CyberChallenge.IT 2025 Programming Solutions

Contents

1 Keyboard [40 points]

1.1 Problem Statement

Lorenzo owns a very old keyboard, which can only type lowercase letters. Furthermore, he spilled some water on it yesterday, so the keys are acting weird now. Specifically, every time he types a string, one and exactly one of the keys does not work. By looking at some strings he typed, can you identify which key stopped working for each string?

1.2 Problem Details

You get a series of strings, each of them composed by lowercase letters only. The number of distinct letters in each string is 25. Find the missing character.

Input

The input consists of $2N + 1$ lines:

- Line 1: an integer, N, representing the number of strings
- Lines $2, ..., 2N + 1$: the length of each string as an integer followed by the string itself, alternated line by line, so that line 2 contains the length of the first string, line 3 contains the first string, line 4 contains the length of the second string, line 5 contains the second string, and so on.

Output

The output consists of N lines, each of them representing the broken key for the corresponding input string.

Scoring

Your program will be tested on a number of testcases grouped in subtasks. In order to obtain the score associated to a subtask, you need to correctly solve all its testcases.

Denoting with L the length of the strings:

- **Subtask 1** [**20 points**]: $L = 25$, $30 \leq N \leq 50$
- **Subtask 2** [**20 points**]: $25 \leq L \leq 4000$, $300 \leq N \leq 1000$

Examples

INPUT	OUTPUT

`2`
`25`
`vfzascmtlngpeuyirxkowbdqh`
`37`
`rpogtvschbedjqmyliaxuzfnkdhryxkalrnqs`
	`j`
	`w`

Explanation

The input consists of two strings, respectively 25 and 37 characters long. We also know from the problem details that all strings contain 25 distinct characters. The only absent letter in the first string is the letter j; the only absent letter in the second string is the letter w. Therefore, they are the broken keys.

1.3 Solution

Each string is guaranteed to contain exactly 25 distinct lowercase letters. Since the English alphabet has 26 lowercase letters, it means exactly one letter is missing in each string; the missing letter corresponds to the broken key. To solve the problem, we can simply iterate over the entire alphabet and check which letter is not present in the input string. The broken key is the one that does not appear in the string.

1.4 Source Code

C++

```cpp
#include <stdio.h>
#include <iostream>
#include <vector>
#include <array>

using namespace std;

string find_key(int L, string s){
    string alph = "abcdefghijklmnopqrstuvwxyz";

    for(auto c : alph){
        if(s.find(c) == std::string::npos)
            return std::string(1,c);
    }
}

int main(){
    int N;
    cin >> N;

    while(N--){
        int L;
        string s;

        cin >> L;
        cin >> s;
        cout << find_key(L, s) << endl;
    }

    return 0;
}
```

Python

```python
import string

def solve(s):
    letters = string.ascii_lowercase
    for c in s:
        letters = letters.replace(c, "")
    return letters

N = int(input())

for _ in range(N):
    __ = input()
    s = input().strip()
    print(solve(s))
```

2 Aliens [80 points]

2.1 Problem Statement

The CyberChallenge.IT space agency managed to intercept messages from an alien species that communicates not that efficiently. It took years and efforts to reverse engineer their language, but they finally made it. The aliens never use words directly: they build their messages by using operations; adding, deleting, swapping and rotating characters.

They send hundreds of operations a day, which are carefully collected continuously, but the space team is so tired of doing all the work manually. They need someone who can make the entire decoding automatic. Someone who can help them build messages starting from operations. Can you be that one?

2.2 Problem Details

You start from an empty string and process an **ordered** series of operations. Each operation can be:

- `add c`: appends character `c` at the end of the string
- `del`: removes last character from the string (if any)
- `swap a b`: swaps character `a` with character `b`, meaning that all occurrences of character `a` are replaced with character `b`, and all occurrences of character `b` are replaced with character `a`
- `rot x`: applies `rot-x` to the current string

Notes:

- each operation is only applied to the string as it is before the operation itself, and it does not influence the characters added after the operation is executed
- the string below represents the alphabet of allowed characters:

