CyberChallenge.IT 2024
Programming Solutions

Contents

1 Sanitization [40 points]

1.1 Problem Statement

Alan Good afternoon, everyone, and welcome to the inaugural lecture of CyberChallenge.IT 2024! I'm Alan, your guide through the fabulous world of hacking! Let's kick off with a little magic trick!

Alan opens a web page, types some peculiar characters in an input field, and... Boom!

A cascade of user data appears publicly!

Bob Whoa! You just broke the internet!

Alan Not the whole internet, Bob. That was just a dummy website with fake data prepared for the lecture. But what can we learn from it?

Bob That most of the internet is broken?

Charlie That some inputs can be dangerous?

Alan Spot on, Charlie! So, your first task for the day is crafting a kind of sanitizer! You'll receive a list of banned words and a set of input strings. For each string, simply type SAFE or BANNED, depending on whether the corresponding input contains banned words.

Bob But wasn't this course about hacking?

Meanwhile, Charlie starts typing...

1.2 Problem Details

Input

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

- Line 1: two space-separated integers, N and M
- Lines $2, \ldots, M + 1$: the list of banned words, one per line
- Lines $M + 2, \ldots, M + N + 1$: the list of input strings, one per line

Both banned words and inputs are composed only of lowercase letters, uppercase letters and digits. In particular, they do not contain spaces. Their length is variable between 3 and 20 characters.

Output

The output consists of N lines. Print either `SAFE` if the corresponding input string does **not** contain any of the banned words, or `BANNED` if it contains at least one of them.

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]**: $1 \leq N \leq 100$, $M = 1$
- **Subtask 2** **[20 points]**: $1 \leq N \leq 1000$, $1 \leq M \leq 100$

Examples

INPUT	OUTPUT
5 1 banned safestring bannedstring notbanndstring notBannedstring StillNotb4nned	SAFE BANNED SAFE SAFE SAFE

Explanation

In the testcase we have only one banned word, the "banned" string. We have 5 given inputs, and only the second one contains exactly it as a substring. In particular, we notice that occurrences of "banned" with uppercase letters or digits are of course not counted.

1.3 Solution

This is an implementation problem. For every string in the input list we just need to check if at least one of the banned words is contained in it. If we find it, we'll output `BANNED`, otherwise we'll output `SAFE`.

1.4 Source Code

C++

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

using namespace std;
vector<string> sanitize(vector<string> words, vector<string> banned){
    vector<string> answers;

    for(auto s : words){
        string ans = "SAFE";
        for(auto b : banned){
            if(s.find(b) != string::npos){
                ans = "BANNED";
            }
        }
        answers.push_back(ans);
    }
    return answers;
}

int main(){
    int N, M;
    vector<string> words, banned, answers;
    cin >> N >> M;

    for(int i = 0; i < M; i++){
        string s;
        cin >> s;
        banned.push_back(s);
    }

    for(int i = 0; i < N; i++){
        string s;
        cin >> s;
        words.push_back(s);
    }

    answers = sanitize(words, banned);
    for(auto s : answers){
        cout << s << endl;
    }
    return 0;
}
```

Python

```python
def solve(words, banned):
    anss = []
    for w in words:
        ans = "SAFE"
        for b in banned:
            if b in w:
                ans = "BANNED"
        anss.append(ans)
    return anss

N, M = map(int, input().strip().split())

words = []
banned = []

for _ in range(M):
    banned.append(input().strip())

for _ in range(N):
    words.append(input().strip())

ans = solve(words, banned)

