
1
1
1
```

12. 换password，对于新的password做a->b, b->c, c->d, z->a的处理，然后看新的password是不是旧的substr





```
char transform(char c){
    if(c=='z')
        return 'a';
    return c+1;
}
bool isSubstr(string newStr,string oldStr){
    if(oldStr.length()>newStr.length())
        return false;
    int pold=0;
    int pnew=0;
    while(pnew<newStr.size() && pold<oldStr.size()){
        if(newStr[pnew]==oldStr[pold] || transform(newStr[pnew])==oldStr[pold])
            pold++;
        pnew++;
    }
    return pold==oldStr.size();
}
int main(){
    vector<string> newPassword={"baacbab","accdb","baacba"};
    vector<string> oldPassword={"abdbc","ach","abb"};
    vector<bool> res;
    for(int i=0;i<newPassword.size();i++)
        res.push_back(isSubstr(newPassword[i],oldPassword[i]));
    for(auto i:res)
        cout<<i<<endl;
    return 0;
}
```

13. lexicographically smallest palindrome

排序 O（NlogN）

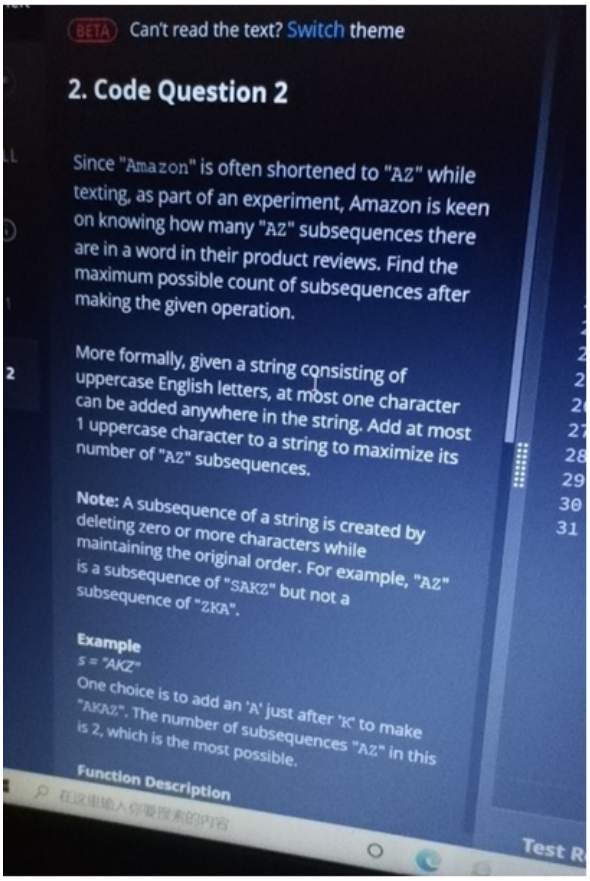
```
int l=s.length();
string res=s.substr(0,l/2);
sort(res.begin(),res.end());
int ls=l/2;
if(l%2==1) //even
    res += s[l/2];
for(int i=ls-1;i>=0;i--)
    res.push_back(res[i]);
```

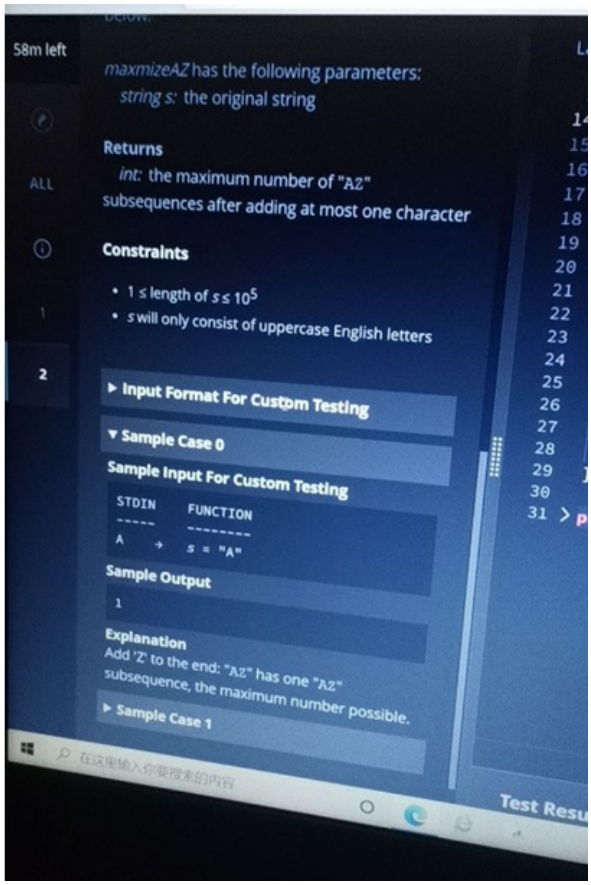
Hashmap (O(N))

```
int main(){
    string password="cbatabc";
    vector<int> v(26,0); //ind: 1-26 means 'a'-'z', record counts
    for(auto c:password)
        v[c-'a']++;
    //build first half string
    char mid;
```

```
    string res;
    for(int i=0;i<26;i++){
        string tmp(v[i]/2,'a'+i);
        res += tmp;
        if(v[i]%2==1)
        mid = 'a' + i;
    }
    //build next half
    int l=res.length();
    if(password.length()%2==1)
        res += mid;
    for(int i=l-1;i>=0;i--)
        res += res[i];
    cout<<res;
    return 0;
}
```

14 Maximize the number of AZ



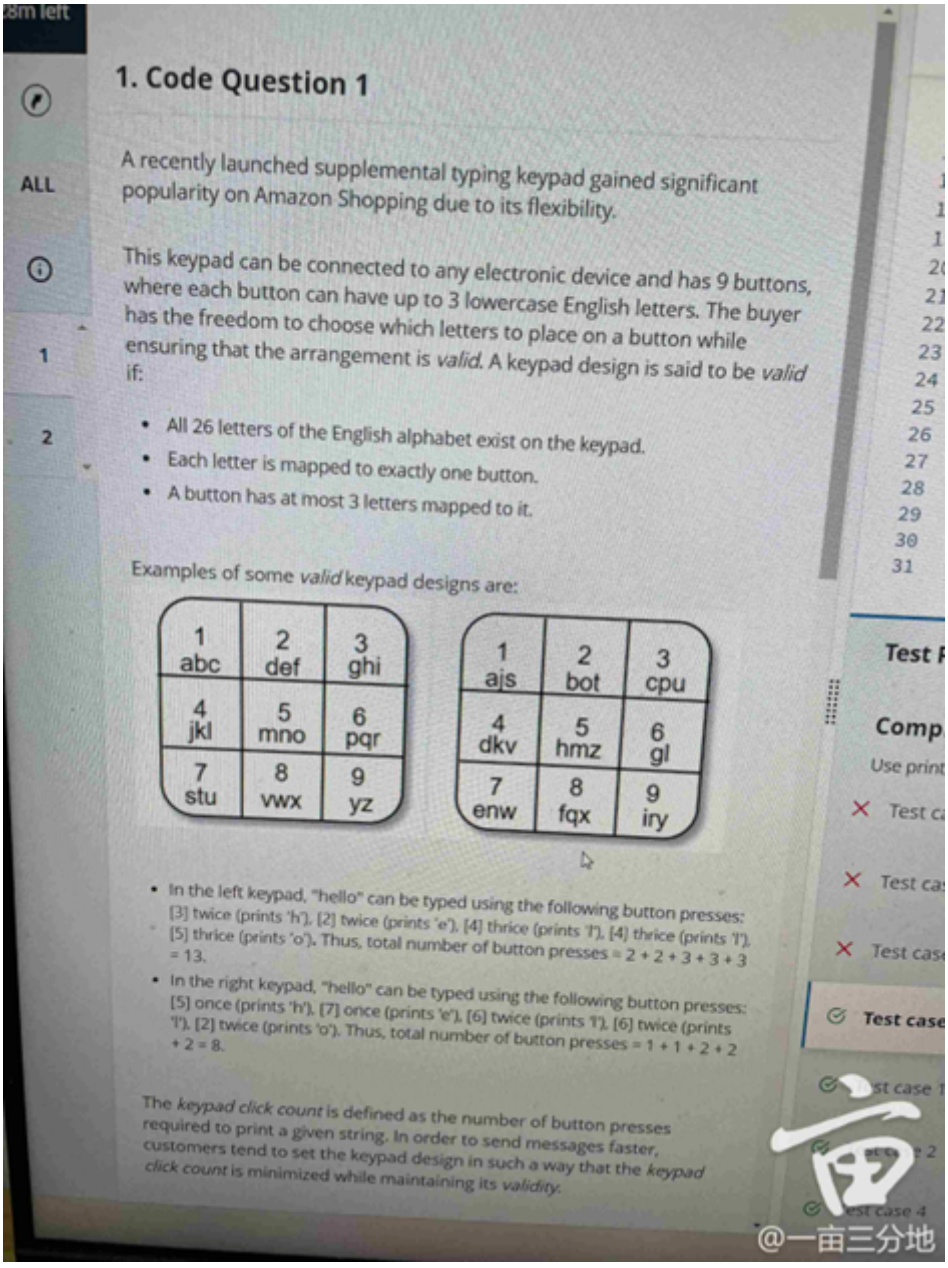


先遍历一边，碰到一个z就看他前面的a的数量，然后总数加上当前a的数量。遍历结束后看a多还是z多，总数再加上两者较大的就得到答案了。

```
int solution(string s){
    int nA=0;
    int nZ=0;
    int res=0;
    for(int i=0;i<s.length();i++){
        if(s[i]=='A')
```

```
            nA++;
        if(s[i]=='Z'){
            res += nA;
            nZ++;
        }
    }
    res += max(nA,nZ);
    return res;
}
```

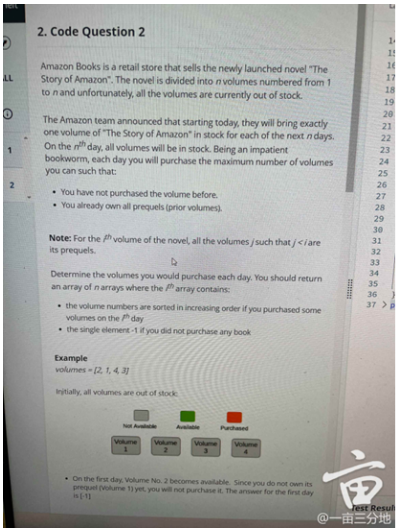
15. minimum keyboard click （9宫格）

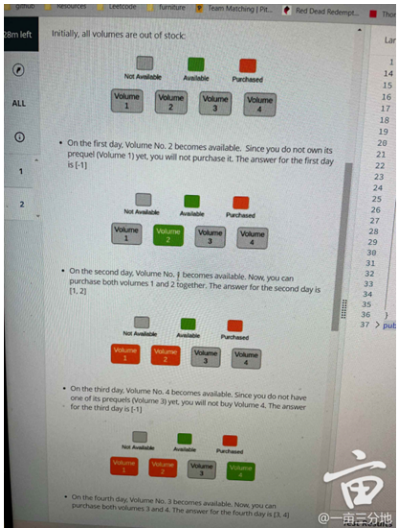


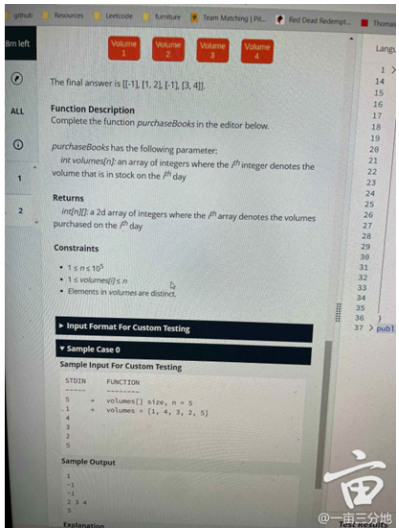

```
int minimumKeypadClickCount(string text){
        unordered_map<char,int> m;
        for(int i=0;i<text.length();i++){
            m[text[i]]++;
        }
        vector<pair<char,int>> freq(m.begin(),m.end());
        sort(freq.begin(),freq.end(),[](const auto &a,const auto& b){return
```

```
            a.second>b.second;}); //对字符按出现的次数进行排序
        vector<int> arr={1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3};
        int count = 0;
        for (int i=0;i<freq.size();i++){
            count+=freq[i].second*arr[i];
        }
        return count;
}
```

16. Amazon books retail

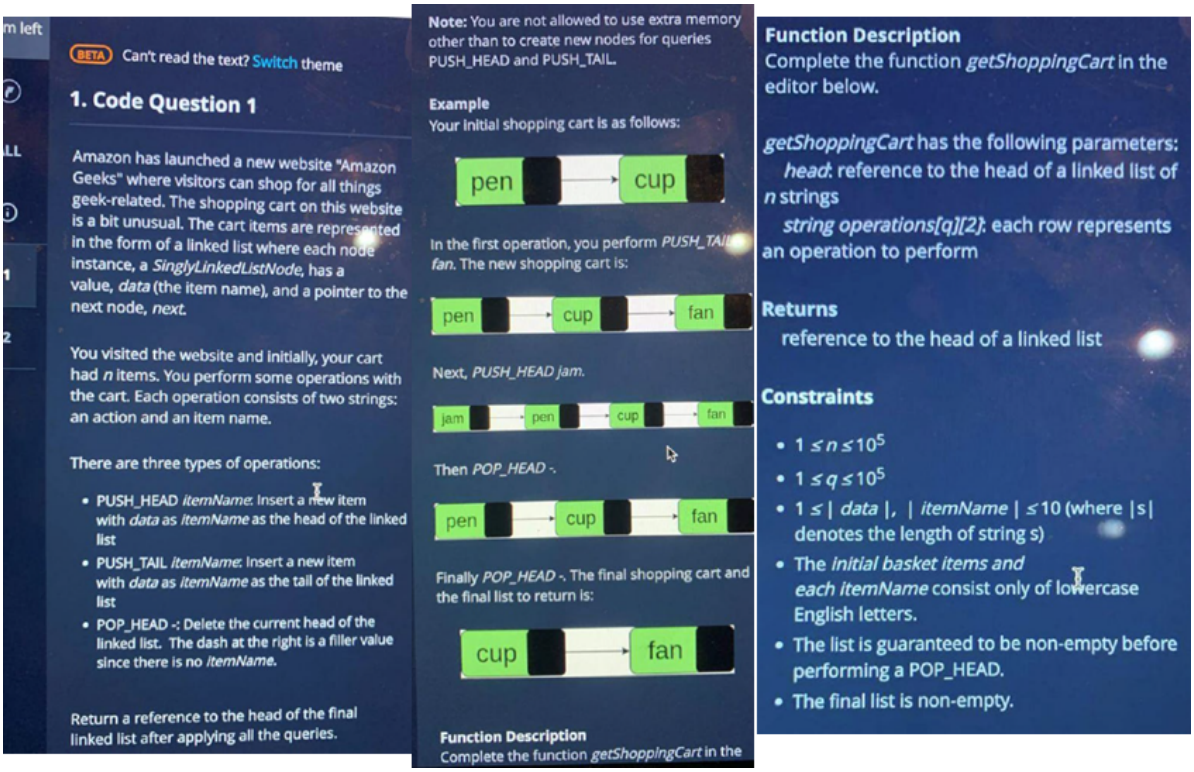