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
```

- when performing `rot` operations, characters are rotated following the order in the alphabet, rotating to the right. For example, performing `rot-6` on character `y` outputs character `E` (because `E` is 6 places to the right of `y` in the alphabet), and performing `rot-3` on character `8` outputs character `b` (because after the end the alphabet restarts from `a`).

Input

The input consists of $N + 1$ lines:

- Line 1: an integer, N, the number of operations
- Line 2, ..., $N + 1$: the operations, one per line

Output

The output consists of 1 line, containing the string generated by executing the input operations.

Scoring

Your program will be tested on a number of testcases grouped in subtasks. In order to obtain the score associated to a subtask, you need to correctly solve all its testcases.

- **Subtask 1** **[20 points]**: $1 \leq N \leq 10^3$, using only `add` and `del` operations
- **Subtask 2** **[40 points]**: $1 \leq N \leq 10^4$, using all operations
- **Subtask 3** **[20 points]**: $1 \leq N \leq 10^6$, using all operations

Examples

INPUT	OUTPUT
12 del add G add q add F del swap q h add Y add y rot 21 swap a F add j swap 1 j	jC1T1

Explanation

Let's go through each operation. We start from an empty string.

- The first line deletes the last character of the string, but there isn't any, so the operation does nothing
- The second line adds the character `G`, so now the string is: `G`
- The third line adds the character `q`, so now the string is: `Gq`
- The fourth line adds the character `F`, so now the string is: `GqF`
- The fifth line deletes the last character, so now the string is: `Gq`
- The sixth line swaps the character `q` with the character `h`. There is one occurrence of `q`, that will become `h`, but there is no `h`, so the string becomes: `Gh`
- The seventh line adds the character `Y`, so now the string is: `GhY`
- The eight line adds the character `y`, so now the string is: `GhYy`
- The ninth line performs `rot-21` on the string, so the string now is: `1CjT`
- The tenth line swaps the character `a` with the character `F`, but there is no `a` and no `F`, so the string stays the same: `1CjT`
- The eleventh line adds the character `j`, so now the string is: `1CjTj`
- The twelveth line swaps the character `1` with the character `j`, so the string now is: `jC1T1`

2.3 Solution

Subtasks 1 and 2

The first two subtasks can be solved by simulating the instructions directly. Subtask 2 adds the `swap` function: it is important to note that this swap cannot be done by simply replacing all occurrences of one character with another, and then doing the opposite, because that would mix up the values and not produce the correct result. Instead, it is enough to go through the string one character at a time, and whenever one of the two target characters is found, swap it with the other. The overall time complexity should be at least $O(N^2)$, but some $O(N^3)$ solutions could also work.

Subtask 3

Subtask 3 has (at least) two approaches. It can be solved with a decently optimized implementation of the simulation used in subtask 2 (with time complexity $O(N^2)$), or with a more clever algorithm. We present the second approach, which is more interesting.
We consider the operations in reverse order. Suppose at some point we have an `add` instruction: the result of this addition in the final string will depend only on the instructions that are **after** this `add` operation in the code.
Let's then keep a map m that initially maps each possible character to itself. We apply all the operations (starting from the end) to this map and, when we find an `add` instruction, we add the corresponding character in our map. With this approach we process the instructions in linear time instead of quadratic. Note that we also need to keep track of the number of `del` instructions: some of the `add` instructions will not result in the final string.

Algorithmically the approach works like this: we initialize our map m and a counter $c = 0$.

- `sub x y` is processed as $m_x = m_y$
- `rot n` is processed as $m_x = \text{rot}(m_x, n)$ for every possible character x
- `del` increments c
- `add x` adds m_x if $c = 0$, otherwise decrements c

2.4 Source Code

C++

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <unordered_map>
#include <algorithm>

using namespace std;

string solve(const vector<string>& code) {
    unordered_map<char, char> m;
    string alph = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
    int count = 0;
    string s = "";

    for (char c : alph) {
        m[c] = c;
    }

    for (int i = code.size() - 1; i >= 0; --i) {
        string line = code[i];
        vector<string> inst;
        string temp;
        for (char ch : line) {
            if (ch == ' ') {
                if (!temp.empty()) inst.push_back(temp);
                temp.clear();
            } else {
                temp += ch;
            }
        }
        if (!temp.empty()) inst.push_back(temp);

        if (inst[0] == "add") {
            if (count > 0) {
                count--;
            } else {
                s += m[inst[1][0]];
            }
        } else if (inst[0] == "del") {
            count++;
        } else if (inst[0] == "swap") {
            unordered_map<char, char> m2 = m;
            m2[inst[1][0]] = m[inst[2][0]];
            m2[inst[2][0]] = m[inst[1][0]];
            m = m2;
        } else if (inst[0] == "rot") {
            int rot_val = stoi(inst[1]);
            unordered_map<char, char> m2;
            for (char k : alph) {
                int new_pos = (alph.find(k) + rot_val) % alph.size();
                m2[k] = m[alph[new_pos]];
            }
            m = m2;
        }
    }

    reverse(s.begin(), s.end());
    return s;
}

int main() {
    int N;
    cin >> N;
    cin.ignore();
    vector<string> code(N);
    for (int i = 0; i < N; ++i) {
        getline(cin, code[i]);
    }

    cout << solve(code) << endl;

    return 0;
}
```