for a in ans:
    print(a)
```

2 Emulation [60 points]

2.1 Problem Statement

Bob Prof. Alan, I think I got infected by malware...

Alan Fear not, dear Bob, it's not malware. It's just a CTF challenge printing amusing things on your terminal. But hey, we can leverage it to dive into some reverse engineering!

Charlie Reverse engineering? What's that all about?

Alan In simple terms, it's dissecting a piece of software to understand its inner workings. In Bob's case, we've got a virtual machine with straightforward custom instructions. One of the initial steps while facing these challenges is usually crafting an emulator. Think you're up for the challenge?

Bob A... What?

Alan Just some code to run arbitrary programs in the given language! I've set up a small table with the available instructions for you. Your mission: give me the result of a given piece of code. Do you think you can print the sum of the variables `a, b, c, d, e, f` at the end of the execution?

Bob starts screaming and runs away...

VM Details
The VM has 6 variables, namely `a, b, c, d, e` and `f` that store integer values. Operations are sums, subtractions, multiplications and conditional jumps to labels. Labels are always identified by 2-characters long name, in lowercase ascii. The syntax is given in the following table.

At the beginning all the 6 variables are set to 0. Operations are always performed between integer values.

Instruction	Syntax	Explanation
Addition	`add <var> <num>`	Adds the value of `num` to the variable `var`. Example: `add a 5` adds 5 to the variable `a`.
Subtraction	`sub <var> <num>`	Subtracts the value of `num` from the variable `var`. Example: `sub a 5` subtracts 5 from the variable `a`.
Multiplication	`mul <var> <num>`	Substitute the value in `var` with its product with `num`. Example: `mul a 5` stores in `a` the value 5`a`.
Labeling	`lab <name>`	Sets a label named `name`. Label names are always 2-characters long. Labels do not affect variables, but can be jumped to with the `jmp` instruction.
Jumping	`jmp <var> <num> <name>`	Jumps at the label `name` if the value stored in the variable `var` is equal to `num`.

2.2 Problem Details

Input

The input consists of $N + 1$ lines:

- Line 1: an integer N, the number of lines the program contains
- Lines $2, \ldots, N + 1$: the instructions of the program, one per line

Output

The output is a single integer, representing the sum `a+b+c+d+e+f`.

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.

For both subtasks, N will not exceed 500. It is also guaranteed that no infinite loops will be given in the testcases (i.e., all the programs terminate). It is also guaranteed that in any point of the computations required, the variables will not exceed 2^{60} or go below -2^{60}.

- **Subtask 1** [**30 points**]: the program only contains `add, sub, mul` instructions.
- **Subtask 2** [**30 points**]: no restrictions on the used instructions.

Examples

INPUT	OUTPUT
10 lab xm add e 33 lab ik add d 79 mul a 48 jmp d 79 xm add c 43 lab ex lab ne jmp c 43 ik	389

Explanation

Let's go through the code step by step:

- The first line creates the label `xm`
- The second line sets `e` to 33
- The third creates the label `ik`
- Fourth and fifth lines set `d` to 79 and `a` to 0
- The sixth line jumps to he label `xm`, since `d` is 79
- Then `e` goes to 66, `d` to 158 and the jump condition is not true anymore, so we go to line 7
- Line 7 sets `c` to 43
- Lines 8 and 9 create two labels, `ex` and `ne`
- Since `c` is 43 we execute the jump at line 10 to label `ik`
- At this point, `d` goes to 237, `a` stays at 0, the jump to `xm` is not executed and in the end `c` goes to 86
- The jump to `ik` is not executed since `c` is `86`
- The final result is $0 + 0 + 86 + 237 + 66 + 0 = 389$

2.3 Solution

The second problem is again an implementation one. The idea is basically to keep a state of the variables and perform the computation in order. We keep the index of the current instruction as an additional variable. For the additions, subtractions and multiplications there is no problem. For the `jmp` instructions, we simply check if the jumping condition is true and find the index of the corresponding label. We move then our computation index to the label's index and restart the computations from there.

2.4 Source Code

C++

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

using namespace std;

long long emulate(vector<string> code){
    int current = 0;
    map<char, long long> var_values = {{'a', 0}, {'b', 0}, {'c', 0}, {'d', 0}, {'e', 0}, {'f', 0},};
    long long ans = 0;

    while(current < code.size()){
        string inst = code[current].substr(0,3);

        if(inst == "add"){
            var_values[(char)code[current][4]] += stoll(code[current].substr(6));
        }
        else if(inst == "sub"){
            var_values[(char)code[current][4]] -= stoll(code[current].substr(6));
        }
        else if(inst == "mul"){
            var_values[(char)code[current][4]] *= stoll(code[current].substr(6));
        }
        else if(inst == "jmp"){
            char var = (char)code[current][4];
            string lab = code[current].substr(code[current].length()-2);
            long long val = stoll(code[current].substr(6, code[current].length()-3-6));

            if(var_values[var] == val){
                current = distance(code.begin(), find(code.begin(), code.end(), string("lab " + lab)));
            }
        }
        current++;
    }

    for(auto &p : var_values){
        ans += p.second;
    }
    return ans;
}

int main(){
    int N;
    vector<string> code;
    string s;
    cin >> N;
    getline(cin, s);
    for(int i = 0; i < N; i++){
        getline(cin, s);
        code.push_back(s);
    }
    cout << emulate(code) << endl;
    return 0;
}
```