```
vector<vector<int>> bookRetail(vector<int> volumes){
    int n=volumes.size();
    //priority_queue store current books on sale
    priority_queue<int> pq;
    vector<vector<int>> result;
    int nextbook=1;
    for(int i=0;i<volumes.size();i++){
        pq.push(volumes[i]);
        vector<int> tmp;
        if(pq.size()==i+1 && pq.top()==i+1){
            for(int j=nextbook;j<=i+1;j++)
                tmp.push_back(j);
            nextbook = i+2;
            result.push_back(tmp);
        }
        else
            result.push_back(vector<int>(1,-1));
      }
  }
```

17. Amazon geeks (linkedlist shopping cart)



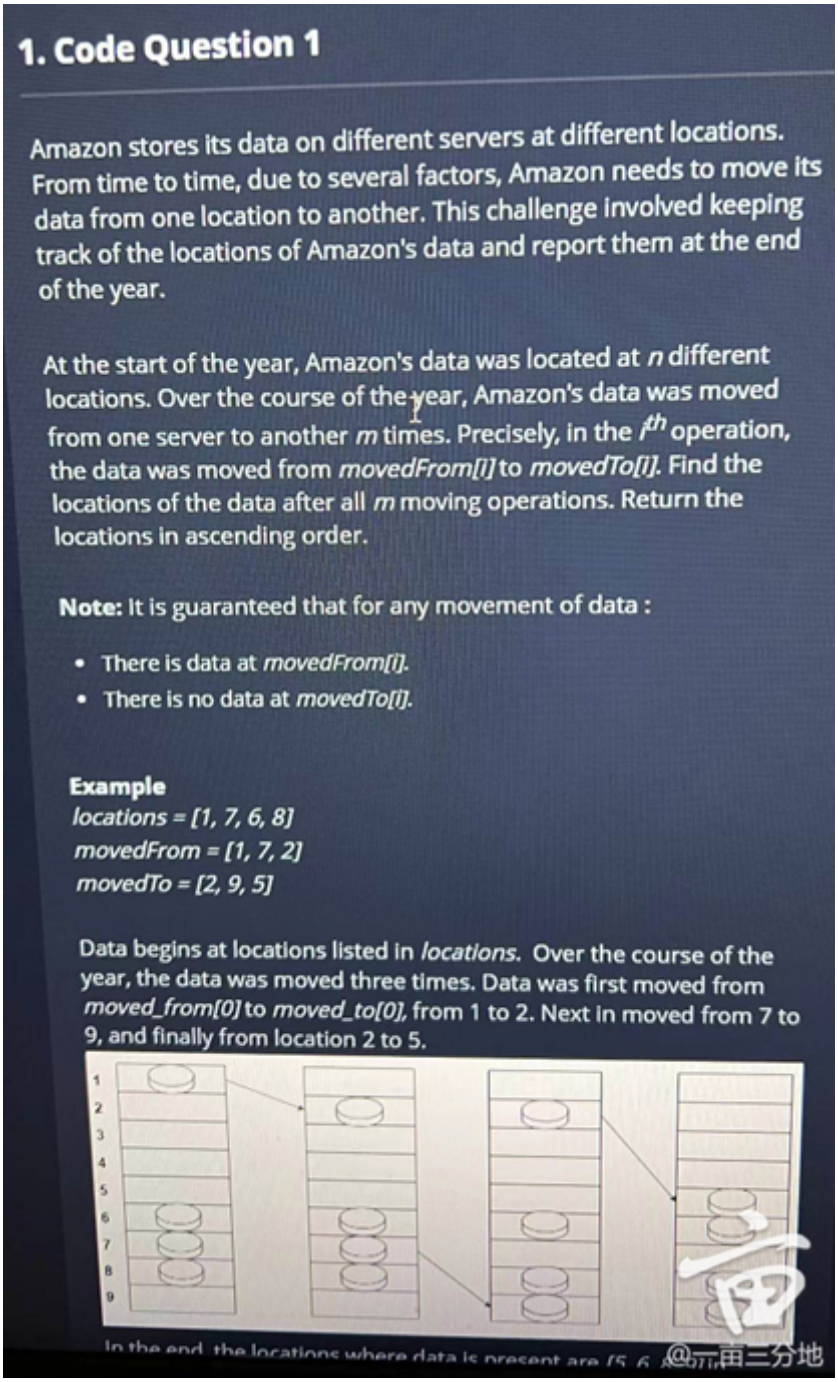
https://www.1point3acres.com/bbs/thread-919328-1-1.html

```python
#购物网站 cart 链表
def sol(head, operations):
    tail = head
    while tail.next:
        tail = tail.next
    for op in operations:
        common, item = op[0], op[1]
        if common == 'PUSH_TAIL':
            nxt = ListNode(item.val)
            if tail:
                tail.next = nxt
                tail = next
            if not head:
                head = tail
        elif common == 'PUSH_HEAD':
            nxt = ListNode(item.val)
            nxt.next = head
            head = nxt
        elif common == 'POP_HEAD':
            if head:
                head = head.next
            else:
                head = None
                tail = None

    return head
```

```java
public static ListNode getshoppingcart(ListNode head, List<List<String>> operations){
    ListNode dummy = new ListNode( val: null,head);
    ListNode tail = dummy.next;
    while (tail.next!=null){
        tail = tail.next;
    }
    for (int i=0;i< operations.size();i++){
        if(dummy.next == null){
            tail=dummy;
        }
        if (operations.get(i).get(0) == "PUSH_HEAD") {
            dummy.next = new ListNode(operations.get(i).get(1), dummy.next);
        } else if (operations.get(i).get(0) == "PUSH_TAIL") {
            ListNode node = new ListNode(operations.get(i).get(1),  next: null);
            tail.next = node;
            tail = node;
        } else if (operations.get(i).get(0) == "POP_HEAD") {
            assert dummy.next != null;
            dummy.next = dummy.next.next;
        }
    }
    return dummy.next;
}
```

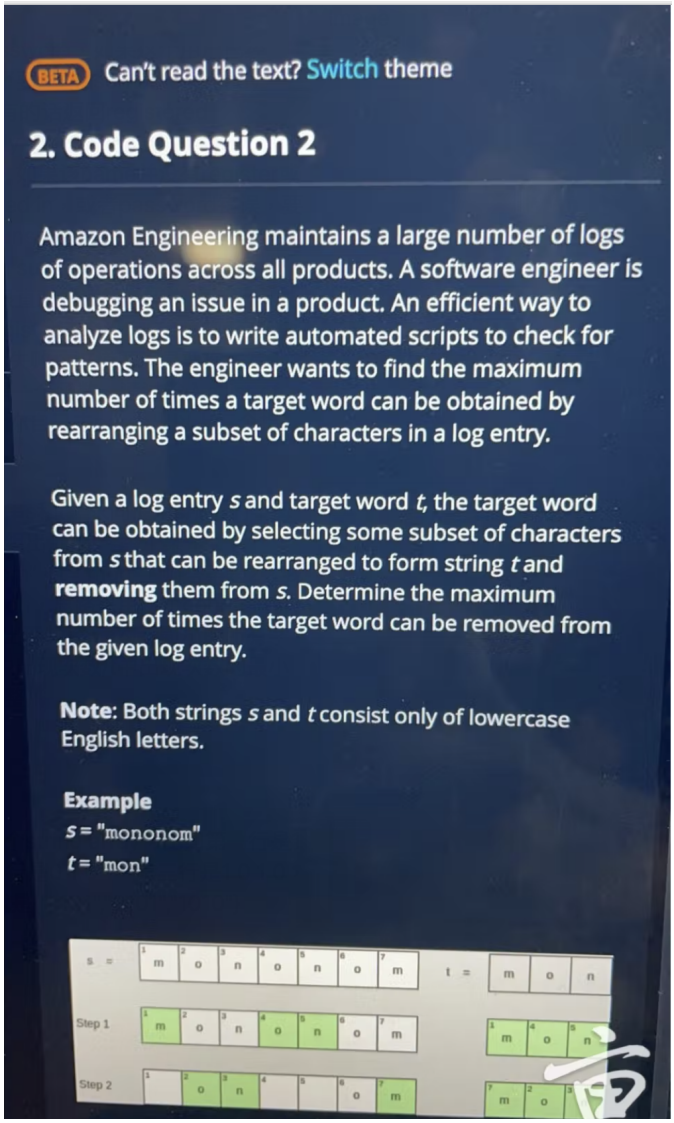

@一亩三分地

18. 服务器移动




```
vector<int> movedata(vector<int> data,vector<int> movefrom,vector<int> moveto){
        vector<int> data = {1,7,6,8};
        vector<int> movefrom={1,7,2};
        vector<int> moveto = {2,9,5};
        set<int> s(data.begin(),data.end());
        for(int i=0;i<movefrom.size();i++){
            s.erase(movefrom[i]);
            s.insert(moveto[i]);
        }
        vector<int> res(s.begin(),s.end());
        for(auto tmp:res)
            std::cout << tmp << std::endl;
        return res;
}
```

19. s里可以拼出多少个t



给两个字符串s和t，求问使用s所有的字母最多能够重组出几个t。举个栗子：s="mononom"，t="mon"，则答案是2。思路是用两个hashmap统计s和t的字母出现频率，然后遍历t的所有字母，找到s中出现次数/t中出现次数的最小值（向下取整）

```
int countMaxOperations(string s, string t){
    unordered_map<char,int> ms;
    unordered_map<char,int> mt;
    int result = INT_MAX;
    for(auto c:s)
        ms[c]++;
    for(auto c:t)
        mt[c]++;
    for(auto tmp:mt){
        if(ms.find(tmp.first)==ms.end()) //didn't find this char
            return 0;
        else
            result = min(result,ms[tmp.first]/tmp.second);
    }

}
```

20. 密码强度

Given a string password, find the strength of that password. The strength of a password, consisting only lowercase english letters only, is calculated as the sum of the number of distinct characters present in all possible substrings of that password.
Example:- password = "good"
possible sub string and count of distinct characters are
g = 1
o = 1
o = 1
d = 1
go = 2
oo = 1
od = 2
goo = 2
ood = 2
good = 3
1+1+1+1+2+1+2+2+2+3 = 16
Example:- password: "test"
Output: 19
Example:- password: "abc"
Output: 10

```
int passwordStrength
```

```
# 密码强度
def password(s):
  ans = 0
  n = len(s)
  yuan = {"a", "e", "i", "o", "u"}
  isyuan, isfu = False, False
  for c in s:
    if c in yuan:
      isyuan = True
    else:
      isfu = True
    if isyuan and isfu:
      ans += 1
      isyuan, isfu = False, False
  return ans

print(password("thisisbeautiful"))
```

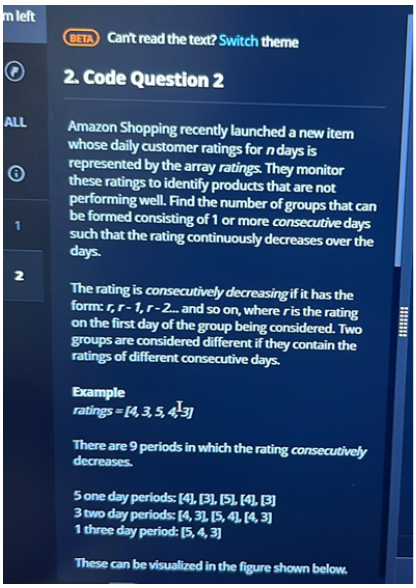
21. 最大的 processor cluster

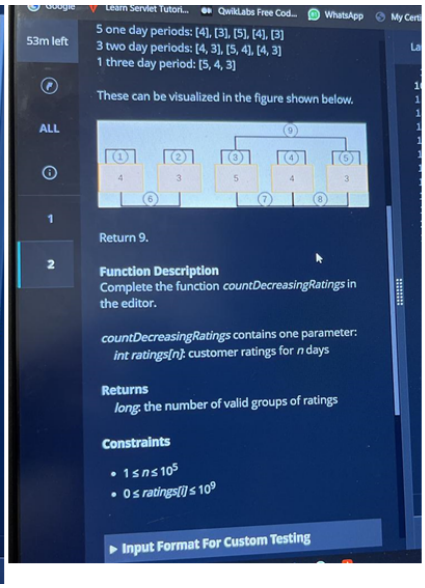
一排机器，input第一个是一个数组，代表每个机器的processing power，第二个数组代表boosting power, 第三个整数Max power。要求出满足条件最长的cluster（连续的sub array），在这个subarray里满足: max(boosting power) + sum(processing power) * k <，k是这个subarray的长度。

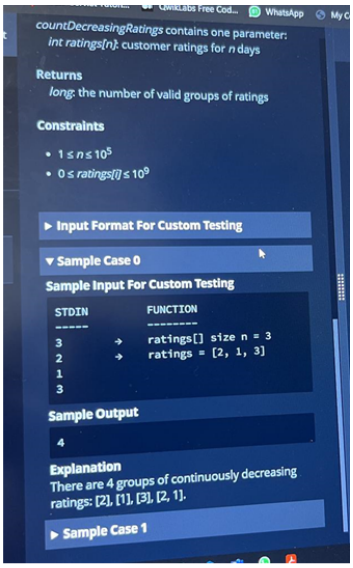
```
int findMaximumSustainableClusterSize(vector<int> processingPower, vector<int>
bootingPower, long powerMax){
    int n = processingPower.size();
    int i = 0;
    int j = 0;
```

```
    deque<vector<int>> q;
    long sum = 0;
    int ans = 0;
    while(j < n){
        sum += processingPower[j];
        while(!q.empty() && q.back()[0] <= bootingPower[j]){
            q.pop_back();
        }
        q.push_back({bootingPower[j], j});

        while(i <= j and q.front()[0] + sum*(j-i+1) > powerMax){
            if (q.front()[1] == i){
                q.pop_front();
            }
            sum -= processingPower[i];
            i++;
        }
        ans = max(ans, j-i+1);
        j++;

    }

    return ans;
}
```

22. countDecreasingRatings 连续递减的子序列个数consecutive decreasing array







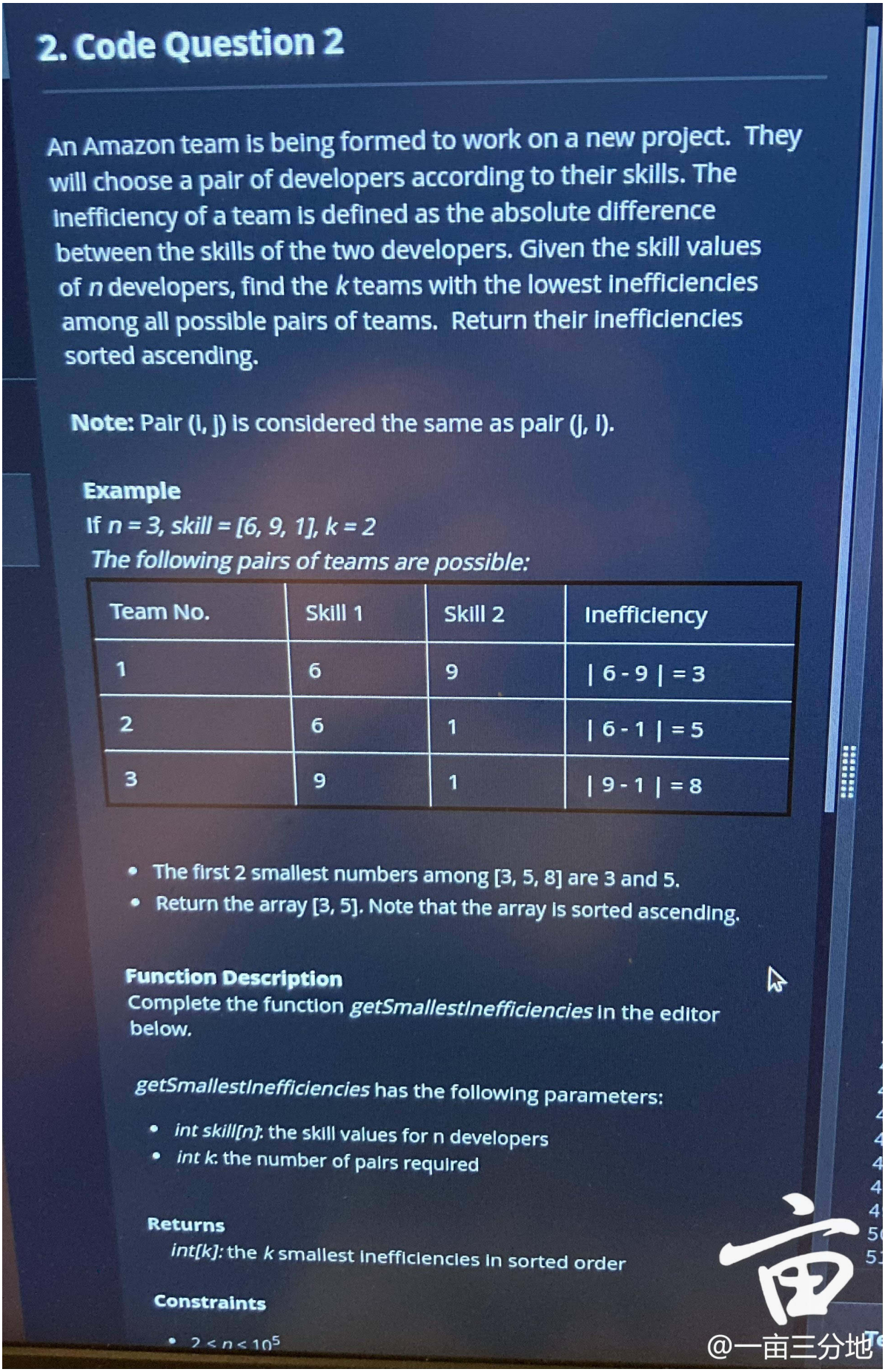
给一个integer数组arr，判断有多少个subarray是consecutive decreasing，consecutive decreasing是指这些相邻元素，后一个 = 前一个 - 1。
例子：arr = [4, 3, 2, 5, 9, 8]
符合条件的subarray有 [4], [4, 3], [4, 3, 2], [3], [3, 2], [2], [5], [9], [9, 8], [8]，所以答案是10。
解法：先把arr分成consecutive decreasing的subarray，记录每个subarray的长度，长度为n的subarray有 n * (n + 1) / 2种case。
上面的例子，先分成[4, 3, 2], [5], [9, 8]，记录所有的长度[3, 1, 2]，再对每个长度n求n * (n + 1) / 2，最后相加。

