Problem A. Hello, world!

Problem Description

The classic task to start a programming course.

Given an integer n:

- If $n = 1$, output `Hello, world!`.
- Otherwise, print the public email you might need to contact the TAs.

Input Format

- line 1: n

Output Format

- line 1: ans

Constraints

- $1 \leq n \leq 2$.
- All the inputs are integers.

Subtasks

1. (50 points) $n = 1$.
2. (50 points) $n = 2$.

For testing purpose, you can get half of the subtask points even if you print the wrong answer.

No.	Testdata Range	Time Limit (ms)	Memory Limit (KiB)
1	1	1000	262144
2	2	1000	262144

Samples

Sample Input 1

```
1
```

Sample Output 1

```
Hello, world!
```

Sample Input 2

```
2
```

Sample Output 2

```
abcdefghijkl@gmail.com
```

Please note that this output is wrong, you will need to find the email yourself.

Problem B. Proficient in I2A

Problem Description

A straightforward path to becoming proficient in "Introduction to Algorithms" is to become familiar with the STL (*Standard Template Library*).

Let's attempt to solve some simple tasks using the STL (although, it's entirely up to you whether or not to use it)!

Task 1

Given an integer array a of length n.

Output n numbers: the array a sorted from smallest to largest.

- Keywords: `std::sort`

Task 2

Given an integer array a of length n.

Output n numbers: the array a in reverse order.

- Keywords: `std::reverse`

Task 3

Given an integer array a of length n, while there are two adjacent elements that are the same, remove one of them.

Output the resulting array when there are no longer any adjacent elements that are the same.

- Keywords: `std::unique`

Task 4

Given an integer array a of length n, let array $c = (c[0], c[1], \ldots, c[n-1])$, where $c[i] = \sum_{k=0}^{i} a[k]$ be the prefix sum of a.

Output n numbers: the array c.

- Keywords: `std::partial_sum`

Task 5

Given an integer array a of length n.

Output 2 numbers: the 0-based index of the **first** minimum element and the **last** maximum element in the array a.

- Keywords: `std::min_element`, `std::max_element`

Task 6

Given an integer n.

Output all permutations of the first n lowercase Latin letters from lexicographically **largest** to **smallest**.

- Keywords: `std::next_permutation`, `std::prev_permutation`

Task 7

Given two **sorted** integer arrays a and b of length n, let array $c = (a[0], a[1], \ldots, a[n-1], b[0], b[1], \ldots, b[n-1])$ be the concatenation of a and b.

Output $2n$ numbers: the array c sorted from smallest to largest.

- Keywords: `std::merge`

Task 8

Given two integer arrays a and b of length n, process n queries.

We use $b[i]$ to denote the i^{th} query. Please output the smallest element in a that is larger than $b[i]$, output 0 if there is no element in a that is larger than $b[i]$.

- Keywords: `std::set`, `std::lower_bound`, `std::upper_bound`

Task 9

Given an integer array a of length n.

Output n numbers: the occurrences of $a[0], a[1], \ldots, a[n-1]$ in the array a.

- Keywords: `std::map`

Task 10

Process n queries to maintain a multiset (initially empty).

We use $a[i]$ to denote the i^{th} query. If $a[i] = 0$ and the multiset is non-empty, output and remove any smallest integer from the multiset, otherwise, insert $a[i]$ into the multiset.

- Keywords: `std::priority_queue`, `std::multiset`

Input Format

The first line of input contains a single integer T --- the number of test cases. The description of test cases follows.

The first line of each test case contains 2 integers op, n, denoting the task you should solve, and the length of the array (which is also the number of queries for $op \in \{8, 10\}$).

If $op \neq 6$, the second line of each test case contains n integers $a[0], a[1], \ldots, a[n-1]$.

If $op \in \{7, 8\}$, the third line of each test case contains n integers $b[0], b[1], \ldots, b[n-1]$.

Output Format

For each test case, output everything in a single line, with spaces in between.

Constraints

- $1 \leq T \leq 100$.
- $1 \leq op \leq 10$.
- $1 \leq n \leq 10\,000$.
- $n \leq 6$ if $op = 6$.
- $0 \leq a[i], b[i] \leq 100\,000$ for $i = 0, 1, \ldots, n-1$.
- $a[0] \leq a[1] \leq \cdots \leq a[n-1]$ and $b[0] \leq b[1] \leq \cdots \leq b[n-1]$ if $op = 7$.
- All the inputs are integers.