Python

```python
import string

def solve(code):
    m = {}
    alph = string.ascii_letters+string.digits
    count = 0
    s = ""

    for c in alph:
        m[c] = c

    code = code[::-1]

    for line in code:
        inst = line.strip().split()
        if inst[0] == 'add':
            if count > 0:
                count -= 1
            else:
                s += m[inst[1]]
        elif inst[0] == 'del':
            count += 1
        elif inst[0] == "swap":
            m2 = m.copy()
            m2[inst[1]] = m[inst[2]]
            m2[inst[2]] = m[inst[1]]
            m = m2
        elif inst[0] == "rot":
            m2 = {}
            for k in m:
                t = alph[(alph.index(k)+int(inst[1]))%len(alph)]
                m2[k] = m[t]
            m = m2
    return s[::-1]

N = int(input())
code = []
for _ in range(N):
    code.append(input().strip())

print(solve(code))
```

3 Emails [80 points]

3.1 Problem Statement

Giovanni needs to send some emails to Giulia. Once sent, these emails go through N servers, which forward them in order. So when an email leaves Giovanni's laptop, it goes through the first server, then the second, the third and so on, until it gets to server N, which finally forwards it to Giulia. These servers, though, act in a weird way:

- each server only forwards emails when the current minute is a multiple of a specific number, let's say t. For example, if server X forwards only when the current minute is a multiple of 4, and an email arrives at minute 22, the server will wait until minute 24 to send it
- each server needs some technical time to process and forward the email, so it introduces a delay, specific for the server itself, named f. In other words when a server sends an email, the next one receives it after f minutes.

3.2 Problem Details

You are given the number of servers and their characteristics (the values t and f for each server); you are also given the number of sent emails and the minute the email was sent for each of them. You need to compute the sum of all emails' arrival minutes.

Note: each e-mail must be processed indipendently from the others, as if it was the only one being sent. Different emails do not influence each other's timings or paths in any way.

Input

The input consists of 4 lines:

- Line 1: N and M, the number of servers and emails, as two space-separated integers
- Line 2: N space-separated integers, where the i-th integer represents the value t for the i-th server
- Line 3: N space-separated integers, where the i-th integer represents the value f for the i-th server
- Line 4: M space-separated integers, where the i-th integer indicates the minute the i-th email was sent

Output

The output consists of 1 integer, containing the sum of the computed arrival minutes for each emails.

Scoring

Your program will be tested on a number of testcases grouped in subtasks. In order to obtain the score associated to a subtask, you need to correctly solve all its testcases.

- **Subtask 1** [**20 points**]: $1 \leq N \leq 500, M = 1, 1 \leq t \leq 10, 1 \leq f \leq 10$
- **Subtask 2** [**30 points**]: $1 \leq N \leq 10^3, 1 \leq M \leq 100, 1 \leq t \leq 10, 1 \leq f \leq 10$
- **Subtask 3** [**30 points**]: $1 \leq N \leq 10^5, 1 \leq M \leq 10^6, 1 \leq t \leq 10, 1 \leq f \leq 100$

Examples

INPUT	OUTPUT

4 2
2 7 4 5
1 2 3 4
3 10
	43

Explanation

There are four servers and two emails:

- The first email leaves Giovanni's laptop at minute 3, and reaches the first server. Here, it needs to wait until minute 4 before it can be forwarded (4 is a multiple of 2, which is the value t for the first server). It will need 1 minute to be processed, because $f = 1$ for the first server, so it will reach the second server at minute 5. It will wait until minute 7 to be forwarded, it will need 2 minutes to be processed and will reach server 3 at minute 9. There, it will depart again at minute 12, it will take 3 minutes to be processed and get to server 4 at minute 15. Since 15 is a multiple of 5, it will be forwarded again immediately, and after 4 minutes of processing time, it will finally reach Giulia at minute 19.
- Following the same reasoning, the second email leaves Giovanni's laptop at minute 10 and reaches Giulia at minute 24.
- The output is $19 + 24 = 43$.