```
int countDecreasingRatings(vector<int> ratings){
    int left=0;
    int right=1;
    int res=0;
    while(right<ratings.size()){
        if(ratings[right-1]==ratings[right]+1)
            res += right-start;
        else
            left=right;
        right++;
    }
    res += ratings.size(); //add all single-element vectors
}
```

23. FindMinHealth (LC 2214)

```
class Solution {
public:
    long long minimumHealth(vector<int>& damage, int armor) {
        int maxDamage=INT_MIN;
        long long totalDamage=0;
        for(int i=0;i<damage.size();i++){
            maxDamage=max(maxDamage,damage[i]);
            totalDamage += damage[i];
        }
        if(maxDamage>armor)
            return totalDamage+1-armor;
        else
            return totalDamage+1-maxDamage;
    }
};
```

24. 变种lc719, smallest distance pair; k smallest inefficiencies



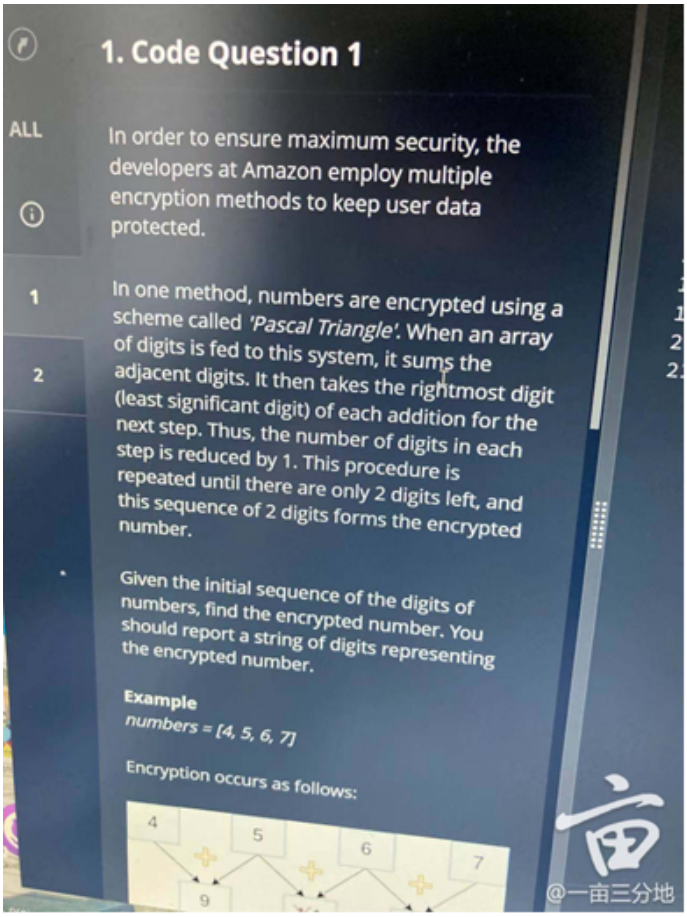
Priority queue

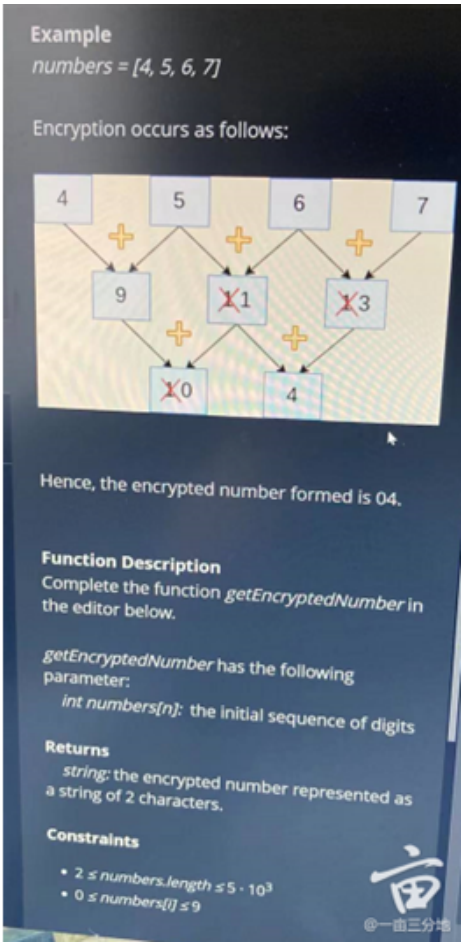
```
class Solution {
public:
    int smallestDistancePair(vector<int>& nums, int k) {
```

```
        vector<int> nums = {6,9,1};
        int k=2;
        sort(nums.begin(),nums.end());
        priority_queue<vector<int>,vector<vector<int>>,greater<vector<int>>> pq;
        for(int i=0;i<nums.size()-1;i++){
            pq.push({nums[i+1]-nums[i],i,i+1});
        }
        vector<int> tmp;
        vector<int> res;
        while(k>0){
            tmp=pq.top();
            pq.pop();
            res.push_back(tmp[0]);
            if(res[2]+1<nums.size()){
                 pq.push({nums[res[2]+1]-nums[res[1]],res[1],res[2]+1});
            }
            k--;
        }
        for(auto c:res)
            cout << c<< endl;
        return res;
    }
};
```

```
vector<int> lowestKInefficient(vector<int> developers, int k){
    int n = developers.size() ;
    priority_queue<int> maxH ;
    for(int i = 0 ; i < n-1; i++){
        for(int j = i+1; j < n; j++){
            maxH.push(abs(developers[j] - developers[i])) ;
            if(maxH.size() > k){
                maxH.pop() ;
            }
        }
    }

    vector<int> ans ;
    while(!maxH.empty()){
        ans.push_back(maxH.top()) ;
        maxH.pop() ;
    }
    reverse(ans.begin(), ans.end()) ;
    return ans;
}
```

25. pascal triangle (变种LC2221)






```
vector<int> triangularSum(vector<int>& nums) {
    if(nums.size() == 2)
        return nums;
    for(int i = 0 ; i < nums.size()-1; ++i)
        nums[i] = (nums[i] + nums[i+1])%10;
        nums.pop_back();
    return triangularSum(nums);
}
```

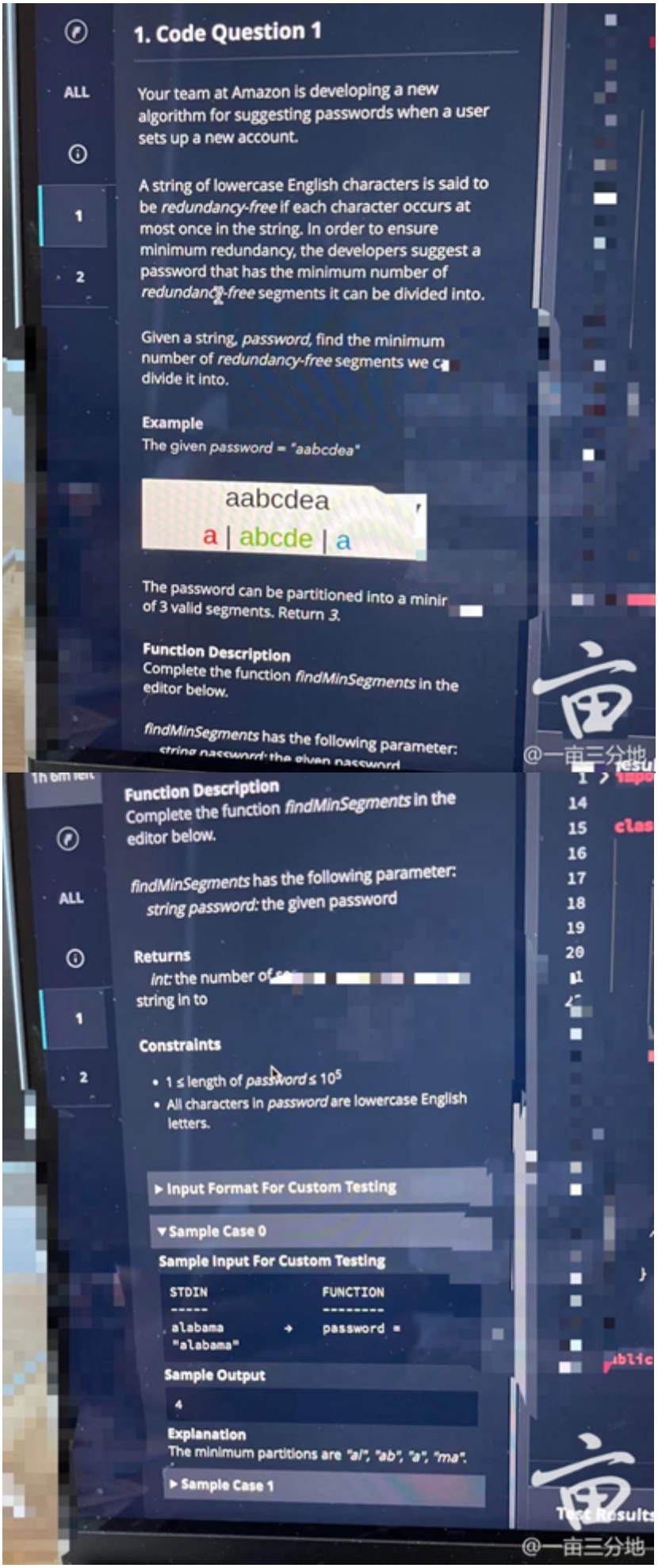
26. 缺失数字

给了一个List, 和一个Int k。要你找出满足k - List.size( )个大小的最小缺失数字和。
Example: [1, 3, 5, 7, 10] k = 7;
Process: miss nums: 2, 4, 6, 8, 9;
res: 我们需要 7 - 5 = 2 个缺失数字， 要想和最小，那就从头开始选，那就是2 + 4 = 6，
所以最后return 6

```
class Solution {
public:
long long minimalKSum(vector<int>& nums, int k) {
    nums.push_back(0);
    sort(nums.begin(), nums.end());
    int maxval;
    long long ret = 0;
    for(int i=0; i<nums.size()-1; i++) {
        int tmp = nums[i+1] - nums[i];
        if(tmp > 0) {
            ret -= nums[i];
```

```
            tmp--;
            if(tmp >= k) {
                maxval = nums[i] + k;
                k = 0;
                break;
            }
            else
                k -= tmp;
        }
    }
    if(k > 0) {
        ret -= nums.back();
        maxval = nums.back() + k;
    }
    ret += (((long long)(maxval)+1)*maxval)>>1;
    return ret;
    }
};
```

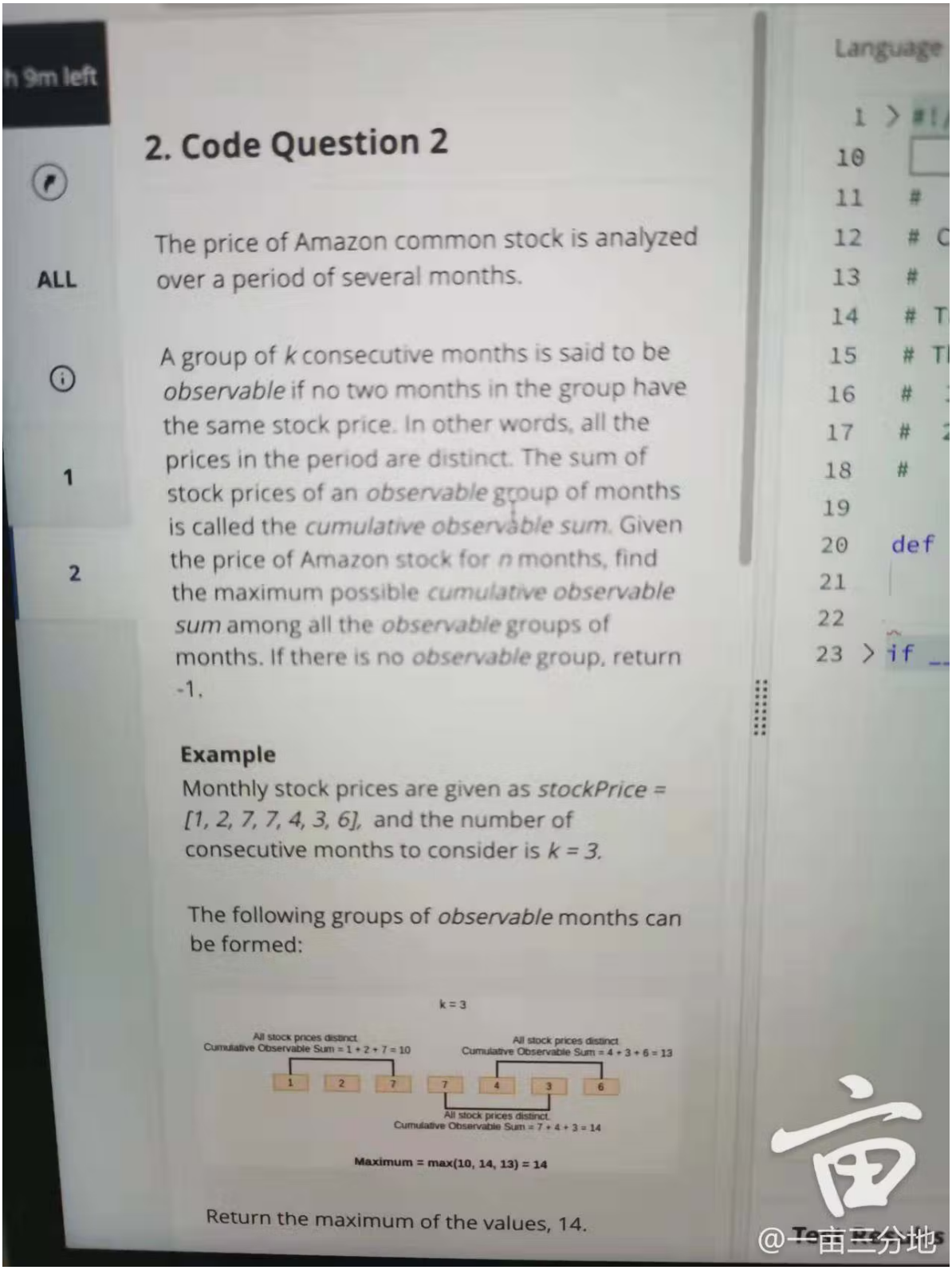
27. 分割字符串,每个字符串无重复 minimum redundancy-free segments