Python

```python
from collections import defaultdict

def solve(code):
    current = 0
    var_values = defaultdict(int)

    while current < len(code):
        inst = code[current][:3]

        if inst == "add":
            var_values[code[current][4]] += int(code[current].split()[-1])
        elif inst == "sub":
            var_values[code[current][4]] -= int(code[current].split()[-1])
        elif inst == "mul":
            var_values[code[current][4]] *= int(code[current].split()[-1])
        elif inst == "jmp":
            var, val, lab = code[current].split()[1:]
            if var_values[var] == int(val):
                current = code.index(f"lab {lab}")
        current += 1

    return sum(var_values[x] for x in 'abcdef')

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

sol = solve(code)
print(sol)
```

3 Pattern Recognition [100 points]

3.1 Problem Statement

Charlie Let me get this straight: in binary exploitation, we chuck massive strings into program input fields, and if we spot our string in memory where it shouldn't be, it's a problem, right?

Alan Spot on, Charlie! And there's more: sometimes, we have to delicately craft these strings to precisely pinpoint our location within them.

Bob Can't I just smash the keyboard randomly?

Alan And what if you need thousands of characters?

Bob Easy! I'll make them all the same and randomly tweak a few at the end!

Alan You could end up in a tight spot with that, Bob... Typically, we resort to de Bruijn sequences, but that's a tale for another time!

Bob I'm not interested in that. My method always works in practice! I can prove it!

Alan Alright, Bob, let's play a game: I'll give you a string S. How many strings R exist such that you can cover all of S using only copies of R?

Bob The problem does not even make sense, what do you mean by *cover*?

Alan I mean that I can recreate the string S using copies of R, possibly overlapping them. For example, I can cover the string "*xyxyxy*" with "*xy*", "*xyxy*" and, of course, "*xyxyxy*" itself. Is it clear now?

Bob Uhm, yes, it makes sense...

Alan takes a breath, hoping this will bring a momentary pause to Bob's enthusiasm...

3.2 Problem Details

Input

The input consists of $3T + 1$ lines:

- Line 1: the number T of testcases you would need to answer
- Lines $2, \ldots, 3T + 1$: every group of 3 lines is formatted as follows
  - Line 1: two space separated integers, N and M, respectively the length of the alphabet from which the string S is sampled, and the length of the string S itself
  - Line 2: a string of length N, representing the alphabet
  - Line 3: a string of length M, the actual string S

Output

The output consists of T lines, each representing the answer to the corresponding testcase.

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**]: $1 \leq T \leq 100, N = 2, 1 \leq M \leq 12$
- **Subtask 2** [**50 points**]: $1 \leq T \leq 100, 1 \leq N \leq 12, 1 \leq M \leq 500$
- **Subtask 3** [**30 points**]: $1 \leq T \leq 100, 1 \leq N \leq 20, 1 \leq M \leq 20000$

Examples

INPUT	OUTPUT

```
3
2 11
SG
GGGSGGSGSGG
2 4
PC
CCCC
2 6
HK
HKHKHK
```

```
1
4
3
```

Explanation

The given input contains 3 different testcases:

- The fist one, the string `GGGSGGSGSGG`, can only be covered with the full string itself
- The second one, `CCCC`, can be covered either with `C`, `CC`, `CCC` or `CCCC`
- The third one, `HKHKHK`, can be covered with `HK`, `HKHK` or `HKHKHK`.

3.3 Solution

Subtask 1

For subtask 1, it is sufficient to bruteforce all the possible strings in the alphabet of length at most M. Checking if one of these candidates covers can be done in a naive way iterating through the string as explained in the code below.

```python
def covers_naive(a, b):
    cur_idx = 0
    last_idx = 0

    while last_idx < len(b):
        if a == b[cur_idx:cur_idx+len(a)]:
            last_idx = cur_idx + len(a)
        cur_idx += 1
        if last_idx < cur_idx:
            return False
    return True
```

This leads to an $O(N^M)$ approach.