Subtasks

1. (5 points) $op = 1$.
2. (5 points) $op = 2$.
3. (5 points) $op = 3$.
4. (5 points) $op = 4$.
5. (5 points) $op = 5$.
6. (5 points) $op = 6$.
7. (5 points) $op = 7$.
8. (15 points) $op = 8$.
9. (15 points) $op = 9$.
10. (15 points) $op = 10$.
11. (20 points) No additional constraints.

No.	Testdata Range	Time Limit (ms)	Memory Limit (KiB)
Samples	1	1000	262144
1	2	1000	262144
2	3	1000	262144
3	4	1000	262144
4	5	1000	262144
5	6	1000	262144
6	7	1000	262144
7	8	1000	262144
8	9	1000	262144
9	10	1000	262144
10	11	1000	262144
11	1-12	1000	262144

Samples

Sample Input 1

```
10
1 5
4 8 7 6 3
2 5
48 76 34 87 63
3 10
2 4 4 2 5 5 5 5 55 1
4 5
4 8 7 6 3
5 10
4 8 7 6 3 4 8 7 6 3
6 3
7 5
3 4 6 7 8
0 2 5 6 9
8 5
4 8 7 6 3
3 1 4 7 9
9 10
2 7 1 8 2 8 1 8 2 8
10 10
3 0 0 7 2 0 7 0 0 0
```

Sample Output 1

```
3 4 6 7 8
63 87 34 76 48
2 4 2 5 55 1
4 12 19 25 28
4 6
cba cab bca bac acb abc
0 2 3 4 5 6 6 7 8 9
4 3 6 8 0
3 1 2 4 3 4 2 4 3 4
3 0 2 7 7
```

References

- https://en.cppreference.com/w/
- https://cplusplus.com/

Problem C. Income Tax Rate

Problem Description

Everyone dreams of making money, but although making money and having a steady income is fun, paying taxes is not fun. We live in a country (or a region 😉, based on your political preference) that adopts a progressive tax system, where the tax rate increases as the taxable amount increases.

The table below shows the tax rate ranges (tax brackets) in Taiwan:

綜合所得淨額	乘法	稅率
0~540,000	×	5%
540,001~1,210,000	×	12%
1,210,001~2,420,000	×	20%
2,420,001~4,530,000	×	30%
4,530,001~以上	×	40%

From National Taxation Bureau of Taipei, Ministry of Taiwan

The first column indicates the taxable income, where the third column shows the tax rates of the respective income range. For example, if you have a total income of NTD $\$600\,000$, then you'll have to pay $\$540\,000 \times 5\% + (\$600\,000 - \$540\,000) \times 12\% = \$34\,200$ in total.

Given the tax brackets of a country, can you calculate how much tax a person should pay according to one's income?

Input Format

On the first line there is an integer n, indicating how many ranges there are.

n lines follow, on each line there are two numbers a_i, r_i, where a_i is an integer indicating the least value of this range and r_i is a floating point number indicating the tax rate of this bracket.

On the next line, there is one integer m indicating the number of people.

The next m lines each gives one integer c_i, which is the income of the i^{th} person.

Output Format

Print one number in each line, the i^{th} number denotes the tax one should pay according to income c_i.

The answer will be considered correct if the relative or absolute error does not exceed $\epsilon = 10^{-6}$. (See the section below for further notes.)

Constraints

- $1 \le n \le 1000$.
- $0 \le a_1 < a_2 < \cdots < a_n \le 10^9$.
- $0 < r_1 < r_2 < \cdots < r_n < 1$.
- $1 \le m \le 10\,000$.
- $0 \le c_i \le 10^9$ for $i = 1, 2, \ldots, m$.
- All the inputs except r_i are integers.

Subtasks

1. (100 points) No additional constraints.

No.	Testdata Range	Time Limit (ms)	Memory Limit (KiB)
Samples	1-2	1000	262144
1	1-7	1000	262144

Samples

Sample Input 1

```
5
1 0.05
540001 0.12
1210001 0.2
2420001 0.3
4530001 0.4
3
1
600000
426410
```

Sample Output 1

```
0.05
34200.00
21320.50
```

For $c_1 = 1$, the income tax needed to be paid is $\$0.05$ since $0.05 = (1 - 0) \times 0.05$ is the least value in the first range.

For $c_2 = 600\,000$, the amount of tax paid is $\$34\,200$ since
$(540\,000 - 0) \times 0.05 + (600\,000 - 540\,000) \times 0.12 = 34\,200$.