3.3 Solution

Subtasks 1 and 2

We consider the first two subtask together, as the only difference is that the number of emails is always 1 in subtask 1, while it is variable in subtask 2. We show how to solve the problem for a single email, then subtask 2 can be solved by repeating the approach in a loop. To solve the problem for a single email, the easiest approach (which works in this case!) is to simulate the process: if an email reaches a server at a certain time x we have two cases:

- x is not multiple of t, then the email will be processed at time $y = t \cdot \lceil x/t \rceil$ and reach the next server at $y + f$
- x is a multiple of t, so the email is processed immediately and reaches the next server at time $x + f$

Iterating on all the servers we get the answer. The total running time is $O(NM)$.

Subtask 3

The previous approach does not work anymore for subtask 3 (hopefully), as the number of emails is too high. Let $G(x)$ denote the arrival time of an email starting at time x and let $L = \text{lcm}(t_1, ..., t_N)$. It can be proven (for example by induction) that $G(x + L) = G(x) + \lfloor x/L \rfloor \cdot L$, which means that the time taken by an email to reach the last server depends only on its value modulo L. A detailed proof will not be presented, but you can think about it as follows: the difference $y - x$ in the equation we used previously ($y = t \cdot \lceil x/t \rceil$) depends only on the value of $x \pmod t$ and not on x itself. Then when two numbers x, x' are the same modulo all the values of t in the path from Giovanni to Giulia, the difference $G(x) - x$ and $G(x') - x'$ are the same. Saying that the values are equal modulo every $t_1, t_2, t_3, \ldots$ means that they are equal modulo the least common multiple $L = \text{lcm}(t_1, t_2, t_3, \ldots)$. So the sequence $G(0) - 0, G(1) - 1, ...$ is periodic with period L. We then need to calculate the arrival times only up to L, moving the time complexity to $O(NL + M)$.

3.4 Source Code

C++

```cpp
#include <stdio.h>
#include <iostream>
#include <vector>
#include <array>

using namespace std;

long long find_sum_of_times(int N, int M, vector<int> t, vector<int> f, vector<int> emails){
    vector<long long> starts;
    long long ans = 0;
    int l = 2520;

    for(int start_time = 0; start_time < l; start_time++){
        int tt = start_time;
        for(int i = 0; i < N; i++){
            if(tt%t[i] != 0)
                tt = t[i] * (tt/t[i] + 1);
            tt += f[i];
        }
        starts.push_back(tt);
    }

    for(auto x : emails){
        ans += starts[x%l] + x/l * l;
    }

    return ans;
}

int main(){
    int N, M;
    cin >> N >> M;
    vector<int> t(N), f(N), emails(M);

    for(auto& x : t) cin >> x;
    for(auto& x : f) cin >> x;
    for(auto& x : emails) cin >> x;

    cout << find_sum_of_times(N, M, t, f, emails);

    return 0;
}
```

Python

```python
import math

def solve(N,M,T,F,S):
    l = math.lcm(*list(range(1,11)))
    prec = []
    for start_time in range(l):
        t = start_time
        for i in range(N):
            if t % T[i] != 0:
                t = T[i] * (t//T[i] + 1)
            t += F[i]
        prec.append(t)

    ans = sum([prec[x%l] + x//l * l for x in S])
    return ans


N, M = map(int, input().strip().split())
T = list(map(int, input().strip().split()))
F = list(map(int, input().strip().split()))
S = list(map(int, input().strip().split()))

ans = solve(N,M,T,F,S)
print(ans)
```

4 Indexes [100 points]

4.1 Problem Statement

The CyberChallenge.IT authors are tired of finding weird stories for the problems they create. They prefer to write numbers. One of them starts writing a sequence $(a_1, a_2, ..., a_N)$ of N positive integers; each of them is between 1 and M and each of them can be repeated. You need to help him finding how many pairs of integers $l < r$ are there such that the sequence $(a_l, a_{l+1}, ..., a_{r-1}, a_r)$ contains at least one tuple (i, j, k) of **distinct** indexes such that $a_i a_j = a_k^2$.

