Competitive Coding @ Berkeley

Decal Lecture #4 - Two Pointers

Welcome Back Everyone!

- Today’s topic will be two pointers!
- If you have not joined the discord yet, please do so with: https://discord.gg/6QYPrP5G2u
- Or scan this QR
- Problemset 4 is out! Feel free to register and check it out

Announcements

- We will have grades ready for next week!
- If you have not given us your codeforces handle and student ID, **please do so ASAP**
- Reminder that submissions are open after the problemset deadlines!
  - This allows you to see solutions
- Competitive Coding @ Berkeley (CCAB) Advanced topic lecture right after this! (8:10PM)
  - Topic: Aho Corasick Algorithm
- A lot of you are new!
  - Two attendance drops
  - Two problemset drops
  - Only first two problems are required on each problemset
    - Graded on effort (making a submission) if you make it to lecture!

Announcements

- One required problem for this week! (The first one called books)
- Assign yourself the DeCal role in the Discord server!
    - React to the 5th message up from “Carl-bot”

Problem C (Dragon Quest Game)

- https://codeforces.com/problemset/problem/1337/B

The dragon has a *hit point* of x initially. When its *hit point* goes to 0 or under 0, it will be defeated. In order to defeat the dragon, Kana can cast the two following types of spells.

- `Void Absorption`
  Assume that the dragon's current *hit point* is h, after *casting* this spell its *hit point* will become $\left\lfloor \frac{h}{2} \right\rfloor + 10$. Here $\left\lfloor \frac{h}{2} \right\rfloor$ denotes h divided by two, rounded down.

- `Lightning Strike`
  This spell will decrease the dragon's *hit point* by 10. Assume that the dragon's current *hit point* is h, after *casting* this spell its *hit point* will be lowered to $h - 10$.

Due to some reasons Kana can only *cast* **no more than** n `Void Absorptions` and m `Lightning Strikes`. She can cast the spells in any order and **doesn't have to** cast all the spells. Kana isn't good at math, so you are going to help her to find out whether it is possible to defeat the dragon.

Input

The first line contains a single integer t $(1 \leq t \leq 1000)$ — the number of test cases.

The next t lines describe test cases. For each test case the only line contains three integers x, n, m $(1 \leq x \leq 10^5, 0 \leq n, m \leq 30)$ — the dragon's intitial *hit point*, the maximum number of `Void Absorptions` and `Lightning Strikes` Kana can *cast* respectively.

Output

If it is possible to defeat the dragon, print "`YES`" (without quotes). Otherwise, print "`NO`" (without quotes).

You can print each letter in any case (upper or lower).

Example

input

```
7
100 3 4
189 3 4
64 2 3
63 2 3
30 27 7
10 9 1
69117 21 2
```

output

```
YES
NO
NO
YES
YES
YES
YES
```

Note

One possible casting sequence of the first test case is shown below:

- `Void Absorption` $\left\lfloor \frac{100}{2} \right\rfloor + 10 = 60$.
- `Lightning Strike` $60 - 10 = 50$.
- `Void Absorption` $\left\lfloor \frac{50}{2} \right\rfloor + 10 = 35$.
- `Void Absorption` $\left\lfloor \frac{35}{2} \right\rfloor + 10 = 27$.
- `Lightning Strike` $27 - 10 = 17$.
- `Lightning Strike` $17 - 10 = 7$.
- `Lightning Strike` $7 - 10 = -3$.

Solution

- **Spell 1**: set dragon’s health ***h*** to **floor**(***h***/2) + 10
- **Spell 2**: set dragon’s health to ***h*** - *10*

Solution

- <u>**Spell 1**</u>: set dragon's health ***h*** to **floor**(***h***/2) + 10
- <u>**Spell 2**</u>: set dragon's health to ***h*** - *10*
- Intuition: **spell 1** deals significantly more damage for large health value ***h***
- So should we always apply spell 1 first???

Solution

- <u>**Spell 1**</u>: set dragon's health ***h*** to **floor**(***h***/2) + 10
- <u>**Spell 2**</u>: set dragon's health to ***h*** - *10*
- Consider the test case ***h*** = 10 and you have one of each spell
  - If we apply spell 1 first, we increase ***h*** to 15 and it becomes impossible!