```
int main() {
    //string str="aabcdea";
```

```
    //string str="alabama";
    string str="aaaaaa";
    int left=0;
    int right=0;
    int result=1;
    unordered_set<char> s;
    while(right<str.length()){
        if(s.find(str[right])!=s.end()){
            left=right;
            s.clear();
            result++;
        }
        s.insert(str[right]);
        right++;
    }
    cout<<result;
    return 0;
}
//"aabcdea": 3
//"alabama":4
//"aaaaaa":6
```

28 Max stock price 给定k值，给连续k月并且k个值各不一样的区间求和，找到最大和。

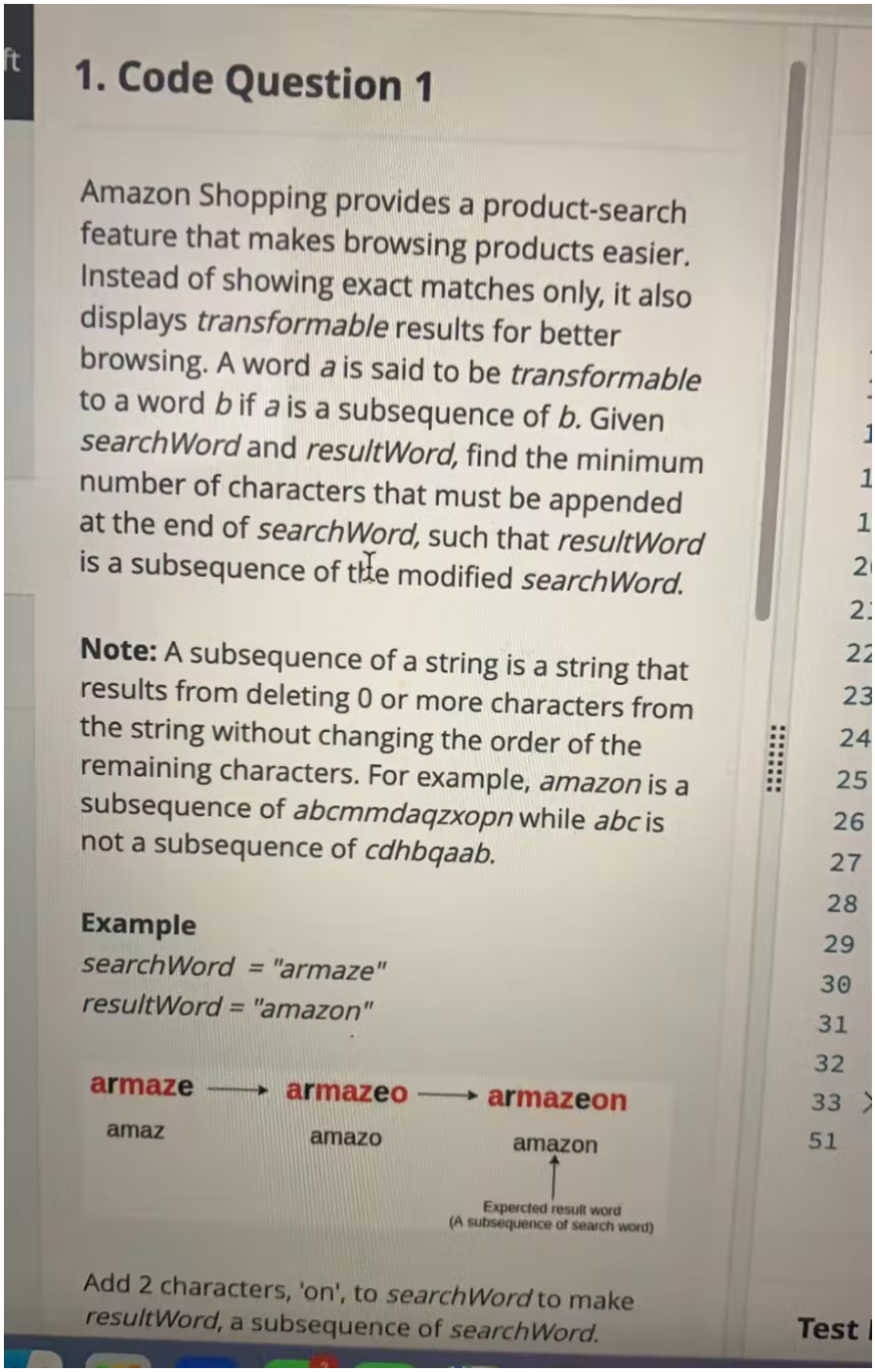



```cpp
long maxSumOfStock(vector<int> stockPrice, int k){
    long result = 0;
    int left=0;
    int right=0;
    unordered_set<int> s;
    while(right<stockPrice.size()){
        while(s.find(stockPrice[right])!=s.end() || right-left+1>k){
            s.erase(stockPrice[left++]);
        }
        s.insert(stockPrice[right]);
        if(right-left+1==k){
            long tmp=0;
```

```
            for(auto i:s)
                tmp += i;
            result = max(result,tmp);
        }
        right++;
    }
    if(result==0)
        return -1;
    return result;
}
```

29 SearchWord & resultWord searchword需要添加多少个字符

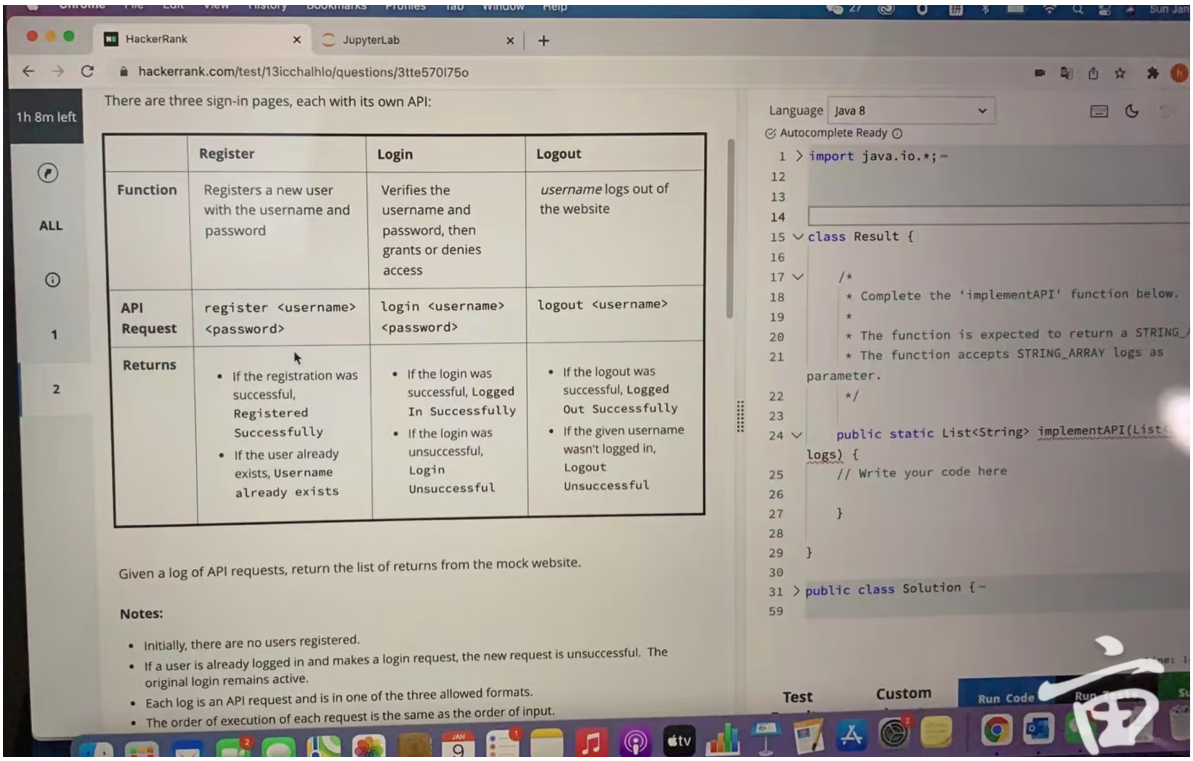
一个searchWord和一个resultWord，问最少给searchWord末尾添加几个字符，可以让resultWord成为它的一个subsequence。举个栗子：search给Word="armaze"，resultWord="amazon"，则应该返回2（添加on）。思路是两个指针p1，p2分别遍历两个字符串，如果指向的字符相同，则将双指针同时向后移动一位，反之只移动指向searchWord的指针，直到任意指针到达末尾。返回resultString的长度与resultString指针位置的差



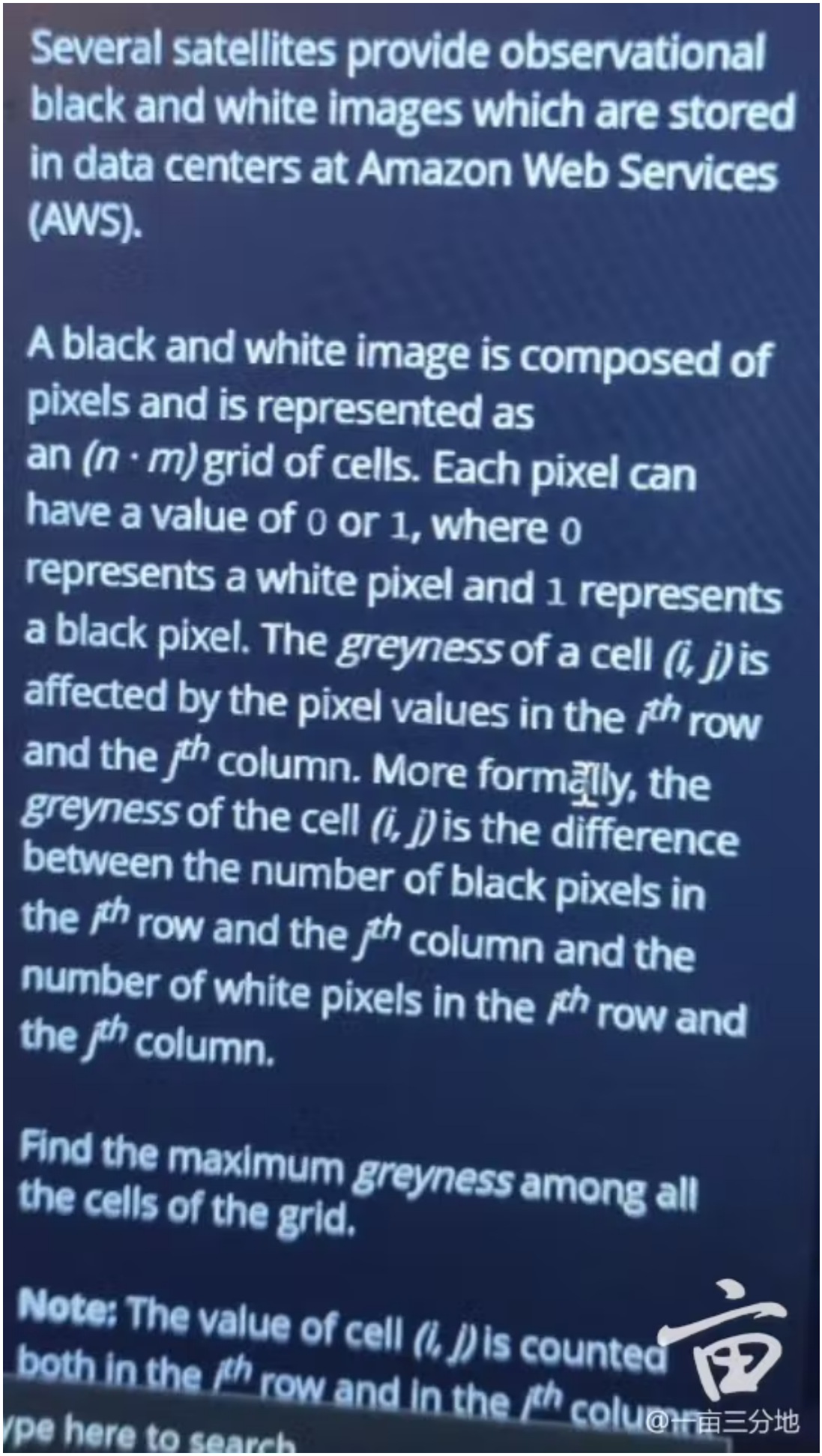

```
int searchWordToResultWord(string searchWord, string resultWord){
    int ps=0;
    int pr=0;
    while(ps<searchWord.length() && pr<resultWord.length()){
        if(searchWord[ps]==resultWord[pr])
            pr++;
        ps++;
    }
    return resultWord.length()-pr;
}
```

30. User System Design 用户登陆API设计

写一个简单的api，有三个功能 register，log in，log out。register的时候要输入name和password，如果这个用户已经register过了要返回username already exists，没有的话返回registered successfully；log in时也要name和password，如果该name并没有register或者已经logged in，或者password错误，要返回log in unsuccessful，如果都满足就返回logged in successfully；最后是log out，也是很直观的逻辑，正常的话返回成功，没有register或者没有log in的name要返回log out失败。这道题不是要你去写class，而是一个函数，通过输入a list of strings然后你自己去parse出以上三个指令。Hashmap储存username和password



31 MaxGreyness 灰度题



给一个2D array，array[i][j]表示在(i, j)点处的灰度值，0为白1为黑，(i, j)处的灰度值为第i行所有点灰度值 + 第j列所有点灰度值之和 （黑+1，白-1），求array中灰度值最大的点的灰度值为多少，两层for循环分别求出colsGrey[]和 rowsGrey[]然后再遍历一遍求max即可

```
int maxGreyness(vector<string> grid){
    vector<int> sumRow(grid.size(),0);
    vector<int> sumCol(grid[0].size(),0);
    for(int i=0;i<grid.size();i++){
        for(int j=0;j<grid[0].size();j++){
            sumRow[i] += 2*(grid[i][j]=='1')-1;
            sumCol[j] += 2*(grid[i][j]=='1')-1;
        }
    }
    int maxRow=INT_MIN;
    int maxCol=INT_MIN;
    for(auto i:sumRow)
        maxRow=max(maxRow,i);
    for(auto i:sumCol)
        maxCol=max(maxCol,i);
    return maxRow+maxCol;
}
```

32 ShipmentImbalance（leetcode 2104）子数组Max/Min求差值，再加和

求array的所有subArray的 max - min的和总和.

O(n^2):

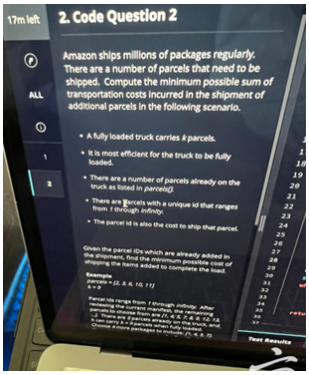
```
    long long subArrayRanges(vector<int>& A) {
        long long res = 0;
        for (int i = 0; i < A.size(); i++) {
            int ma = A[i], mi = A[i];
            for (int j = i; j < A.size(); j++) {
                ma = max(ma, A[j]);
                mi = min(mi, A[j]);
                res += ma - mi;
            }
        }
        return res;
    }
```

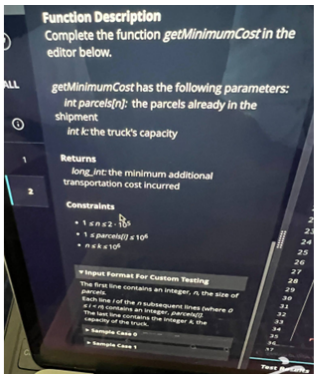
O(n):

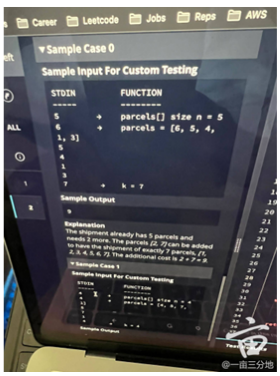
```
    long long subArrayRanges(vector<int>& A) {
        long res = 0, n = A.size(), j, k;
        stack<int> s;
        for (int i = 0; i <= n; ++i) {
            while (!s.empty() && A[s.top()] > (i == n ? -2e9 : A[i])) {
                j = s.top(), s.pop();
                k = s.empty() ? -1 : s.top();
                res -= (long)A[j] * (i - j) * (j - k);
            }
            s.push(i);
        }
        s = stack<int>();
        for (int i = 0; i <= n; ++i) {
            while (!s.empty() && A[s.top()] < (i == n ? 2e9 : A[i])) {
```

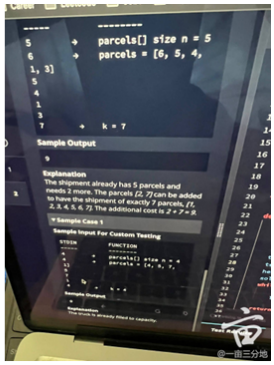
```
                j = s.top(), s.pop();
                k = s.empty() ? -1 : s.top();
                res += (long)A[j] * (i - j) * (j - k);
            }
            s.push(i);
        }
        return res;
    }
```

33 MinShippingCost 包裹运输(LC2195)





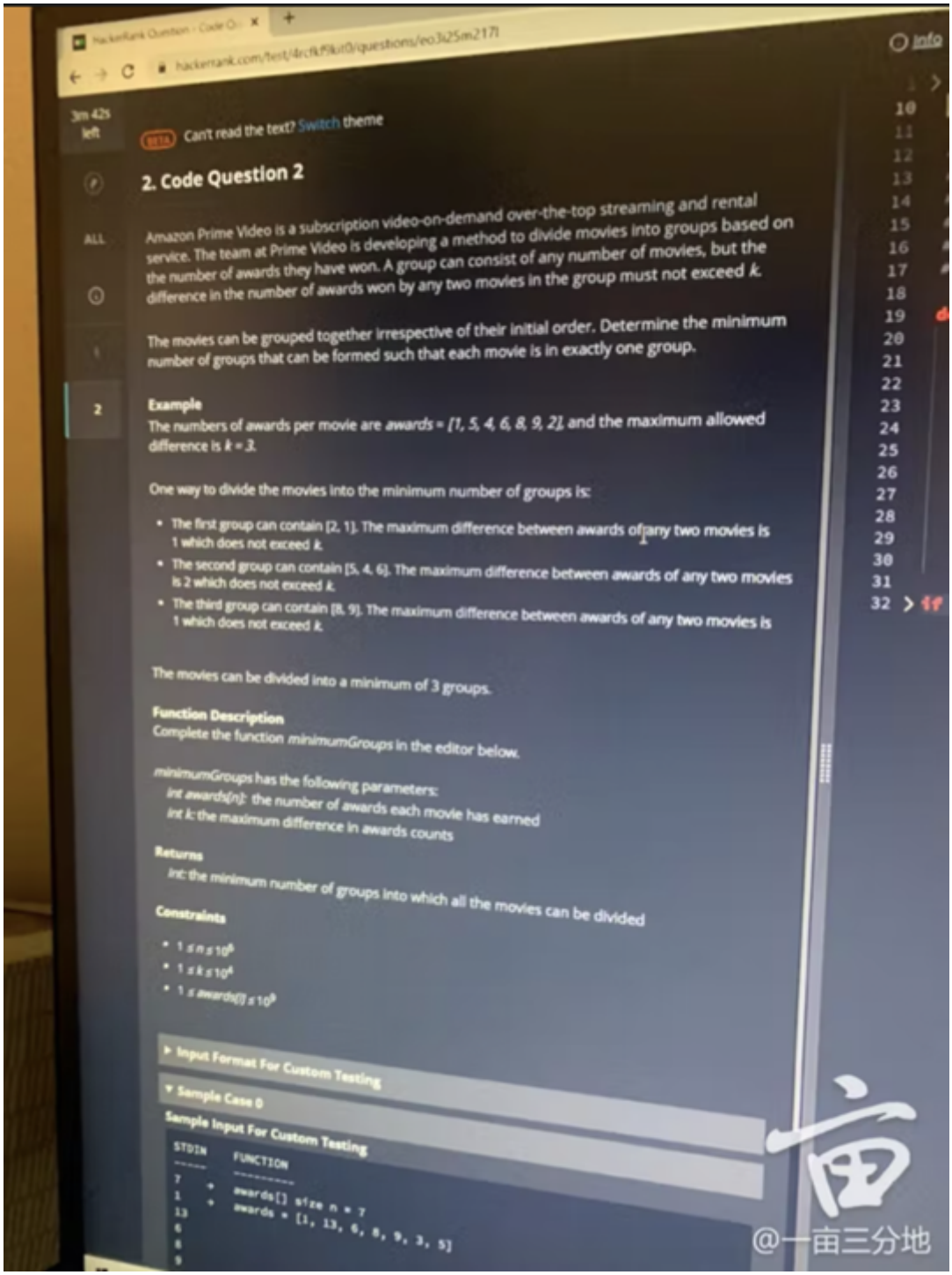