4.2 Problem Details

You are given a N-long array in the form of space-separated integers. You need to find the number of **distinct** pairs (l, r) that respect the condition mentioned in the problem statement. Note: given that you only need to consider **distinct** pairs (l, r), if there are multiple tuples (i, j, k) respecting the required property inside the same interval, the interval should be counted once.

Input

The input consists of $2T + 1$ lines:

- Line 1: an integer, T, representing the number of sequences
- Lines $2, ..., 2T + 1$: the length N of each sequence as an integer followed by the sequence itself as space-separated integers, alternated line by line, so that line 2 contains the length of the first sequence, line 3 contains the first sequence, line 4 contains the length of the second sequence, line 5 contains the second sequence, and so on.

Output

The output consists of T lines. Each of them representing the number of pairs (l, r) for the corresponding input sequence.

Scoring

Your program will be tested on a number of testcases grouped in subtasks. In order to obtain the score associated to a subtask, you need to correctly solve all its testcases.

- **Subtask 1** **[20 points]**: $1 \le N \le 20$, $1 \le M \le 10$
- **Subtask 2** **[40 points]**: $1 \le N \le 10^4$, $1 \le M \le 10$
- **Subtask 3** **[30 points]**: $1 \le N \le 10^5$, $1 \le M \le 100$
- **Subtask 4** **[10 points]**: $1 \le N \le 10^5$, $1 \le M \le 10^4$

Examples

INPUT	OUTPUT

2
4
1 4 3 2
10
2 9 3 1 3 4 10 8 7 1
	1
	14

Explanation

The input contains 2 testcases. In the first one it is clear that the only possible way to pick indexes is $4 \cdot 1 = 2^2$, which can be picked only with $(l, r) = (1, 4)$. In the second case we can take again the same combination with (l, r) in $(1, 6), (1, 7), (1, 8), (1, 9), (1, 10)$. Another combination is $9 \cdot 1 = 3^2$. This can be done picking the subarrays corresponding to $(1, 4), (1, 5)$ and again the ones used before, that should not be counted twice. Moreover it can be done with the subarrays corresponding to $(2, 4), (2, 5), (2, 6), (2, 7), (2, 8), (2, 9), (2, 10)$. There are no more ways to select indexes.

4.3 Solution

Subtask 1

The problem is asking the number of contiguous subsequences of $(a_1, ..., a_N)$ such that for every tuple (i, j, k) of distinct indexes we have $a_i a_j \neq a_k^2$. Since for this subtask N is at most 20, we can simply enumerate all the possibilities: we have $O(N^2)$ ways to select the boundaries of our sequence, and $O(N^3)$ possible tuples, for a total running time of $O(N^5)$.

Subtask 2

In this subtask we cannot bruteforce everything anymore. Let's focus our attention on the equation we need to (not) satisfy: the easiest solution to $a_i a_j = a_k^2$, if we can choose the values, is simply $a_i = a_j = a_k$. Consider now the possible different values of the elements of our array: we have only integers between 1 and $M = 10$, that is, we have 10 possibilities for each value. By the pigeonhole principle we know that if we take 21 values from a set of 10, at least 3 of them must be equal. This implies that every subsequence with length at least 21 contains at least one solution to our equation. We then need to check only the subsequences with $r - l < 2M + 1$. This can be done naively in $O(NM^4)$ running time, that fits our constraints.

Subtask 3

Subtask 3 is just an optimization of subtask 2: in fact it is not necessary to check all the possible tuples (i, j, k) inside every interval, but we can keep track of what we already checked during the previous iterations. For example, if we need to check all the tuples in the subsequence $(a_4, a_5, ..., a_{10})$ but we already checked all the tuples in the subsequence $(a_4, a_5, ..., a_9)$, we only need to check the tuples containing a_{10}. The time complexity of this approach is $O(NM^2)$. A snippet of the implementation of this approach is given below.

```
for(int l = 0; l < N; l++){
   for(int r = 0; r < N; r++){
       bool found = false;
       for(int i = l; i < r; i++){
           for(int j = i+1; j < r; j++){
               if(a[j]*a[j] == a[i]*a[r] || a[i]*a[i] == a[j]*a[r] || a[r]*a[r] == a[i]*a[j])
                   found = true;
           }
        }
        if(found){
            ans += N-r;
            break;
        }
    }
}
```