Solution

- **Spell 1**: set dragon's health ***h*** to **floor**(***h***/2) + 10
- **Spell 2**: set dragon's health to ***h*** - *10*
- Two solutions to overcome this
  - Apply all of **spell 1** first only if ***h*** > 20, then apply all of **spell 2**
  - Alternatively, it suffices to first check if we can win only using **spell 2**.
    - This allows us to remove the condition of checking ***h*** > 20,

Problem E (Create The Teams)

- https://codeforces.com/problemset/problem/1380/C

There are n programmers that you want to split into several non-empty teams. The skill of the i-th programmer is a_i. You want to assemble the maximum number of teams from them. There is a restriction for each team: the number of programmers in the team multiplied by the minimum skill among all programmers in the team must be at least x.

Each programmer should belong to at most one team. Some programmers may be left without a team.

Calculate the maximum number of teams that you can assemble.

Input

The first line contains the integer t ($1 \le t \le 1000$) — the number of test cases.

The first line of each test case contains two integers n and x ($1 \le n \le 10^5; 1 \le x \le 10^9$) — the number of programmers and the restriction of team skill respectively.

The second line of each test case contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le 10^9$), where a_i is the skill of the i-th programmer.

The sum of n over all inputs does not exceed 10^5.

Output

For each test case print one integer — the maximum number of teams that you can assemble.

Example

input Copy

```
3
5 10
7 11 2 9 5
4 8
2 4 2 3
4 11
1 3 3 7
```

output Copy

```
2
1
0
```

There are n programmers that you want to split into several non-empty teams. The skill of the i-th programmer is a_i. You want to assemble the maximum number of teams from them. There is a restriction for each team: the number of programmers in the team multiplied by the minimum skill among all programmers in the team must be at least x.

Each programmer should belong to at most one team. Some programmers may be left without a team.

Calculate the maximum number of teams that you can assemble.

Input

The first line contains the integer t $(1 \le t \le 1000)$ — the number of test cases.

The first line of each test case contains two integers n and x $(1 \le n \le 10^5; 1 \le x \le 10^9)$ — the number of programmers and the restriction of team skill respectively.

The second line of each test case contains n integers $a_1, a_2, \dots, a_n$ $(1 \le a_i \le 10^9)$, where a_i is the skill of the i-th programmer.

The sum of n over all inputs does not exceed 10^5.

Output

For each test case print one integer — the maximum number of teams that you can assemble.

Example

input

```
3
5 10
7 11 2 9 5
4 8
2 4 2 3
4 11
1 3 3 7
```

output

```
2
1
0
```

Solution

- Process the students in descending order of skill
- Let **cnt** be the number of count of students that don’t fit in a group yet

Solution

- Process the students in descending order of skill
- Let **cnt** be the number of count of students that don't fit in a group yet
- The current student will have lowest skill a_i seen so far
- If ***cnt*** * a_i >= x then make this a team and set ***cnt*** to 0
- Otherwise we cannot make a team, so we want to increment ***cnt*** by 1

Alternative Solution

- Sort the student’s skill level by decreasing order
- Make a new array ***b*** where $\boldsymbol{b_i}$ represents the number of teammates student ***i*** must be paired with to form a valid team

Alternative Solution

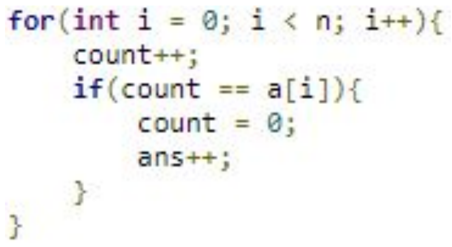
- Sort the student’s skill level by decreasing order
- Make a new array ***b*** where $\boldsymbol{b_i}$ represents the number of teammates student ***i*** must be paired with to form a valid team
- $\boldsymbol{b_i = \mathrm{ceil}(x / a_i)}$
- For example:
- $\boldsymbol{a = [11, 9, 7, 5, 2]}$
- $\boldsymbol{b = [1, 2, 2, 2, 5]}$

Alternative Solution

- a = [11, 9, 7, 5, 2]
- b = [1, 2, 2, 2, 5]
- Now we can go from left to right in ***b*** with the same ***cnt*** variable as before
- If ***cnt*** == b_i then this forms a valid team
    - Increment the answer by one and reset ***cnt*** to 0

Alternative Solution

- *a* = [11, 9, 7, 5, 2]
- *b* = [1, 2, 2, 2, 5]
- Now we can go from left to right in ***b*** with the same ***cnt*** variable as before
- If ***cnt*** == ***b***$_i$ then this forms a valid team
    - Increment the answer by one and reset ***cnt*** to 0