Sample Input 2

```
3
1 0.4
101 0.45
501 0.55
3
1
10
100
```

Sample Output 2

```
0.4
4.0
40.0
```

Notes on Floating Points

As we all know, computers use floating point variables to store fractions and irrational numbers, and truncated or rounded errors occur due to the limited bits a variable can use to represent a value.These errors accumulate over multiple floating point calculations, with different methods or even just the order of calculation leading to different accuracies.

How to approximate the answer to it's closest is an interesting problem, but the task is often addressed in numerical analysis. In our course, we focus on getting the algorithm concepts right, so we tolerate some errors ϵ between the answer a and your output x. To do so, we use both **absolute** and **relative** errors to judge your answer.

The former calculates the difference $|x - a|$, so if your output is close enough to the judge's answer, it will be considered correct.

The latter divides the absolute error by the magnitude of the answer, which is $\left|\frac{x-a}{a}\right|$. This is often used when the original answer is a number too large to reach a decent absolute error, thus the answer will also be accepted if it differs a tiny ratio to the judge's answer.

For example, let's say the answer $a = 10\,000$ and the error tolerance is $\epsilon = 10^{-6}$. Considering the absolute value, all x satisfying $10\,000 - 10^{-6} \leq x \leq 10\,000 + 10^{-6}$ are accepted. From the relative error's perspective, all x meeting the conditions $\left|\frac{x-10\,000}{10\,000}\right| \leq 10^{-6}$ are taken as correct answers, lengthening the tolerance range to $[10\,000 - 10^{-2}, 10\,000 + 10^{-2}]$.

In short, if we tolerate an error of ϵ, then your output will be judged as correct if $\frac{|a-b|}{\max\{|a|,|b|,1\}} \leq \epsilon$.

Problem D. Withdraw

Problem Description

You signed up for the Introduction to Algorithms course this semester, and found yourself stuck on Lab 0 (yep, the pre-course basics). Discouraged, you made up your mind on Week 3 of the semester that this is the course to withdraw. With the forms filled, you came the professor's door for the last signature. Unsurprisingly, many of your classmates are there too, queuing up to quit the course.

Let's switch to the perspective of the frustrated professor. He knew the TAs "accidentally" made Lab 0 too hard, but sees the potential in these students and tried to encourage them to stay on. For each student, the professor has a potential value p_i. As long as the potential value is positive, the professor will ask the student to rethink about it. But once the student is out of the room, the nightmare of Lab 0 shadows his mind once again, thus he rejoins the queue immediately. Thus in the end, every student that came to queue never left it with the form unsigned.

Of course the professor has other researches to work on, thus he can't do this all day. He still rejects students' withdrawal if he thinks they're still capable, but the potential value decreases by one on each time of ask. On the occasion the potential value turns zero, he finally gets fed up and signs the form for that student.

To remind himself of how many occasions each student came asking, the professor asked the TAs to keep a long record of the order each student came to withdraw from the course. The TAs are indeed annoying, but this time it is them annoyed. Given the initial order, IDs and potential values of each student, can you help the TAs get the overall sequence from the first ask to the last sign?

Input Format

On the first line there is an integer n indicating the number of students that chose to drop out of the course. n lines follow, each with two numbers, the student id id_i and their initial potential p_i. The n lines are listed in the initial order of the queue.

Output Format

Output one number on each line, each with an ID of the student that the professor met. Note that the numbers must be printed in time order, thus the first ID indicates the first student the professor met, and the last student the professor met is printed on the last line.

Constraints

- $1 \leq n \leq 100\,000$.
- $1 \leq id_i \leq 10^9$ for $i = 1, 2, \ldots, n$.
- $id_i \neq id_j$ for $i \neq j$.
- $1 \leq p_i \leq 100\,000$ for $i = 1, 2, \ldots, n$.
- $\sum_{i=1}^{n} p_i \leq 10\,000\,000$.
- All the inputs are integers.

Subtasks

1. (100 points) No additional constraints.

No.	Testdata Range	Time Limit (ms)	Memory Limit (KiB)
Samples	1	2000	262144
1	1-15	2000	262144

Samples

Sample Input 1

```
5
3 2
2 2
5 1
1 3
4 2
```

Sample Output 1

```
3
2
5
1
4
3
2
1
4
1
```

Everyone visited the professor in their first round, but student 5 left the queue with his formed signed. The remaining 4 students queued for a second time, and all except student 1 left the queue afterwards. Student 1 signed his withdrawal form after the third round, thus the queue is emptied.