```
class Solution {
public:
    long long minimalKSum(vector<int>& nums, int k) {
        int count=k;
        nums.push_back(0);
        nums.push_back(INT_MAX);
        sort(nums.begin(),nums.end());
        long long res=0;
        int startNum=0;
        for(int num:nums){
            int diff = num-startNum;
            if(diff>1 && count >0){
                long range = min(diff-1,count);
                res += (startNum+1+startNum+range)*range/2;
                count -= range;
            }
            startNum=num;
            if(count<=0)
                break;
        }
        return res;
    }
};
```

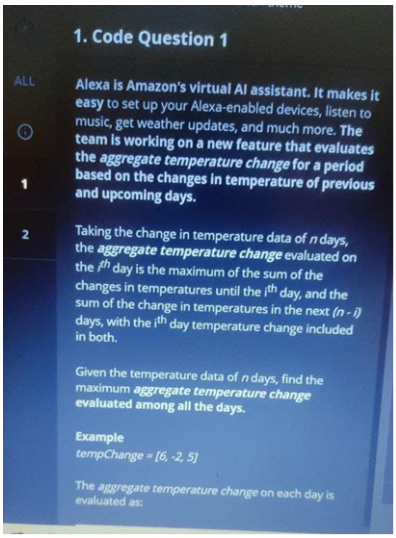
34. PrimeMovieAward

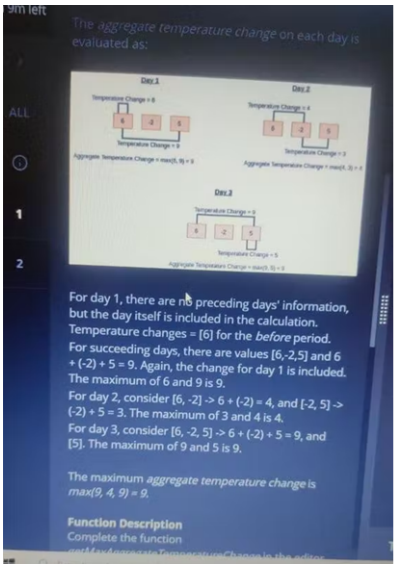
awards 分组，每组max-min不超过k， 就直接sort一遍，然后贪心的把最大能加到当前组的加进来，加不进来的count++, 换个 新组

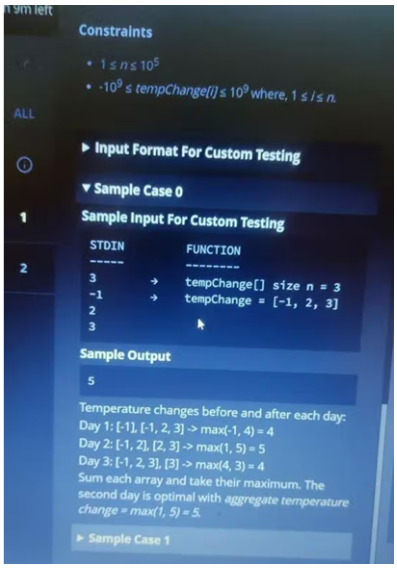



```cpp
int minGroups(vector<int> awards,int k){
    int left=0;int right=0;
    sort(awards.begin(),awards.end());
    int res=0;
    while(right<awards.size()){
        if(awards[right]-awards[left]>k){
            left=right;
            res++;
        }
        right++;
    }
    return res+1;
}
```

35 MaxAggregateTemperatureChange 最大浮动温度








```
long getMaxAggreTempChange(vector<int> changes){
    int n=changes.size();
    vector<long> prefixSum(n);
    prefixSum[0]=changes[0];
    for(int i=1;i<n;i++)
        prefixSum[i]=prefixSum[i-1]+changes[i];
    long res=max(prefixSum[i],prefixSum[n-1]);
    for(int i=1;i<n;i++){
        long tmp=max(prefixSum[i],prefixSum[n-1]-prefixSum[i-1]);
        res=max(res,tmp);
    }
    return res;
}
```

36. 股价findEarliest month

2. Code Question 2

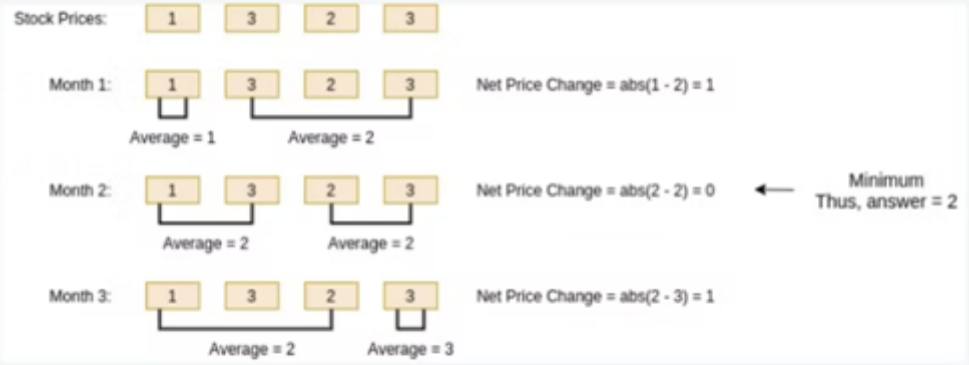
The interns at Amazon were asked to review the company's stock value over a period. Given the stock prices of *n* months, the *net price change* for the i^{th} month is defined as the absolute difference between the average of stock prices for the first *i* months and for the remaining *(n - i)* months where *1 ≤ i < n*. Note that these averages are rounded down to an integer.

Given an array of stock prices, find the month at which the *net price change* is minimum. If there are several such months, return the earliest month.

Note: The average of a set of integers here is defined as the sum of integers divided by the number of integers, rounded down to the nearest integer. For example, the average of [1, 2, 3, 4] is the floor of (1 + 2 + 3 + 4) / 4 = 2.5 and the floor of 2.5 is 2.

Example

stockPrice = [1, 3, 2, 3]



The minimum *net price change* is 0, and it occurs in the 2^{nd} month. Return 2.

Function Description

Complete the function *findEarliestMonth* the editor below.

findEarliestMonth has the following parameter:

int stockPrice[n]: the stock prices

Returns

int: the earliest month in which the *net price change* is minimum

Constraints

- $2 \le n \le 10^5$
- $1 \le stockPrice[i] \le 10^9$

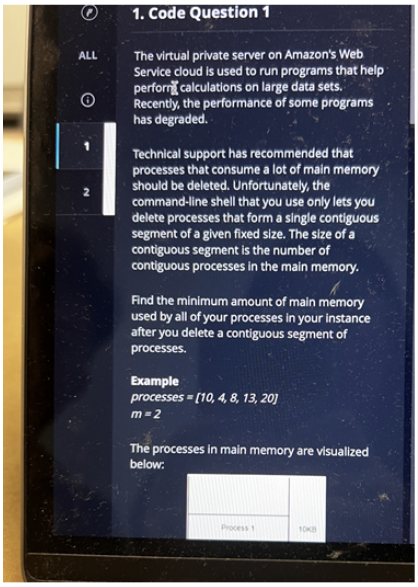
prefixSum 记录从开始到现在的总和。

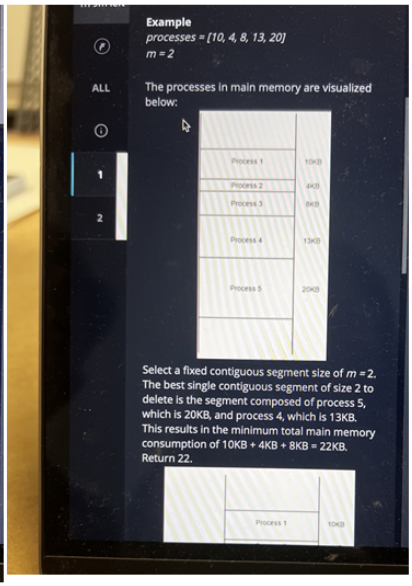
股票stock[]切一刀，切在哪里时左半边Avg和右半边Avg的差值最小，注意数组0-index，返回月份要+1。

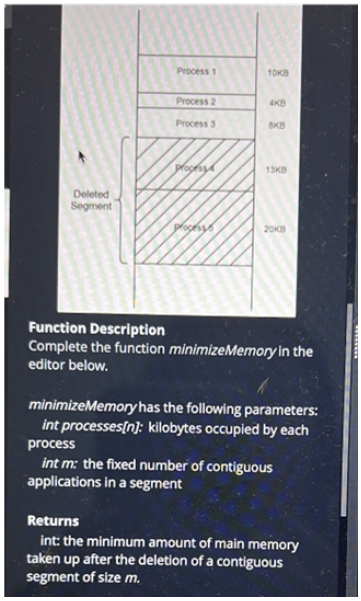
```
int earliestMonth(vector<int> stockPrice){
      int n=stockPrice.size();
        vector<long> prefix(n);
        prefix[0] = stockPrice[0];
        for(int i=1;i<n;i++)
            prefix[i]=prefix[i-1]+stockPrice[i];
        long minDiff=INT_MAX;
        int res=0;
        for(int i=0;i<n-1;i++){
            long left = prefix[i]/long(i+1);
            long right = (prefix[n-1]-prefix[i])/long(n-i-1);
            long diff=abs(left-right);
```

```
            if(diff<minDiff){
                res=i;
                minDiff=diff;
            }
        }
        if(prefix[n-1]/n<minDiff)
            res = n-1;
        return res;
}
```

37. 给一组数组, 删掉k个连续元素, 问剩下最小的和是多少?



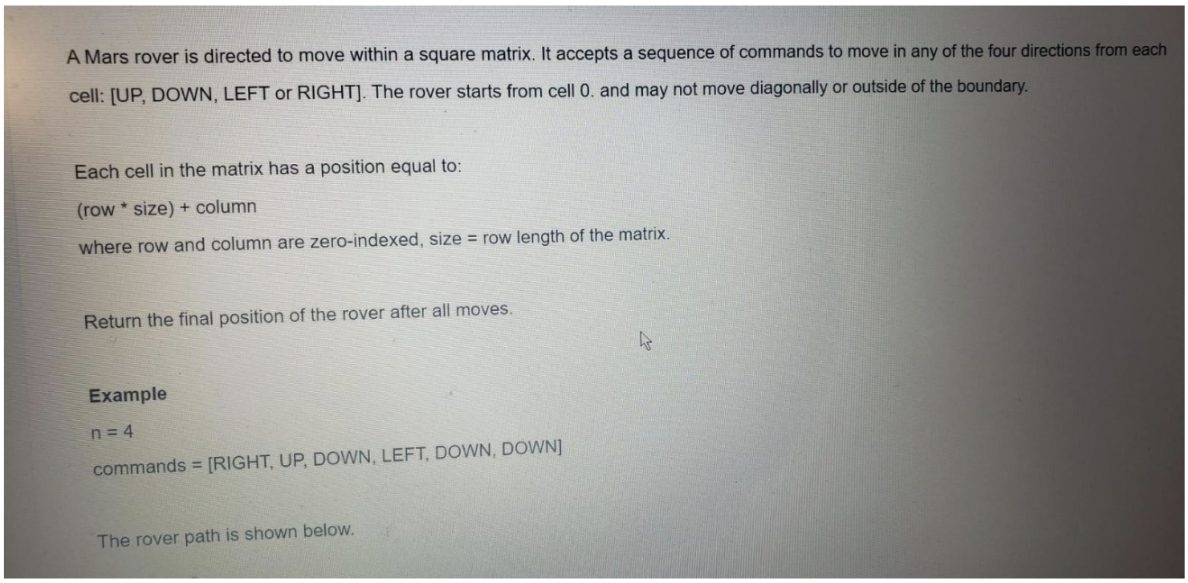


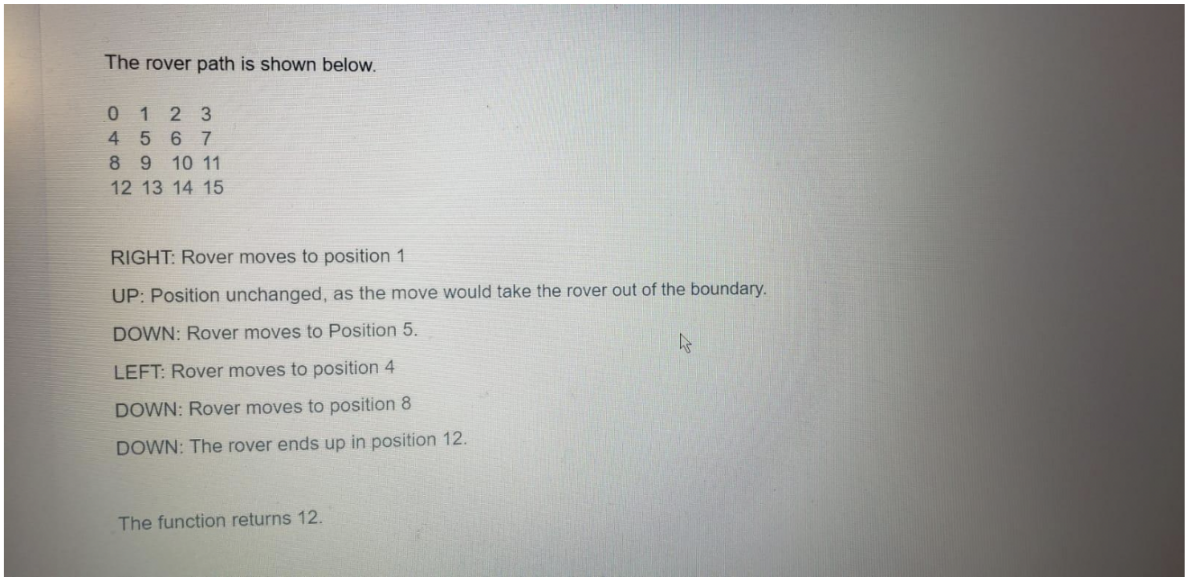


sliding window, 求长度为K的subarray的和的最大值，然后用总和减去