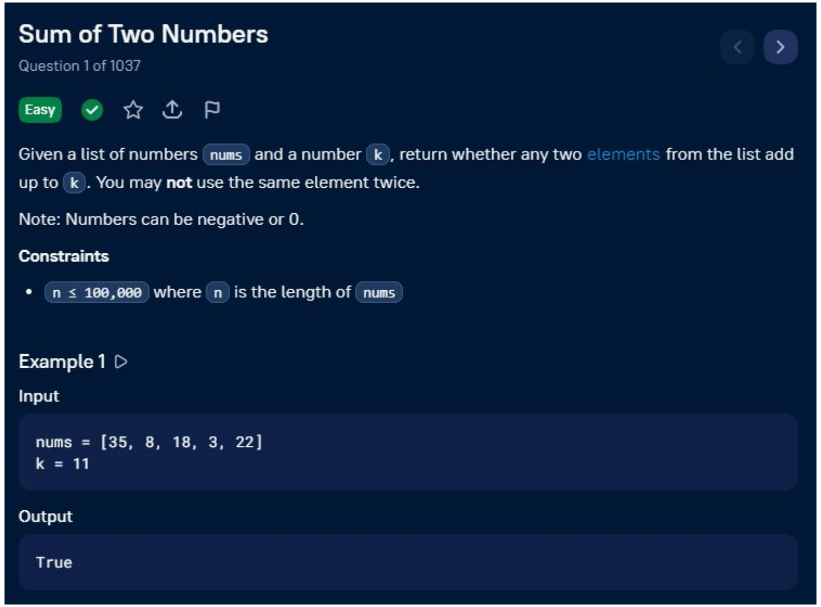


```
for(int i = 0; i < n; i++){
    count++;
    if(count == a[i]){
        count = 0;
        ans++;
    }
}
```

Motivating Problem

1. **Motivating Problem**
2. Two Pointers and Variations
3. Example Problems

2SUM in O(1) Space


Sum of Two Numbers

Question 1 of 1037

Given a list of numbers `nums` and a number `k`, return whether any two elements from the list add up to `k`. You may **not** use the same element twice.

Note: Numbers can be negative or 0.

Constraints

- `n ≤ 100,000` where `n` is the length of `nums`

Example 1

Input

```
nums = [35, 8, 18, 3, 22]
k = 11
```

Output

```
True
```

Brute Force

Iterate through all pairs of values, check if each pair is equal to K.

Brute Force

Iterate through all pairs of values, check if each pair is equal to K.

Time complexity will be $O(N^2)$

Two Pointers

1. Motivating Problem
2. **Two Pointers**
3. Variations
4. Example Problems

Two Pointers

- Main idea: traverse some data structure while keeping track of two pointers
- At each iteration, update one or both pointers, and update some external information
- Often useful in problems requiring greedy algorithms
- Often accompanied with sorting

2SUM in O(1) Space

```
1. Sort input array A
2. Initiate left pointer l at 0 and right pointer r at N - 1
3. While l and r pointers are not pointing at the same index:
    a. If A[l] + A[r] == K
        i. Woohoo! Return true
    b. If A[l] + A[r] < K
        i. Our total is too small, so we increment l
    c. If A[l] + A[r] > K
        i. Our total is too big, so we decrement r
4. Return false
```

2SUM in O(1) Space

O(NlogN) time

O(1) space

Variations

1. Motivating Problem
2. Two Pointers
3. **Variations**
4. Example Problems

Variations

- Pointers going at same/different speeds
- Pointers going in different directions
- Pointers going in the same direction
    - Sometimes called "sliding window"
- Pointers on vertices in a graph
- 3 pointers?

Example Problems

1. Motivating Problem
2. Two Pointers
3. Variations
4. **Example Problems**

Books

When Valera has got some free time, he goes to the library to read some books. Today he's got t free minutes to read. That's why Valera took n books in the library and for each book he estimated the time he is going to need to read it. Let's number the books by integers from 1 to n. Valera needs a_i minutes to read the i-th book.

Valera decided to choose an arbitrary book with number i and read the books one by one, starting from this book. In other words, he will first read book number i, then book number $i+1$, then book number $i+2$ and so on. He continues the process until he either runs out of the free time or finishes reading the n-th book. Valera reads each book up to the end, that is, he doesn't start reading the book if he doesn't have enough free time to finish reading it.