Hint

On output, avoid using `cout << endl;` as it may exceed time limits. `cout << '\n'` or `printf('\n')` are execution-wise faster options.

Problem E. Queuing

Problem Description

Little T loves queuing! Whether it's for enjoying a bowl of ramen, riding amusement park attractions, or exploring a comic convention, you can always find Little T in the midst of a queue. However, this time, Little T isn't queuing for leisure; he's preparing for the future of humanity.

After three days and nights of observation, Little T noticed that the people in line in front of the pharmacy are remarkably patient. They won't leave the queue until their turn comes. Little T has identified three possible scenarios:

1. The pharmacy provides each person in the queue with k masks.
2. The first x people at the front of the queue leave content.
3. y more people join the back of the queue.

Little T wants to know how many masks the person with the most masks in the queue has after each of these scenarios occurs. So, he has provided you with this data and hopes you can write a program to calculate the results.

Input Format

The first line contains a positive integer q, representing the number of observed scenarios by Little T.

The following q lines each start with a positive integer op_i, followed by some parameters indicating the upcoming scenario. Each scenario can be one of the following three:

- `1` k: Everyone in the line gets k masks.
- `2` x: x people in the queue leave content.
- `3` y: Another y people join the queue.

Output Format

For each scenario, please output an integer representing how many masks the person with the most masks in the queue has right now.

Constraints

- $1 \leq q \leq 500\,000$.
- $op_i \in \{1, 2, 3\}$ for $i = 1, 2, \ldots, q$.
- $1 \leq k, x, y \leq 10^6$ for each scenario.
- It is guaranteed that the number of people in the queue will never be negative at any moment, and if the queue is empty, there will be no mask distribution operations.
- All the inputs are integers.

Subtasks

1. (20 points) $q \leq 1000$; $x = y = 1$.
2. (50 points) $q \leq 1000$.
3. (30 points) No additional constraints.

No.	Testdata Range	Time Limit (ms)	Memory Limit (KiB)
Samples	1-3	1000	262144
1	4-12	1000	262144
2	1-18	1000	262144
3	1-26	1000	262144

Samples

Sample Input 1

```
3
3 5
1 4
1 9
```

This sample input satisfies the constraints of Subtasks 2, 3.

Sample Output 1

```
0
4
13
```

Sample Input 2

```
7
3 1
1 6
3 1
3 1
1 2
2 1
1 1
```

This sample input satisfies the constraints of Subtasks 1, 2, 3.

Sample Output 2

```
0
6
6
6
8
2
3
```

Operation	Queue
3 1	$[0]$
1 6	$[6]$
3 1	$[6, 0]$
3 1	$[6, 0, 0]$
1 2	$[8, 2, 2]$
2 1	$[2, 2]$
1 1	$[3, 3]$

Sample Input 3

```
8
3 1000000
1 1000000
2 999999
3 1000000
1 1000000
2 999999
3 1000000
1 1000000
```

This sample input satisfies the constraints of Subtasks 2, 3.

Sample Output 3

```
0
1000000
1000000
1000000
2000000
1000000
1000000
2000000
```

Problem F. Postfix

Problem Description

Postfix notation is a mathematical notation in which operators follow their operands. It does not need any parentheses to indicate the order of operations when evaluating the values of the expression. It is found that using postfix notation may lead to faster calculations and can save more memory compared to other mathematical notations during evaluation.

Now, given a postfix expression, please write a program to calculate the value of it. Since the answer might be very large, please output the answer modulo $(10^9 + 7)$.

Input Format

Each input will contain one mathematical expression presented in the postfix notation. Each integer x in the expression would be a 32-bit integer, while operators op will be one of $\{+, *\}$. Two consecutive operands or operators will be seprated by a single space.

Output Format

Output an integer, the result of the expression.

Constraints

- $1 \leq x \leq 2^{31} - 1$.
- $op \in \{+, *\}$.
- All the numbers in inputs are integers.

Subtasks

1. (100 points) No additional constraints.

No.	Testdata Range	Time Limit (ms)	Memory Limit (KiB)
0	1	1000	262144
1	1-8	1000	262144

Samples

Sample Input 1

```
3 8 + 5 6 + *
```

Sample Output 1

```
121
```

The postfix expression can be solved by parsing the expression from the left, and do the calculation once we found two consequtive integers follow by a operator.