```
int getMinSum(vector<int> nums, int k){
    int left=0;
    int right=0;
    int maxSum = 0;
    int currentSum = 0;
    int totalSum = 0;
    while(right<nums.size()){
        currentSum += nums[right];
        totalSum += nums[right];
        if(right>=k-1){
            maxSum=max(currentSum,maxSum);
            currentSum -= nums[left++];
        }
        right++;
    }
    return totalSum-maxSum;
}
```

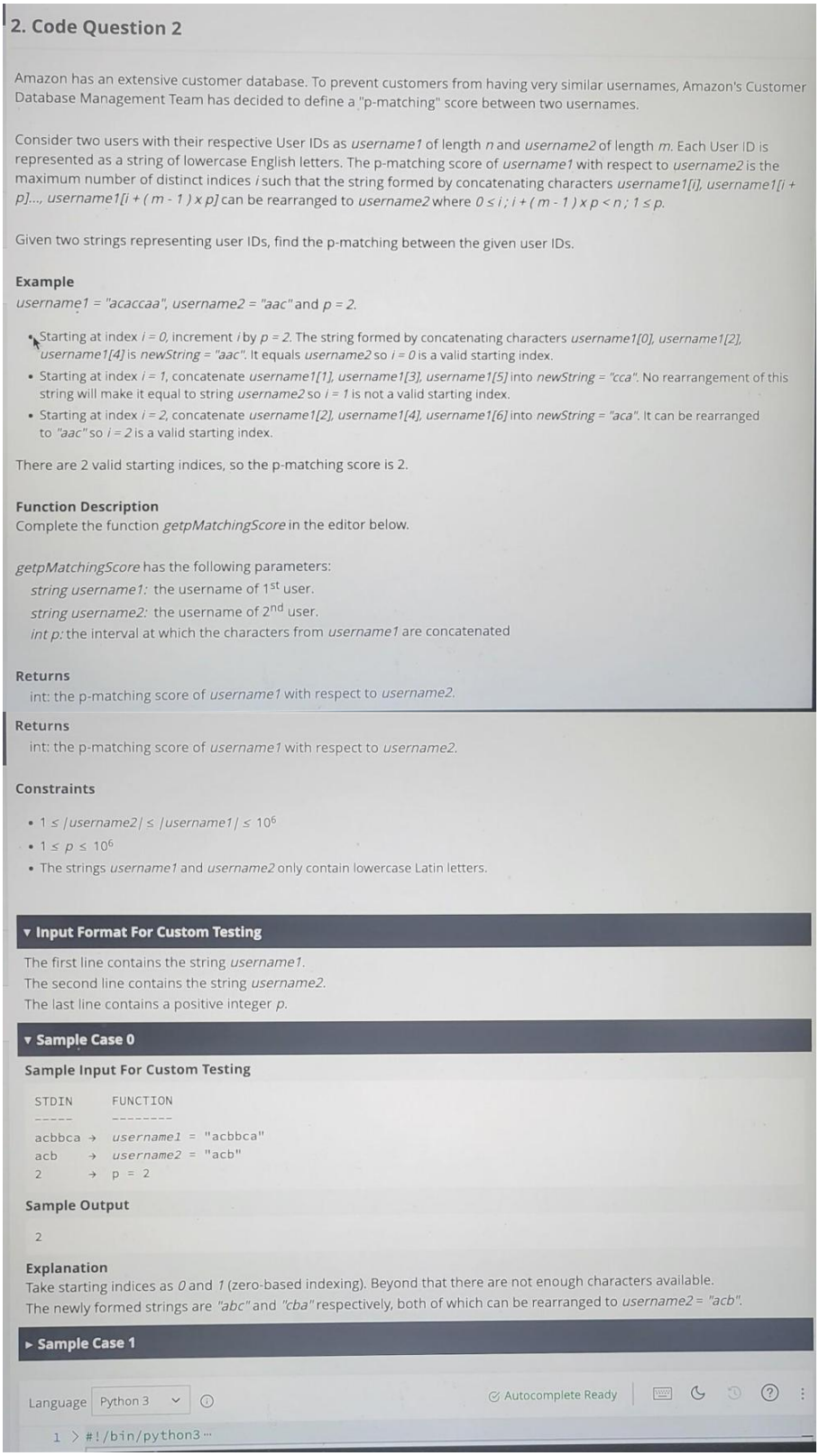
38. River Control






```
int riverControl(vector<string> int n)
```

39. P-matching score



2. Code Question 2

Amazon has an extensive customer database. To prevent customers from having very similar usernames, Amazon's Customer Database Management Team has decided to define a "p-matching" score between two usernames.

Consider two users with their respective User IDs as *username1* of length *n* and *username2* of length *m*. Each User ID is represented as a string of lowercase English letters. The p-matching score of *username1* with respect to *username2* is the maximum number of distinct indices *i* such that the string formed by concatenating characters *username1[i], username1[i + p]..., username1[i + (m - 1) x p]* can be rearranged to *username2* where $0 \le i ; i + (m - 1) x p < n ; 1 \le p$.

Given two strings representing user IDs, find the p-matching between the given user IDs.

Example

username1 = "acaccaa", username2 = "aac" and *p = 2*.

- Starting at index *i = 0*, increment *i* by *p = 2*. The string formed by concatenating characters *username1[0], username1[2], username1[4]* is *newString = "aac"*. It equals *username2* so *i = 0* is a valid starting index.
- Starting at index *i = 1*, concatenate *username1[1], username1[3], username1[5]* into *newString = "cca"*. No rearrangement of this string will make it equal to string *username2* so *i = 1* is not a valid starting index.
- Starting at index *i = 2*, concatenate *username1[2], username1[4], username1[6]* into *newString = "aca"*. It can be rearranged to *"aac"* so *i = 2* is a valid starting index.

There are 2 valid starting indices, so the p-matching score is 2.

Function Description

Complete the function *getpMatchingScore* in the editor below.

getpMatchingScore has the following parameters:

string username1: the username of 1st user.
string username2: the username of 2nd user.
int p: the interval at which the characters from *username1* are concatenated

Returns

int: the p-matching score of *username1* with respect to *username2*.

Returns

int: the p-matching score of *username1* with respect to *username2*.

Constraints

- $1 \le |username2| \le |username1| \le 10^6$
- $1 \le p \le 10^6$
- The strings *username1* and *username2* only contain lowercase Latin letters.

▾ Input Format For Custom Testing

The first line contains the string *username1*.
The second line contains the string *username2*.
The last line contains a positive integer *p*.

▾ Sample Case 0

Sample Input For Custom Testing

```
STDIN       FUNCTION
-----       --------
acbbca →    username1 = "acbbca"
acb    →    username2 = "acb"
2      →    p = 2
```

Sample Output

```
2
```

Explanation

Take starting indices as *0* and *1* (zero-based indexing). Beyond that there are not enough characters available.
The newly formed strings are *"abc"* and *"cba"* respectively, both of which can be rearranged to *username2 = "acb"*.

▸ Sample Case 1

Language Python 3 — Autocomplete Ready

```
1 > #!/bin/python3 ···
```

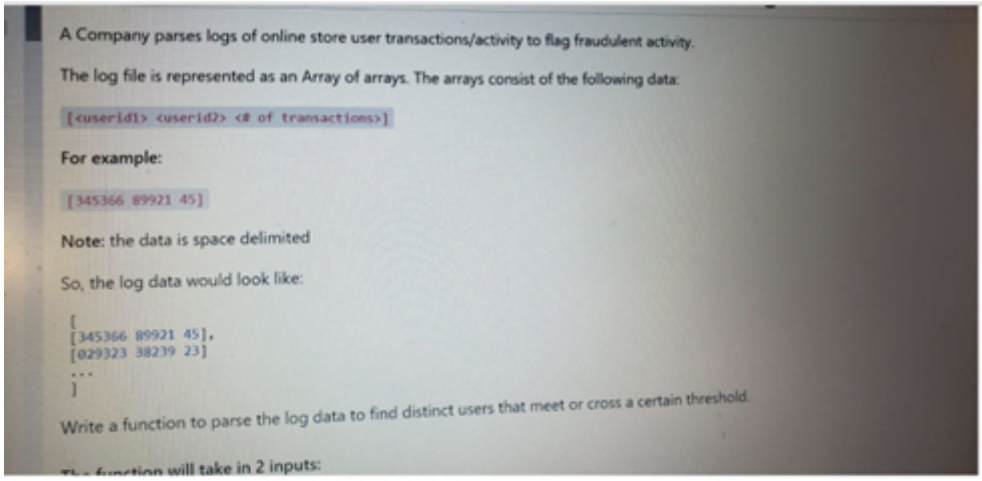
```cpp
int pMatchingScore(string s1,string s2, int p){
    int result = 0;
    int start=0;
    while(start<s1.length()-(s2.length()-1)*p){
        int ps1=start;
        int ps2=0;
        result += valid(s1,s2,ps1,p);
        start++;
    }
}
bool valid(string &s1,string &s2, int ps1,int p){
    ps2=0;
```

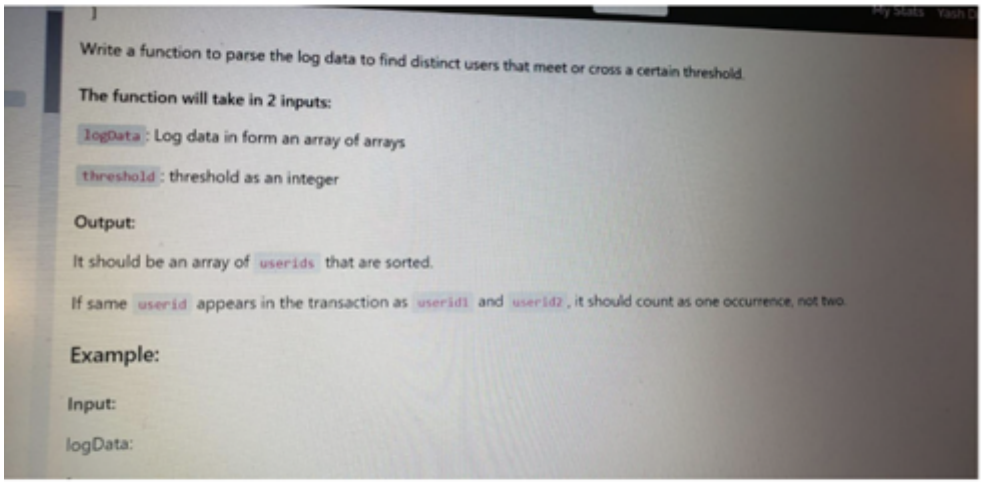
```
    while(ps<s2.length() && ps1<s1.length()){
        if(s1[ps1]==s2[ps2]){
            ps2++;
            ps1 += p;
        }
        else return false;
    }
    return ps==s2.length();
}
```

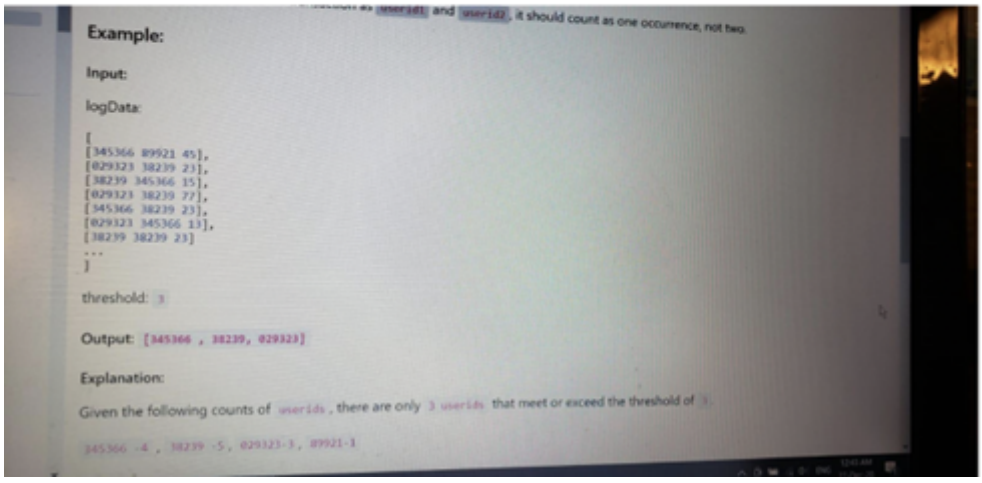
要理解为什么能用slicing window， 你得先发现遍历原字符串是存在字符串复用的可能性的。. 1point 3 acres
传统的sliding window每次往右边移动一格就能得到一个需要evaluate的substring，但这个题因为有个长度为k的间隔 你每次要往右移动k个坐标，才能得到下一个要evaluate的substring 然后你会发现这个substring跟你刚才evaluate的那个只有首尾两个字符之差。比如原字符串string1是abcdefgh， string2是ace，k = 2，那就相当于对aceg做一次定长为3的sliding window 对bdfh也做一次，总共分成了k个需要进行sliding window的substring

```
```

40. Processing Logs 交易日志处理 返回超过阈值的用户ID



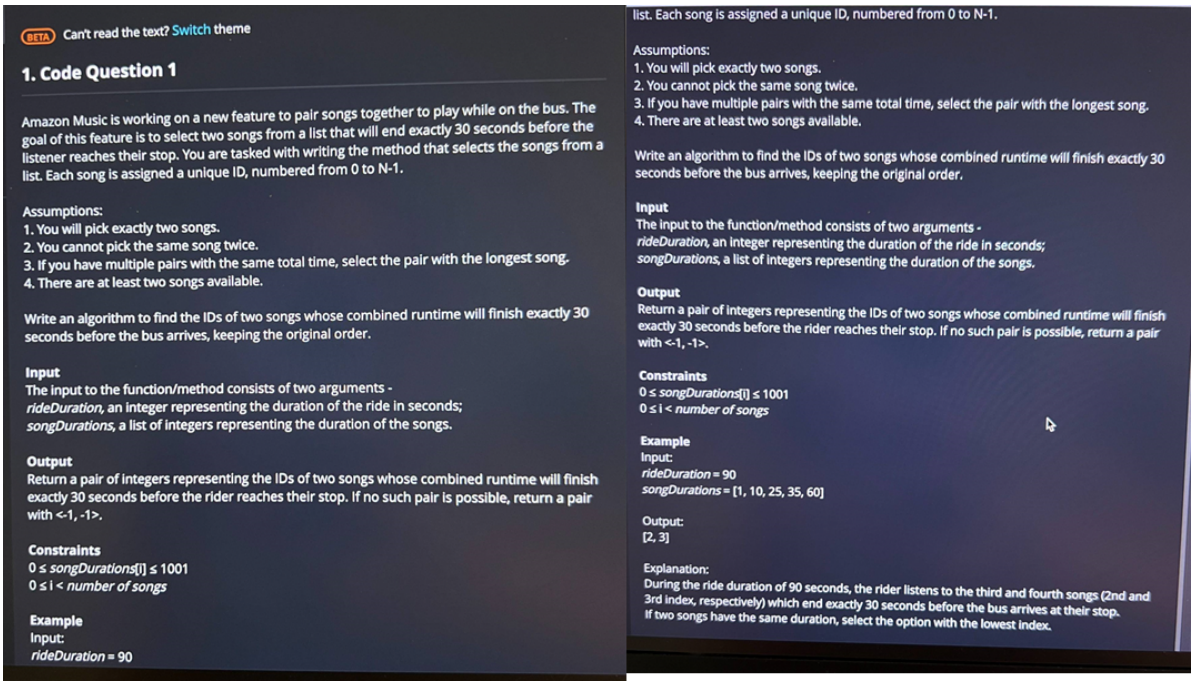





```
vector<string> processLog(vector<string> log, int k){
    vector<string> result;
    unordered_map<string,int> m;
    for(auto s: log){
        string s1;
        string s2;
        int nspace=0;
        for(int i=0;i<s.length();i++){
            if(s[i]==' ')
                nspace++;
            else if(nspace==0)
                    s1 += s[i];
                if(nspace==1)
                    s2 += s[i];
                else
                    break;
```