Print the maximum number of books Valera can read.

Input
The first line contains two integers n and t $(1 \leq n \leq 10^5;\ 1 \leq t \leq 10^9)$ — the number of books and the number of free minutes Valera's got. The second line contains a sequence of n integers $a_1, a_2, ..., a_n$ $(1 \leq a_i \leq 10^4)$, where number a_i shows the number of minutes that the boy needs to read the i-th book.

Output
Print a single integer — the maximum number of books Valera can read.

Books Solution

- Maintain two pointers and a sum ***S*** representing the sum the books in our selected range
  - ***L*** represents the left bound of the books we will take
  - ***R*** represents the right bound of the books we will take
  - We will read all books from ***L*** to ***R***
- Both ***L*** and ***R*** will start at the first book

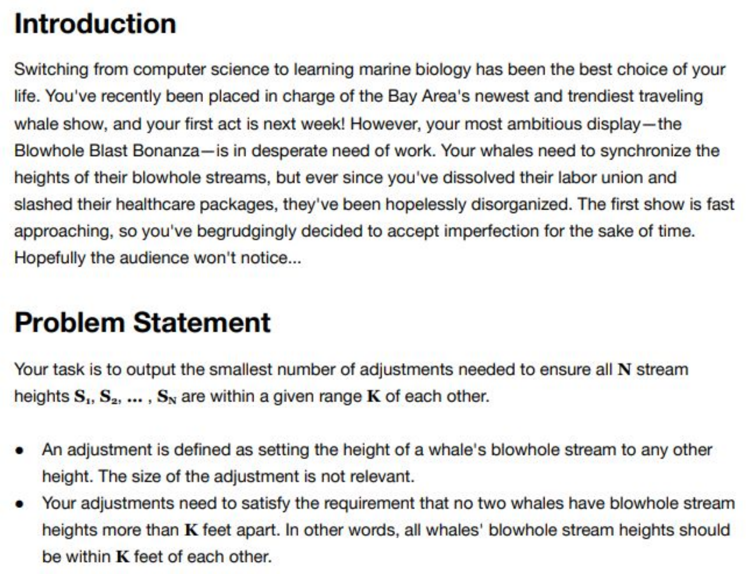
Books Solution

- We will have a loop that moves ***R*** to the right one index at a time
    - At the start of the loop, increment ***R*** and add time to read book ***R*** to our sum ***S***
    - For each position of ***R***, we will calculate the maximum number of books we can read
        - This means we might have to move ***L*** to the right to make the sum of the range less than or equal to ***t*** (the number of free minutes Valera has to read)

Books Solution

- We will have a loop that moves ***R*** to the right one index at a time
  - At the start of the loop, increment ***R*** and add time to read book ***R*** to our sum ***S***
  - For each position of ***R***, we will calculate the maximum number of books we can read
    - This means we might have to move ***L*** to the right to make the sum of the range less than or equal to ***t*** (the number of free minutes Valera has to read)
- After adding book ***R***, it's possible our sum ***S*** is greater than ***t***!
  - This means that we've select too many books and have to move ***L*** to the right
  - While our sum ***S*** is more than the time we have ***t***
    - Subtract time to read book ***L*** from our sum ***S*** (we've removed it) and then increment ***L***

CALICO Editors' Choice Spring '22: Blowhole Blues



Introduction

Switching from computer science to learning marine biology has been the best choice of your life. You've recently been placed in charge of the Bay Area's newest and trendiest traveling whale show, and your first act is next week! However, your most ambitious display—the Blowhole Blast Bonanza—is in desperate need of work. Your whales need to synchronize the heights of their blowhole streams, but ever since you've dissolved their labor union and slashed their healthcare packages, they've been hopelessly disorganized. The first show is fast approaching, so you've begrudgingly decided to accept imperfection for the sake of time. Hopefully the audience won't notice...

Problem Statement

Your task is to output the smallest number of adjustments needed to ensure all $\mathbf{N}$ stream heights $\mathbf{S_1}, \mathbf{S_2}, \dots, \mathbf{S_N}$ are within a given range $\mathbf{K}$ of each other.

- An adjustment is defined as setting the height of a whale's blowhole stream to any other height. The size of the adjustment is not relevant.
- Your adjustments need to satisfy the requirement that no two whales have blowhole stream heights more than $\mathbf{K}$ feet apart. In other words, all whales' blowhole stream heights should be within $\mathbf{K}$ feet of each other.