Subtask 4

The approach for subtask 4 is slightly different. Let $S_{i,j}$ denote the smallest $j > i$ such that $a_i = j$. This sequence can be constructed in $O(NM)$ in both space and time. Let now m_i be the minimum index such that in the subsequence $(a_i, ..., a_{m_i})$ there is at least a solution to our equation. If we can create this sequence, then the solution to the problem is simply $\sum_{i=0}^{N} N - m_i$. Let's build this sequence in 2 steps:

- Let m'_i be the smallest j such that there exist $i < k < j$ such that, for some ordering of the tuple (i, j, k), there is a solution of our equation. This can be calculated efficiently (again in $O(MN)$) using our precomputed table S looking directly for the expected missing value in the equation and verifying it.
- At this point $m_N = m'_N$ and $m_i = \min(m_{i+1}, m'_i)$ (be careful to calculate these going backwards!)

The source code is given below.

4.4 Source Code

C++

```cpp
#include <stdio.h>
#include <iostream>
#include <vector>
#include <array>
#include <cmath>
#include <algorithm>

using namespace std;

long long count_intervals(int N, vector<int> a){
    long long ans = 0;

    int M = (*std::max_element(a.begin(), a.end())) + 1;
    vector<vector<int>> succ(N+1, vector<int>(M, 0));

    for(int i = 0; i<M;i++){
        succ[N][i] = N;
    }

    for(int i = N-1; i>-1; i--){
        for(int j = 0; j < M; j++){
            if(a[i] == j)
                succ[i][j] = i;
            else
                succ[i][j] = succ[i+1][j];
        }
    }

    vector<int> mins(N,N);

    for(int i = 0; i < N; i++){
        for(int j = 1; j < M; j++){
            if((j*j) % a[i] == 0){
                int k = (j*j)/a[i];
                if(k>0 && k < M && succ[i+1][j]<N){
                    mins[i] = min(mins[i],succ[succ[i+1][j]+1][k]);
                }
            }
            if((a[i]*a[i]) % j == 0){
                int k = (a[i]*a[i])/j;
                if(k>0 && k < M && succ[i+1][j]<N){
                    mins[i] = min(mins[i],succ[succ[i+1][j]+1][k]);
                }
            }
            int k = sqrt(a[i]*j);
            if(k*k == a[i]*j){
                if(k>0 && k < M && succ[i+1][j]<N){
                    mins[i] = min(mins[i],succ[succ[i+1][j]+1][k]);
                }
            }
        }
    }

    for(int i = N-2; i>-1; i--)
        mins[i] = min(mins[i], mins[i+1]);

    for(int i = 0; i < N; i++)
        ans += N-mins[i];

    return ans;
}

int main(){
    int T;
    cin >> T;

    while(T--){
        int N;
        cin >> N;
        vector<int> a(N);

        for(auto &x : a) cin >> x;

        cout << count_intervals(N, a) << endl;
    }

    return 0;
}
```

Python

```
import math

def solve(N,a):
    ans = 0
    M = max(a)+1
    succ = [[0 for _ in range(M)] for __ in range(N+1)]

    for i in range(M):
        succ[N][i] = N

    for i in range(N-1,-1,-1):
        for j in range(M):
            if a[i] == j:
                succ[i][j] = i
            else:
                succ[i][j] = succ[i+1][j]

    mins = [N]*N

    for i in range(N):
        for j in range(1,M):
            if (j*j) % a[i] == 0:
                k = j*j//a[i]
                if k > 0 and k < M and succ[i+1][j] < N:
                    mins[i] = min(mins[i], succ[succ[i+1][j]+1][k])
            if (a[i]*a[i]) % j == 0:
                k = a[i]*a[i]//j
                if k > 0 and k < M and succ[i+1][j] < N:
                    mins[i] = min(mins[i], succ[succ[i+1][j]+1][k])
            if int(math.sqrt(a[i]*j))**2 == a[i]*j:
                k = int(math.sqrt(a[i]*j))
                if k > 0 and k < M and succ[i+1][j] < N:
                    mins[i] = min(mins[i], succ[succ[i+1][j]+1][k])

    for i in range(N-2, -1, -1):
        mins[i] = min(mins[i], mins[i+1])

```

```python
    for i in range(N):
        ans += N-mins[i]

    return ans

T = int(input())

for _ in range(T):
    N = int(input())
    a = list(map(int, input().strip().split()))
    print(solve(N,a))
```