Subtask 2

For subtask 2 we need a crucial observation: if a string R covers S, then R is a prefix of S (since we need to cover at least the beginning of S). So we can reduce our candidates to the strings that are a prefix of S. Listing all of them is linear in time, while the check we described before is quadratic. This leads to an $O(M^3)$ approach, that is sufficient for this subtask.

Subtask 3

For subtask 3 a quadratic implementation is enough. It can be obtained easily from the subtask 2 solution avoiding direct string comparison, but comparing some sort of hash function of them. One possible solution is using a rolling hash, or some sort of polynomial over the caracters.

Another quadratic solution can be obtained using the KMP algotihm or equivalent pattern-matching algorithms, but it's a bit more complex to implement.

For completeness, we illustrate also an $O(M \log M)$ solution (that can be further obtimized to $O(M)$ using better sorting algorithms). We stress that this is not needed for the problem constraints.

This solution has 2 main ideas:

- Covering strings should not only be prefixes of S, but also suffixes.
- If we take two covering strings R_1 and R_2, then one is a prefix of the other. Assume R_1 is a prefix of R_2. Consider now the starting indices of the copies of R_2 while covering S: this is a subset of the same indices for R_1 by construction. We can use this property to track the occurrences without recomputing everything for every prefix.

We proceed like this:

- We list all the suffixes of S with their starting index, and we sort them in lexicographic order.
- We keep two pointers, starting at the beginning and at the end of the string.
- We shrink the pointers keeping track of the gaps between two successive occurrences of the strings that we want to check (the prefixes), using the ordered suffixes.
- If the maximum of these gaps is shorter or equal than the length of the currently considered string, we increment the answer.

Python code for this approach is provided below.

3.4 Source Code

C++

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

using namespace std;

int count_patterns(int N, int M, string alph, string s){
    vector<pair<string, int>> sfx;
    int ans = 0, max_gap = 1;
    unordered_map<int, array<int, 2>> gaps;
    set<int> alive;
    int left = 0, right = M-1;


    for(int i = 0; i<M; i++){
        sfx.push_back(make_pair(s.substr(i), i));
    }

    for(int i = 0; i < M+1; i++){
        gaps[i] = {max(i-1, 0), i+1};
        alive.insert(i);
    }

    sort(sfx.begin(), sfx.end());

    for(int i = 0; i<M; i++){
        while(left < right){
            if(M - sfx[left].second > i && sfx[left].first[i] == s[i])
                break;
            int idx = sfx[left].second;
            int prec = *(prev(alive.lower_bound(idx)));
            int succ = *alive.upper_bound(idx);
```

```
            alive.erase(idx);
            gaps[prec][1] = succ;
            max_gap = max(max_gap, succ - prec);
            left++;
        }

        while(left < right){
            if(M - sfx[right].second > i && sfx[right].first[i] == s[i])
                break;
            int idx = sfx[right].second;
            int prec = *(prev(alive.lower_bound(idx)));
            int succ = *alive.upper_bound(idx);
            alive.erase(idx);
            gaps[prec][1] = succ;
            max_gap = max(max_gap, succ - prec);
            right--;
        }

        if(max_gap <= i+1)
            ans++;
    }


    return ans;
}

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

    while(T--){
        int N, M;
        string alph, s;
        cin >> N >> M;
        cin >> alph >> s;
        cout << count_patterns(N, M, alph, s) << endl;
    }

    return 0;
}
```

Python

```
from sortedcontainers import SortedSet

def solve(N,M,alph,s):
    sfx = [(s[i:],i) for i in range(len(s))]
    sfx = sorted(sfx)
    ans = 0
    gaps = {i : [max(i-1,0), i+1] for i in range(M+1)}
    alive = SortedSet([i for i in range(M+1)])
    max_gap = 1

    left, right = 0, M-1

    for i in range(M):
        while left < right:
            if M - sfx[left][1] > i and sfx[left][0][i] == s[i]:
                break
```