B. Number of Smaller

time limit per test: 1 second

memory limit per test: 256 megabytes

input: standard input

output: standard output

You are given two arrays, sorted in non-decreasing order. For each element of the second array, find the number of elements in the first array strictly less than it.

Input

The first line contains integers n and m, the sizes of the arrays ($1 \le n, m \le 10^5$). The second line contains n integers a_i, elements of the first array, the third line contains m integers b_i, elements of the second array ($-10^9 \le a_i, b_i \le 10^9$).

Output

Print m numbers, the number of elements of the first array less than each of the elements of the second array.

Example

input

```
6 7
1 6 9 13 18 18
2 3 8 13 15 21 25
```

output

```
1 1 2 3 4 6 6
```

Number of Smaller Solution

- We will first loop through all elements of array ***b*** since we want to construct the answer for each element in ***b***
- Have a pointer ***j*** that holds the index of the first element in ***a*** that is **>=** to our current element b_i

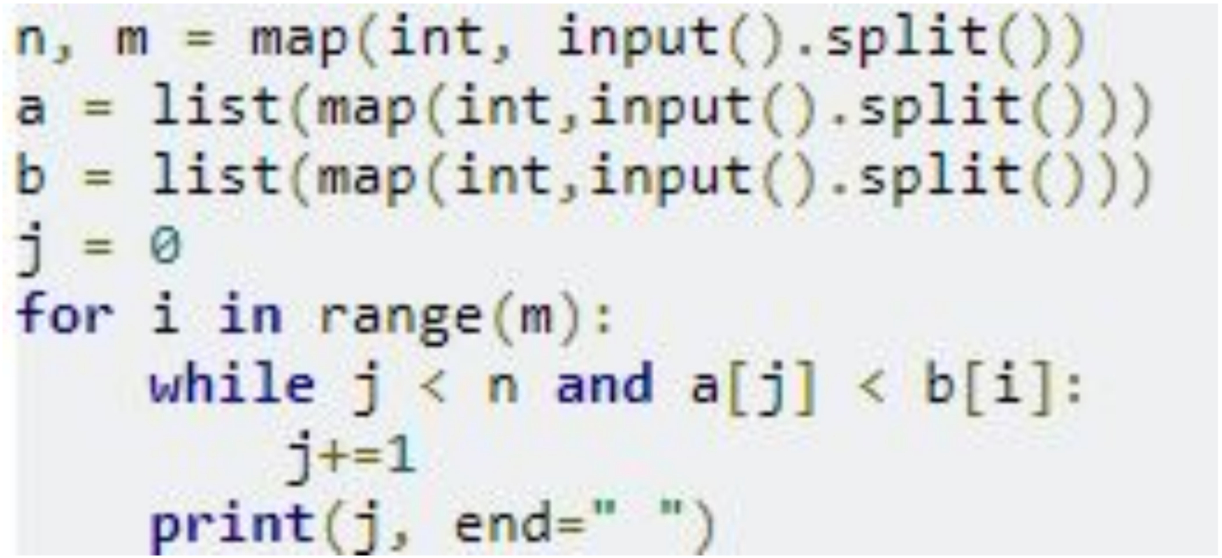
Number of Smaller Solution

- We will first loop through all elements of array ***b*** since we want to construct the answer for each element in ***b***
- Have a pointer ***j*** that holds the index of the first element in ***a*** that is **>=** to our current element b_i
- The value of ***j*** is equal to the number of elements in ***a*** that are less than current element b_i
- Since both of our lists are sorted, whenever we go to the next element element b_i , all ***j*** elements that were less than b_{i-1} will also be less than b_i

Number of Smaller Solution

- We will first loop through all elements of array ***b*** since we want to construct the answer for each element in ***b***
- Have a pointer ***j*** that holds the index of the first element in ***a*** that is **>=** to our current element b_i
- The value of ***j*** is equal to the number of elements in ***a*** that are less than current element b_i
- Since both of our lists are sorted, whenever we go to the next element element b_i , all ***j*** elements that were less than b_{i-1} will also be less than b_i
- So for each b_i
    - While $a_j < b_i$, increment index ***j*** by 1

Number of Smaller Implementation



```python
n, m = map(int, input().split())
a = list(map(int,input().split()))
b = list(map(int,input().split()))
j = 0
for i in range(m):
    while j < n and a[j] < b[i]:
        j+=1
    print(j, end=" ")
```

Check in

Thanks for coming! :)

No class next week!!!