```
        }
        if(s1==s2)
            m[s1]++;
        else{
            m[s1]++;
            m[s2]++;
        }
    }
    for(auto tmp:m){
        if(m.second>k)
            result.push_back(tmp.first);
    }
    return result;
}
```

41 Pick song with total time = rideduration - 30



BETA Can't read the text? Switch theme

1. Code Question 1

Amazon Music is working on a new feature to pair songs together to play while on the bus. The goal of this feature is to select two songs from a list that will end exactly 30 seconds before the listener reaches their stop. You are tasked with writing the method that selects the songs from a list. Each song is assigned a unique ID, numbered from 0 to N-1.

Assumptions:
1. You will pick exactly two songs.
2. You cannot pick the same song twice.
3. If you have multiple pairs with the same total time, select the pair with the longest song.
4. There are at least two songs available.

Write an algorithm to find the IDs of two songs whose combined runtime will finish exactly 30 seconds before the bus arrives, keeping the original order.

Input
The input to the function/method consists of two arguments -
rideDuration, an integer representing the duration of the ride in seconds;
songDurations, a list of integers representing the duration of the songs.

Output
Return a pair of integers representing the IDs of two songs whose combined runtime will finish exactly 30 seconds before the rider reaches their stop. If no such pair is possible, return a pair with <-1, -1>.

Constraints
$0 \le songDurations[i] \le 1001$
$0 \le i < number\ of\ songs$

Example
Input:
rideDuration = 90

list. Each song is assigned a unique ID, numbered from 0 to N-1.

Assumptions:
1. You will pick exactly two songs.
2. You cannot pick the same song twice.
3. If you have multiple pairs with the same total time, select the pair with the longest song.
4. There are at least two songs available.

Write an algorithm to find the IDs of two songs whose combined runtime will finish exactly 30 seconds before the bus arrives, keeping the original order.

Input
The input to the function/method consists of two arguments -
rideDuration, an integer representing the duration of the ride in seconds;
songDurations, a list of integers representing the duration of the songs.

Output
Return a pair of integers representing the IDs of two songs whose combined runtime will finish exactly 30 seconds before the rider reaches their stop. If no such pair is possible, return a pair with <-1, -1>.

Constraints
$0 \le songDurations[i] \le 1001$
$0 \le i < number\ of\ songs$

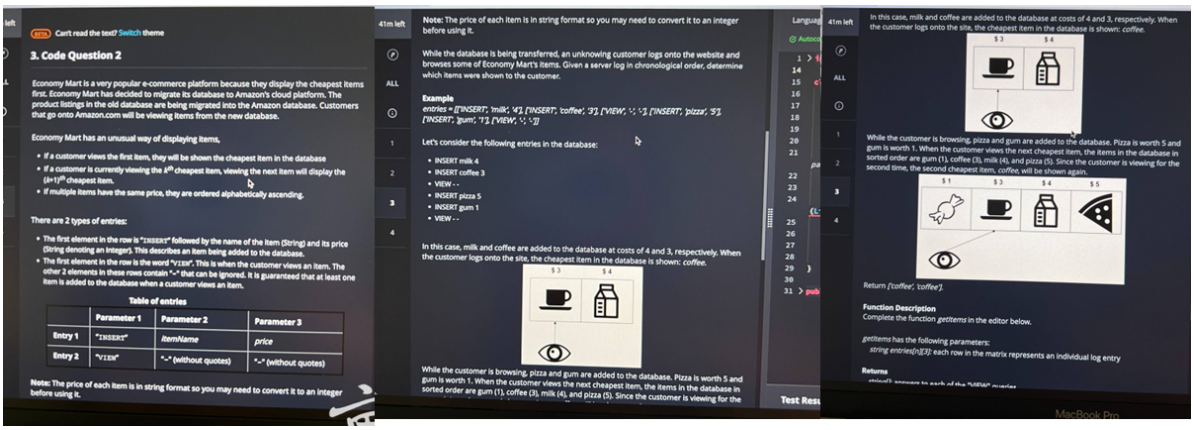
Example
Input:
rideDuration = 90
songDurations = [1, 10, 25, 35, 60]

Output:
[2, 3]

Explanation:
During the ride duration of 90 seconds, the rider listens to the third and fourth songs (2nd and 3rd index, respectively) which end exactly 30 seconds before the bus arrives at their stop.
If two songs have the same duration, select the option with the lowest index.

```
vector<int> findSong(int rideDuration, vector<int> songDuration){
    unordered_map<int,int> s;
    int target = rideDuration - 30;
    vector<int> result(2,-1);
    for(int i=0;i<songDuration.size();i++){
        if(s.find(target-songDuration[i])!=s.end()){
            if(abs([target-2*songDuration])>abs(songDuration[result[0]]-
songDuration[result[1]])){
                result[0]=s[target-songDuration[i]];
                reuslt[1]=i;
            }
        }
        s[songDuration[i]]=i;
    }
    return result;
}
```

42 .View items from the economy mart platform




```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

int main()
{
    vector<vector<string>> entries = {{"INSERT","milk","4"},
{"INSERT","coffee","3"},\
                                      {"VIEW","-","-"},{"INSERT","pizza","5"},
{"INSERT","apple","3"},{"INSERT","gum","1"},\
                                      {"VIEW","-","-"}};
    vector<pair<string,string>> v;
    int k=0;
    for(auto e:entries){
        if(e[0]=="INSERT")
            v.push_back(pair<string,string>(e[2],e[1]));
        else{
            sort(v.begin(),v.end());
            cout<<v[k].second<<endl;
            k++;
        }
    }
    return 0;
}
//coffee
//apple
```

43. findMaximumMaximaCount

Amazon has a string of categories of items purchased by a particular customer, each represented as a lowercase English Letter. To analyze customer behavior we define a metric called the MaximaCount of a category c is the maximum among all categorys is defined as the number of indices i such taht the frequency of char is maximum in the prefix of the string up to the index i given the string categories, find the maximum MaximaCount amoung all the categories

Given categories = "bccaaacb" there are three categories [a,b,c]
MaximaCount of a = 4 at indices 5,6,7,8
MaximaCount of b = 2 at indices 1,2
MaximaCount of c = 6 at indices 2,3,4,5,7,8

complete the findMaximumMaximaCount in the edior below the function returs an integer denoting the maximum attainable favourabilit

categories = adbcbcbcc
out = 6

the string categories consists of lorcase english characters only.

```
int maxMaximaCount(string s)
{
    priority_queue<pair<int, char>> pq; // To get the character with the maximum
frequency at any time
    map<char, int> mp; // A map that will store the frequency of each character
    map<int, set<char>> freq; // A map to store all the characters that have a
same frequency. Used a set because searching and deleting in set will be faster
    map<char, int> maxima; // A map that will store the maxima of a character
    for (auto &x : s)
    {
        if (mp[x] > 0)
        {
            auto &cur = freq[mp[x]];
            cur.erase(cur.find(x)); // erasing the character from the old
frequency
        }
        mp[x]++; // increasing the frequency of the current character and then
adding it to the frequency map and priority queue
        freq[mp[x]].insert(x);
        pq.push({mp[x], x});
        int val = pq.top().first; // getting the element with the maximum
frequency at the current position.
        for (auto &y : freq[val]) // next updated the maxima of all the elements
that have the highest frequency
            maxima[y]++;
    }
    int ans = 0;
    for (auto &x : maxima)
        ans = max(ans, x.second); // finding the max maxima
    return ans;
}
```

44. check similar password

Amazon wouilf like to enfoce a new password policy taht when a user changes their password, the new password cannot be similar to the current one. To determine wheather two passwords are similar, they take the new password choose a set of indices and change the characthers at these inddices to the next cyclic character exactly once Characther 'a' is changed to 'b', 'b' to 'c' and so on, and 'z' changes to 'a' the password is said to be similar after applying the operation the old [assword is a subsequeunce of the new password

the developers come up with a set of n password change requests, where newPasswords denores the array of new passwords and oldPasswords denores the array of old password. For each pair newPasswords[i] and oldPasswords[i], return "YES" if the passwords are similar, that is, newPasswords[i] becomes a subsquence of oldPasswords[i] after performing the operation and "NO" otherwise

Example the two lists of passwords are given as newPasswords = ["baacbab","accdb", "baacba"], and oldPasswords = ["abdbc", "ach", "abb"]
consider the first pait: newPasswords[0] = "baacbab" and oldPasswords = "abdbc". Change "ac" to "bd" at the 3rd and 4th postions and "b" to "c" at the last position
the answer for the pair is YES

the newPasswords[1] = "accdb" and oldPasswords = "ach". it is not possible to change the character of the new password to "h" which occours in the old password so there is no subsequence that matches the answer for this pair is NO

newPasswords[2] = "baacba" and oldPasswords = "abb" the asnwer for this pair is YES

return ["YES", "NO", "YES"

Starting code is

public static List checkSimilarPasswords(List newPasswords, List oldPasswords){

}

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cin >> n;
    vector<string> nwp;
    vector<string> oldp;
    int s = 0, mx = 0;
    for (int i = 0; i < n; i++)
    {
        string p;
        cin >> p;
        nwp.push_back(p);
    }
    for (int i = 0; i < n; i++)
    {
        string p;
        cin >> p;
        oldp.push_back(p);
    }
    for (int i = 0; i < n; i++)
    {
        int p1 = 0;
        for (int j = 0; j < nwp[i].size(); j++)
```

```
        {
            if (oldp[i][p1] == nwp[i][j] || oldp[i][p1] == nwp[i][j] + 1 ||
(oldp[i][p1] == 'a' && nwp[i][j] == 'z'))
            {
                p1 += 1;
            }
            if (p1 == oldp[i].size())
                break;
        }
        if (p1 == oldp[i].size())
            cout << "YES";
        else
            cout << "NO";
    }
}
```

45.makePowerNonDecreasing

n left

1. Code Question 1

Lang

AWS provides scalable systems. A set of n servers are used for horizontally scaling an application. The goal is to have the computational power of the servers in non-decreasing order. To do so, you can increase the computational power of each server in any contiguous segment by *x*. Choose the values of *x* such that after the computational powers are in non-decreasing order, the sum of the *x* values is minimum.

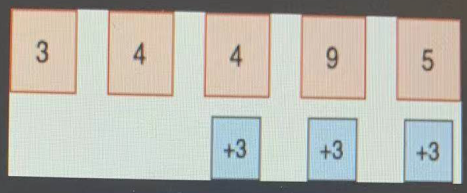
Example

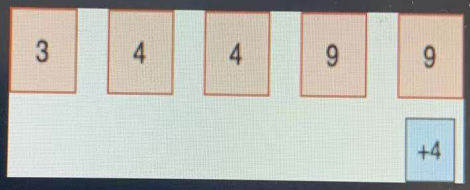
There are n = 5 servers and their computational power = [3, 4, 1, 6, 2].

Add 3 units to the subarray (2, 4) and 4 units to the subarray (4, 4). The final arrangement of the servers is: [3, 4, 4, 9, 9]. The answer is 3 + 4 = 7.

19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41

Add 3 units to the subarray (2, 4) and 4 units to the subarray (4, 4). The final arrangement of the servers is: [3, 4, 4, 9, 9]. The answer is 3 + 4 = 7.

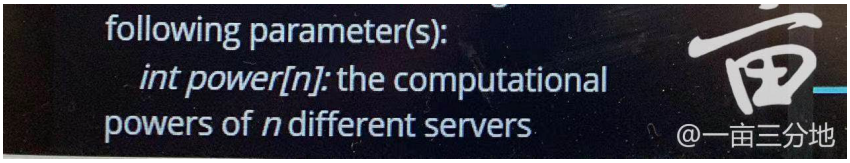




Function Description

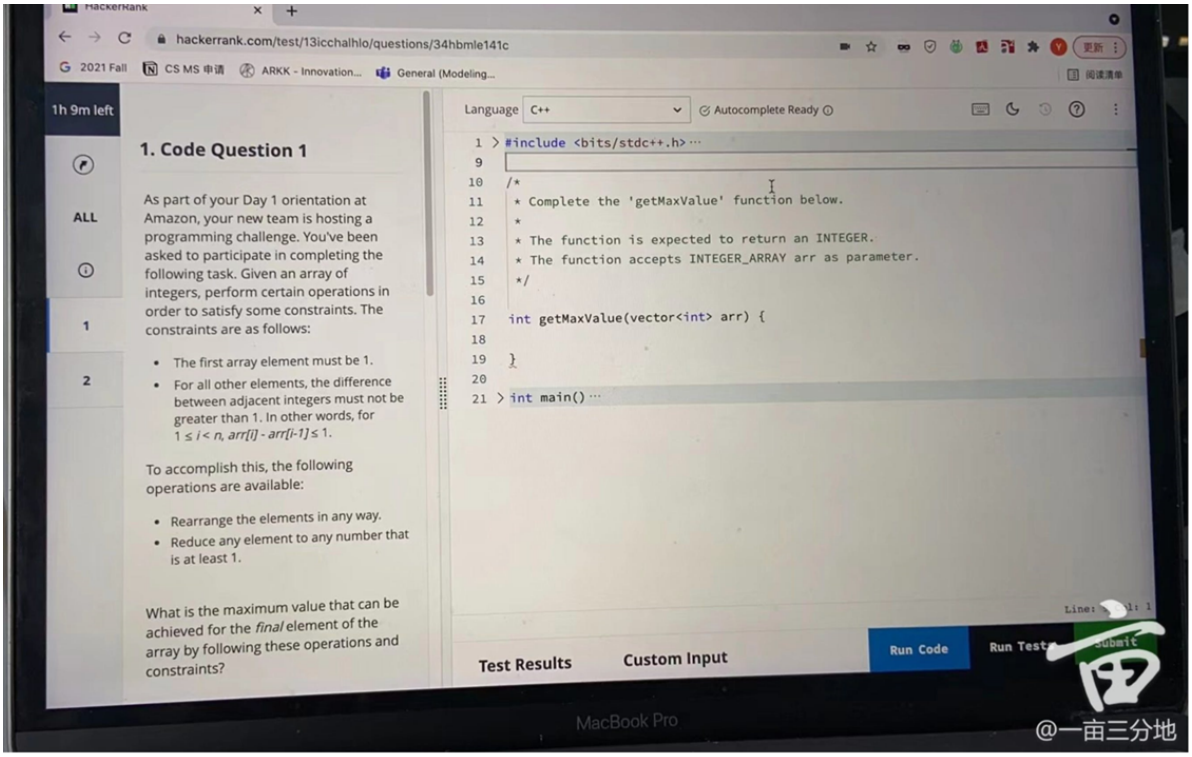
Complete the function *makePowerNonDecreasing* in the editor below.

makePowerNonDecreasing has the



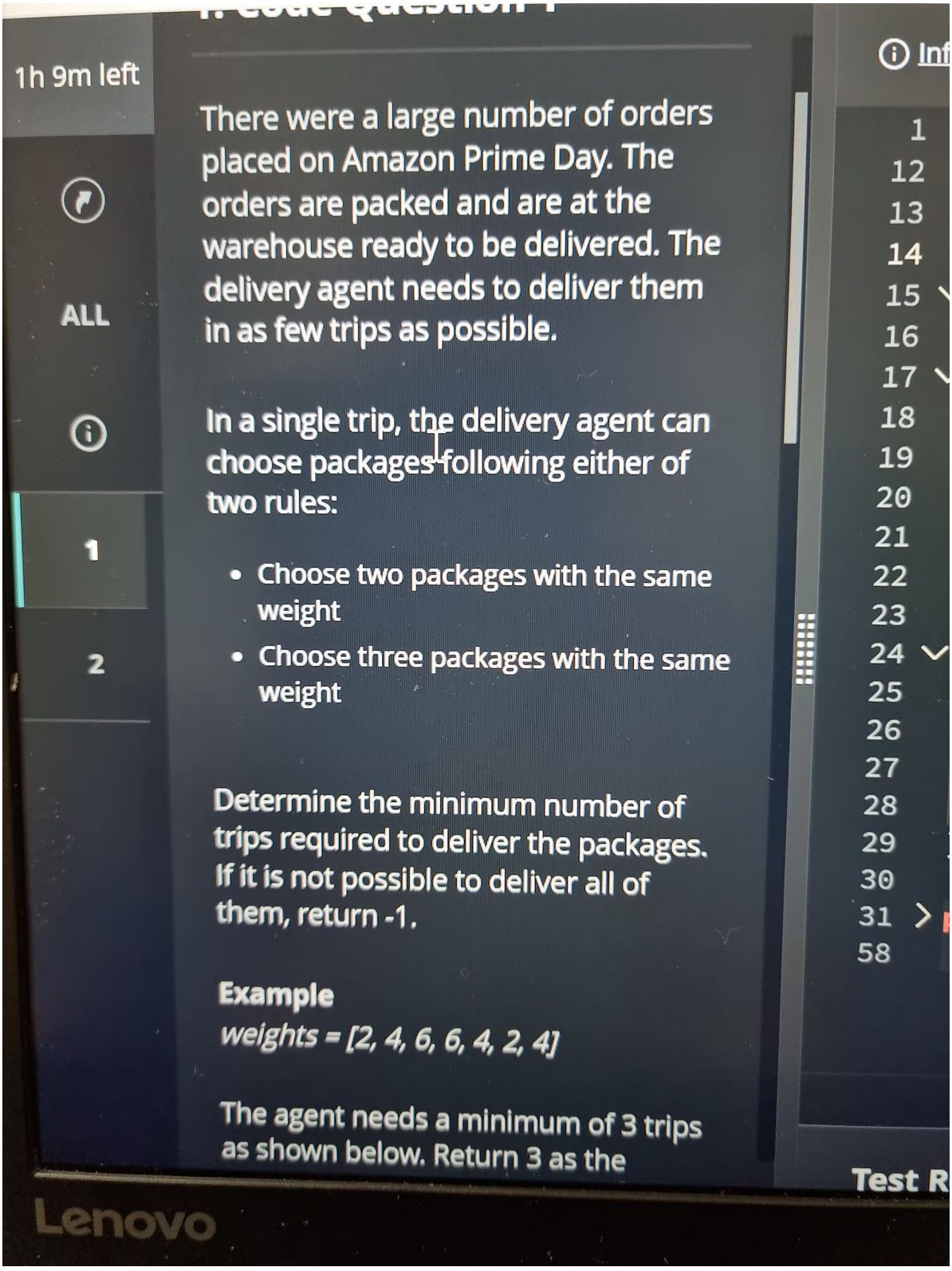

```
vector<int> makePowerNonDecreasing(vector<int> v){
    int result=0;
    if(v.size()==1)
        return v
    for(int i=1;i<v.size();i++){
        v[i]+=result;
        if(v[i]<v[i-1])
            result += v[i-1]-v[i];
    }
    return result;
}
```