```
            idx = sfx[left][1]
            left_idx = alive.index(idx)
            prev = alive[left_idx-1]
            succ = alive[left_idx+1]
            alive.discard(idx)
            gaps[prev][1] = succ
            gaps[prev][2] = succ - prev
            max_gap = max(max_gap, gaps[prev][2])
            left += 1

        while left < right:
            if M - sfx[right][1] > i and sfx[right][0][i] == s[i]:
                break
            idx = sfx[right][1]
            right_idx = alive.index(idx)
            prev = alive[right_idx-1]
            succ = alive[right_idx+1]
            alive.discard(idx)
            gaps[prev][1] = succ
            max_gap = max(max_gap, succ - prev)

            right -= 1

        if max_gap <= i+1:
            ans += 1

    return ans

T = int(input().strip())

for _ in range(T):
    N, M = map(int,input().strip().split())
    alph = input().strip()
    S = input().strip()
    ans = solve(N,M,alph,S)
    print(ans)
```

4 Subset Count [100 points]

4.1 Problem Statement

Alan Alright, guys, today we'll delve into the fascinating realm of public-key cryptography!

Bob Public key?! So, we just throw our passwords out there for everyone to see?

Alan ...not quite, Bob. But you're on the right track. In public-key cryptography, we share certain information publicly while keeping the rest private...

Charlie Oh, I get it! It's that thing where we can multiply numbers together, and then it's a real pain to figure out the original numbers!

Alan You've got the gist of it, Charlie. In public-key cryptography, we utilize one-way functions: mathematical problems that are easy to compute but challenging to reverse. It's like creating a maze: simple to design if you know the path, tricky to navigate backward.

Charlie That's what I said, isn't it?

Alan Of course factorization is one example of a hard problem, but there are many more: discrete logarithms, knapsack problems, learning with errors, solving multivariate systems...

Bob interrupts abruptly

Bob If there are so many, can't I just create my own?

Alan No, Bob, that's not how it works...

Bob Wait, hear me out! I give you a big set S of distinct positive integers and a number D. Then, I choose two subsets A and B of S with no elements in common. But here's the twist: if you pick any two numbers in the same subset (either A or B), their difference (in absolute value) won't be greater than D! I bet that given S and D, you can't compute the maximum of the sum of the number of elements of A and the number of elements of B!

Alan This doesn't even make sense, Bob...

Bob Can you compute this number or not?

Alan is left speechless...

4.2 Problem Details

Input

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

- Line 1: the number of testcases T
- Lines $2, \ldots, 2T + 1$: every group of 2 lines is formatted as follows
  - Line 1: two space separated integers, N (the size of the set S) and the number D
  - Line 2: N space-separated integers, representing the set S

Output

The output consists of T lines, each representing the answer to the corresponding testcase.

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** **[30 points]**: $1 \leq T \leq 20, 1 \leq N \leq 500, 1 \leq D \leq 250$
- **Subtask 2** **[30 points]**: $1 \leq T \leq 20, 1 \leq N \leq 10000, 1 \leq D \leq 5000$
- **Subtask 3** **[40 points]**: $1 \leq T \leq 20, 1 \leq N \leq 100000, 1 \leq D \leq 50000$

For each testcase, every entry of the array S is a positive integer not exceeding 2^{60}.

Examples

INPUT	OUTPUT
3 11 500 1598 2672 660 1864 1672 2942 1075 4744 3685 2893 2777 15 100 278 3176 4710 1836 777 3152 584 4548 1126 2195 3482 3945 4201 1556 3140 10 10 431 4202 2861 4287 2514 3694 4068 3125 2083 3434	7 4 2

Explanation

In the first testcase, we can construct the sets as follows: $A = \{2672, 2942, 2893, 2777\}$, $B = \{1598, 1864, 1672\}$, for a total of $4 + 3 = 7$ elements.

In the second one, a possibility for the sets is: $A = \{3176, 3152, 3140\}$, $B = \{278\}$, for a total of 4 elements.

In the third testcase, there are no elements with difference less or equal to 10, so we can pick any pair of numbers and create two one-element sets, resulting in 2 total elements.

4.3 Solution

Subtask 1

The basic idea to solve this subtask is that both subsets must have a minimum. If we know this minimum, we can easily check if a bigger element can fit into the subset. We proceed as follows: we loop two times over the full set to select 2 (distinct) minimums. We then loop over the remaining elements: if they are smaller than both minimums we throw them away, if they are bigger we check if they can fit in at least one of them. If so, we increase our answer. Notice that is it not important if an element ends up being in A or B, since we only care about the total number of used elements. The time complexity of this approach is $O(N^3)$.

Subtask 2

To solve this subtask, a slight improvement of the above approach is enough. We keep the idea of bruteforcing the two minimums, but instead of going through the set as is, we sort the elements at the beginning. We then bruteforce the two minimums and, for each of them, we binary-search the number of elements that can be counted in each subset. Notice that we have to remove the intersection of A and B at the end. This leads to an $O(N^2 \log N)$ algorithm, that is enough to pass the subtask.

Subtask 3

Optimized quadratic solutions may pass this subtask too but, instead, we propose a cleaner $O(N \log N)$ approach. The crucial observation is that, without loss of generality, $\max(A) < \min(B)$. We can then start again by sorting our values and compute two arrays independently: the array of the biggest size of A, knowing that the maximum is at most a fixed $x \in S$, and the one of the biggest size of B knowing that the minimum is at least a fixed $x \in S$. We then merge these results in linear time, taking the maximum. The implementation of this approach follows.

4.4 Source Code

C++

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

using namespace std;

vector<int> getright(vector<int> S, int D){
    vector<int> res(S.size());
    int j = S.size()-1;

    for(int i = S.size()-1; i>=0; i--){
        while(j >= 0 && S[j]-S[i] > D)
            j--;
        res[i] = j;
    }
    return res;
}

vector<int> getleft(vector<int> S, int D){
    vector<int> res(S.size());
    int j = 0;

    for(int i = 0; i<S.size(); i++){
        while(j<S.size() && S[i]-S[j] > D)
            j++;
        res[i] = j;
    }
    return res;
}

int find_subsets(int N, int D, vector<int> S){
    sort(S.begin(), S.end());
    vector<int> rm = getright(S, D);
    vector<int> lm = getleft(S, D);
    int ans = 0;
    vector<int> ls(N), rs(N);

    for(int i = 0; i<N;i++){
        ls[i] = i - lm[i] + 1;
        if(i>0) ls[i] = max(ls[i], ls[i-1]);
    }

    for(int i = N-1; i>=0; i--){
        rs[i] = rm[i] - i + 1;

        if(i<N-1) rs[i] = max(rs[i], rs[i+1]);
    }

    for(int i = 0; i<N-1; i++)
        ans = max(ans, ls[i] + rs[i+1]);

    return ans;
}

int main(){
```

```
    int T;
    cin >> T;

    while(T--){
        int N, D;

        vector<int> S;

        cin >> N >> D;
        for(int i = 0; i < N; i++){
            int x;
            cin >> x;
            S.push_back(x);
        }

        cout << find_subsets(N, D, S) << endl;
    }

    return 0;
}
```

Python

```
def getright(S, D):
    res = [0]*len(S)
    j = len(S)-1

    for i in range(len(S)-1, -1, -1):
        while j >= 0 and S[j]-S[i] > D:
            j -= 1
        res[i] = j

    return res

def getleft(S, D):
    res = [0]*len(S)
    j = 0

    for i in range(len(S)):
        while j<len(S) and S[i]-S[j] > D:
            j += 1
        res[i] = j

    return res

def solve(_S, D):
    S = sorted(_S[::])

    rm = getright(S, D)
    lm = getleft(S, D)
    ans = 0

    ls = [0]*len(S)
    rs = [0]*len(S)

    for i in range(len(S)):
        ls[i] = i - lm[i] + 1

        if i>0:
```

```
            ls[i] = max(ls[i], ls[i-1])

    for i in range(len(S)-1, -1, -1):
        rs[i] = rm[i] - i + 1

        if i<len(S)-1:
            rs[i] = max(rs[i], rs[i+1])

    for i in range(len(S)-1):
        ans = max(ans, ls[i] + rs[i+1])

    return ans

T = int(input().strip())

for _ in range(T):
    N, D = map(int, input().strip().split())
    S = list(map(int, input().strip().split()))

    assert len(S) == N

    ans = solve(S, D)
    print(ans)
```