46 getmaxValue, array starting with one, adjacent value should have difference <=1



```cpp
class Solution {
public:
    int maximumElementAfterDecrementingAndRearranging(vector<int>& arr) {
        sort(arr.begin(),arr.end());
        arr[0]=1;
        for(int i=1;i<arr.size();i++){
            if(arr[i]>arr[i-1]+1)
                arr[i]=arr[i-1]+1;
        }
        return arr[arr.size()-1];
    }
};
```

47. 搬箱子 一次搬2/3个 count frequency + coin change, hash_map + dp（LC 2244）




```cpp
class Solution {
public:
    int minimumRounds(vector<int>& tasks) {
        unordered_map<int,int> m;
        int l=0;
        int result=0;
        for(auto i:tasks){
            m[i]++;
            l=max(l,m[i]);
        }
        l=max(l,4);
        vector<int> dp(l+1,0);
```

```
        dp[2]=1;
        dp[3]=1;
        dp[4]=2;
        for(int i=5;i<=l;i++){
            dp[i]=min(dp[i-2]+1,dp[i-3]+1);
        }
        for(auto tmp:m){
            if(tmp.second == 1)
                return -1;
            result += dp[tmp.second];
        }
        return result;
    }
};
```

#

48 Merge Intervals

```
vector<vector<int>> getNonInclusiveRetailor(vector<vector<int>> v){
    sort(v.begin(),v.end());
    int result = 0;
    vector<int> current;
    for(int i=0;i<v.size();i++){
        if(current.empty()||v[i][0]>=current[1])
            current = v[i];
        else{
            current[1]=max(current[1],v[i][1]);
            result++;
        }
    }
    return result;
}
```

49 number of subarray with k odd numbers

brute force: O(n^2)

```
int countSubarray(vector<int> v, int k){
    vector<int> ind; //record the index of odd numbers
    ind.push_back(-1);
    for(int i=0;i<v.size();i++){
        if(i%2==0)
            ind.push_back(i);
    }
    ind.push_back(v.size());
    for(int j=1;j+k-1<ind.size();j++){
        int left = ind[j]-ind[j-1];
        int right = ind[j+k]-ind[j+k-1];
        result += left*right;
    }
    return result;
}
```

50 find longest non-increasing singlylindedlist

max consective ones(LC 1004)

把 k 个 0 变成 1 后最长的 1, 滑动窗口保持当前window最多只有k个0.

```
class Solution {
public:
    int longestOnes(vector<int>& nums, int k) {
        int left=0;
        int right=0;
        int result=0;
        while(right<nums.size()){
            if(nums[right]==0)
                k--;
            while(k<0)
                k += (nums[left++]==0);
            result=max(result,right-left+1);
            right++;
        }
        return result;
    }
};
```

Minimum number of swaps required to make a binary string palindrome

https://www.1point3acres.com/bbs/thread-922001-1-1.html

https://devsolus.com/2022/04/26/minimum-number-of-swaps-required-to-make-a-binary-string-palindrome/

Give a binary string consisting of 0's and 1's only. E.g., 0100101. We are allowed to pick any two indexes and swap them. We have to return the minimum number of swaps required to make
it a palindrome or -1 if it cannot. The string 0100101 can be made a palindrome by swapping (3,4)-> 0101001 and swapping (0,1) -> 1001001 which is a palindrome. In this case, the correct answer is 2.

You could compare the outermost two digits: if they are the same, nothing needs to happen, nor should these digits ever be involved in a swap.

If they are different, take note of that, and don't swap anything yet.

Continue with the second digit at each end. Compare them. If they are different, and this is the second time we have a difference, note how we can solve those two differences with one swap. An example:

```
10.....10
```

With one swap this can become:

```
10.....01
```

So we can resolve two differences with one swap. This means that if we find and *even* number of differences — as we walk along the binary string from the two ends inwards — there is a solution, and it takes half that number of swaps.

If the number of differences is odd, there is still a solution if the input string is odd, because then that middle digit can be swapped with a digit from that last difference, and so get a palindrome.

There is however *no* solution when the number of differences (in this algorithm) is odd **and** the number of digits in the input is even.

Now, you don't actually have to *perform* the swaps, as the challenge only asks for the *number* of swaps.

This leads to the following code:

```
def numswaps(binary):
    n = len(binary)
    count = 0
    for i in range(n // 2):
        if binary[i] != binary[n - i - 1]:
            count += 1
    if count % 2 == 1 and n % 2 == 0:
        return -1
    return (count + 1) // 2
```

Z Sequence

A Z sequence is defined as:

Zi=P×X(Zi−1)+Q for i>0
Zi = 2 for i=0
X(K) is defined as the number of set bits in the binary form of a number K.
Print the number of set bits in the binary form of ZN

Example

N = 2
P = 1
Q = 3
Approach

So , Z[0] = 2 , Z[1] = P*X[2]+Q. Now X[2] = 1 as 2 can be written as 10 in binary form So, Z[1] = 1*1+3 = 4 . Similarly , Z[2] = P*X[4]+Q .Now 4 can be written as 100 . So , Z[2] = 1*1+3 = 4. Now answer is the number of set bits in Z[2] = 4, so 1.

EXAMPLE:
Sample input
1
3 5 1

Sample output
1
Explanation
Based on expression, sequence will be like 2, 8, 8,...

As N=1 so Z1 is 8 Number of set bits in 8(i.e 1000) is 1

```
#include <bits/stdc++.h>
using namespace std;

int setBitsInZOfI(int n, int p, int q)
{
  // Z(0) = 2
  unsigned int pZ = 2;
  unsigned int ans = pZ;

  for (int i = 1; i <= n; i++)
  {
    ans = p * __builtin_popcount(pZ) + q;
    pZ = ans;
  }
  return __builtin_popcount(ans);
}
```

被2或者4整除

BETA Can't read the text? Switch theme

1. Code Question 1

Given an integer denoting a total number of wheels, help Amazon Logistics find the number of different ways to choose a fleet of vehicles from an infinite supply of two-wheeled and four-wheeled vehicles such that the group of chosen vehicles has that exact total number of wheels. Two ways of choosing vehicles are considered to be different if and only if they contain different numbers of two-wheeled or four-wheeled vehicles.

For example, if our array *wheels = [4,5,6]* our return array would be *res = [2, 0, 2]*. Case by case, we can have *1* four-wheel or *2* two-wheel to have *4* wheels. We cannot have *5* wheels. We can have *1* four-wheel and *1* two-wheel or *3* two-wheel vehicles in the final case.

Function Description

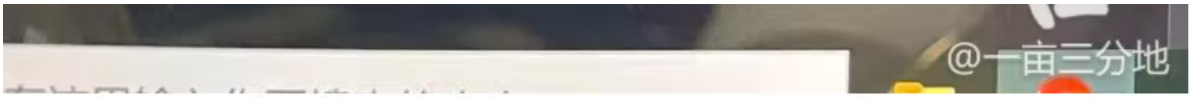
Complete the function *chooseFleets* in the editor below. The function should return an array of integers representing the answer for each *wheels[i]*.

chooseFleets has the following parameter(s):

wheels[wheels[0],...wheels[n-1]]: an array of integers

Constraints

- $1 \leq n \leq 10^5$
- $1 \leq wheels[i] \leq 10^6$

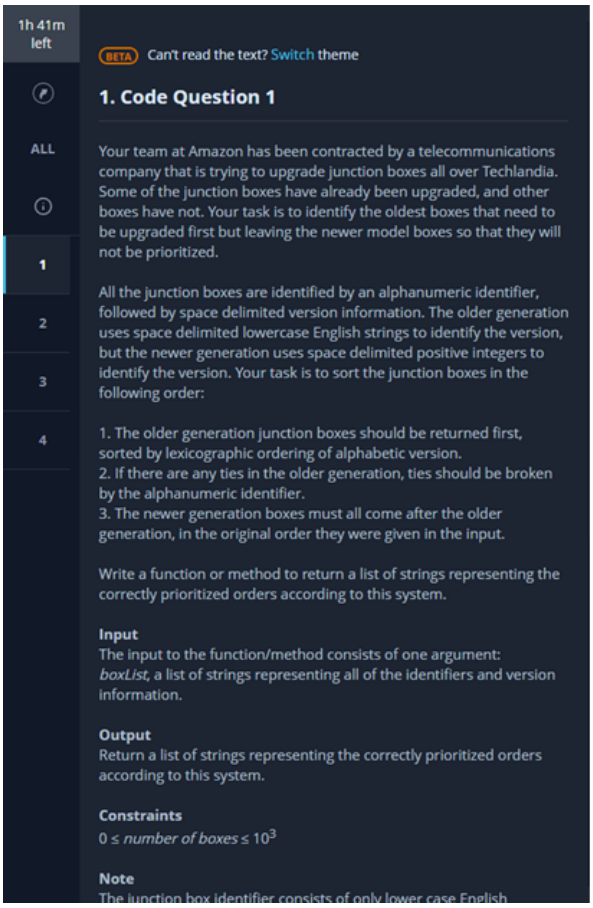


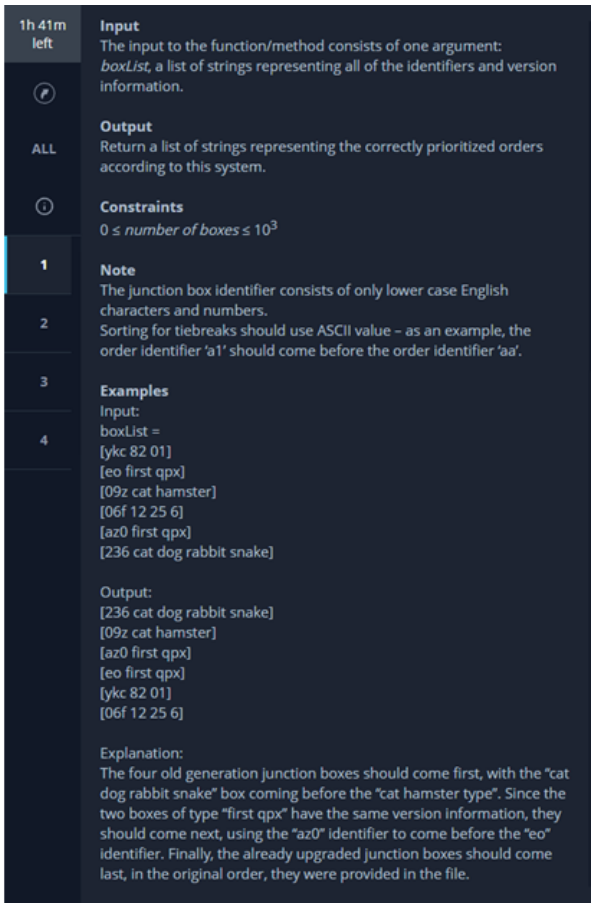
LC 1730

https://www.1point3acres.com/bbs/thread-915292-1-1.html

box 版本号

https://www.1point3acres.com/bbs/thread-922190-1-1.html

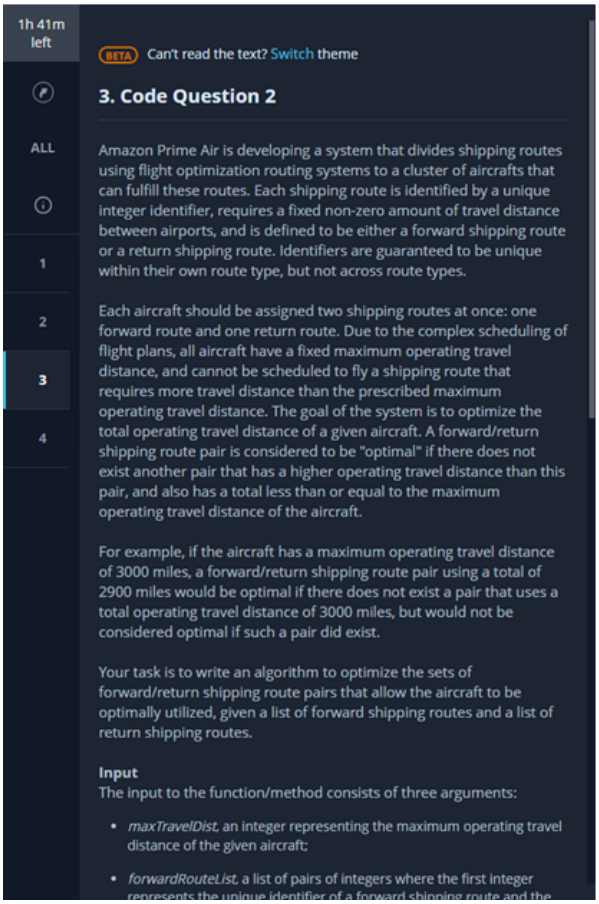


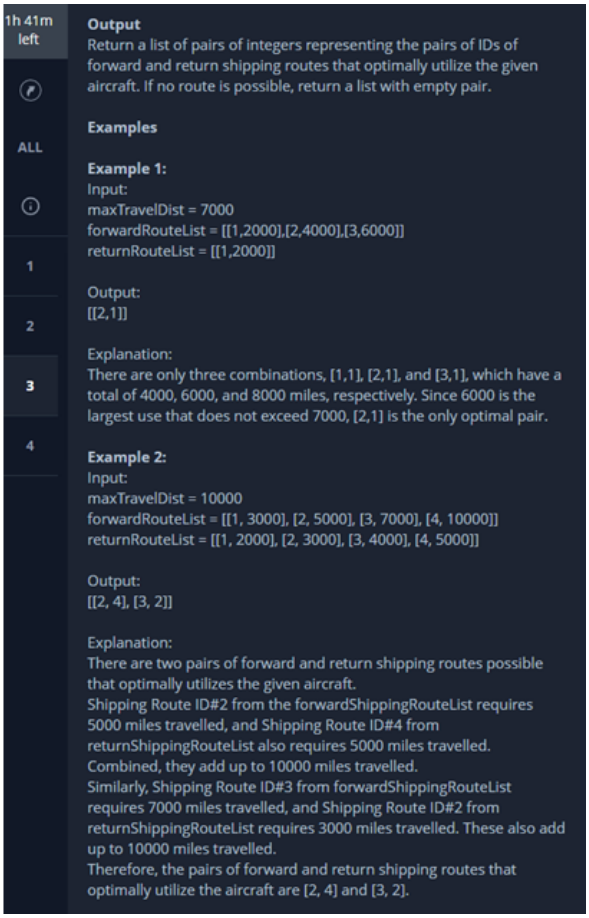


先把新的旧的根据string最后一个字母分出来

然后对旧的排序，再合并到一起。

Route Pair





#

Minimum difference (LC 2256)