Takes this testcase for example. Starting from the left-hand side, first, we found `3 8 +`, which match the condition showing above, we do `3 + 8` and replace it back into the equation, then the equation become `11 5 6 + *`.

Keep doing the procedure that mentioned above, we can get the correct answer as showing below:

```
  3 8 + 5 6 + *
→ 11 5 6 + *
→ 11 11 *
→ 121
```

Sample Input 2

```
2147483647 2147483647 *
```

Sample Output 2

```
850618742
```

Don't forget to output the answer modulo $(10^9 + 7)$.

Problem G. Jujutsu Kaisen 呪術廻戦

Problem Description

無量空處

Recently, deadly curses have caused numerous tragedies, and many innocent people have been hurt and killed. Today, Yuji, a student of Jujutsu High School (呪術高専), has been sent to eliminate the vicious curses that are lingering on a street to rescue the survivors. However, Yuji doesn't know how to use any jujutsu (呪術) yet, so he decided to seek Satoru Gojō for help. After thinking for a while, Satoru asks Yuji to put jubutsus (呪物s) on several **distinct** positions to attract curses so that he can kill them all at once.

However, to prevent Ryomen Sukuna, the most notorious curse that is currently hiding in Yuji's body from getting out of control, **in the beginning**, Yuji will place a jubutsu at the **begin** and the **end** of the street respectively. Meanwhile, every time Yuji places a jubutsu, Satoru will summon a new shikigami (式神) as well. In order to maximize the protection, each shikigami has to be placed as far as possible to its closest jubutsu.

Now, given the length of the road and the target position of the jubutsu to be placed, figure out the location that the shikigami should be put and output the maximum distance between the newly summoned shikigami and its closest jubutsu after each placement of the jubutsu. In this problem, you can treat the road as a 1-dimension line of length L with the position numbered $0, 1, 2, 3, \ldots, L$ in this problem. Also the begin and the end of the road means position 0 and position L, individually.

In case of confusing, the distance between shikigami and jubutsu on the postion p_1 and p_2 is defined as $|p_1 - p_2|$.

Input Format

The first line contains two integers L and n, which stand for the length of the street and the

number of jubutsu.

The second line contain n integers $p_1, p_2, \ldots, p_n$, where p_i $(1 \leq i \leq n)$ denotes the position of the i^{th} jubutsu being placed.

Output Format

Print the the longest distance of the newly summoned shikigami to its closest jubutsu in a single line every time Satoru add a shikigami.

Constraints

- $1 \leq L \leq 10^9$.
- $1 \leq n \leq 200\,000$.
- $1 \leq p_i \leq L - 1$ for $i = 1, 2, \ldots, n$.
- $p_i \neq p_j$ for $i \neq j$.
- All the inputs are integers.

Subtasks

1. (100 points) No additional constraints.

No.	Testdata Range	Time Limit (ms)	Memory Limit (KiB)
Samples	1	1000	262144
1	1-8	1000	262144

Samples

Sample Input 1

```
8 3
6 3 1
```

Sample Output 1

```
3
1
1
```

In the sample input, at the beginning, two jubutsu are placed at 0 and 8.

First, Yuji places a jubutsu at position 6, so the positions of the placed jubutsus are $\{0, 6, 8\}$. The position that Satoru can place a shikigami at is 3, with this placement, the distance between the new shikigami and the nearest jubutsu will be $|3 - 0| = |6 - 3| = 3$.

Then, Yuji places a jubutsu at position 3, so the positions with jubutsu are $\{0, 3, 6, 8\}$. This time, the positions that Satoru can place shikigamis are $\{1, 2, 4, 5, 7\}$, with these placements, the distance between the new shikigami and the nearest jubutsu will be 1. Take the placement 2 for example, $|2 - 0| = 2$ and $|3 - 2| = 1$, and $\min(2, 1) = 1$.

Problem H. Classical Data Structure Problem

Problem Description

You are given a set $S = \{a[0], a[1], \ldots, a[n-1]\}$, and you will have to support q operations of three types on this set:

1. Remove the smallest integer from S. It is guaranteed that $S \neq \emptyset$.
2. Remove the largest integer from S. It is guaranteed that $S \neq \emptyset$.
3. Insert $\mathsf{MEX}(S)$ into S.

Define $\mathsf{MEX}(S)$ as the smallest non-negative integer that has not appeared in S.

Let $ans[i]$ be the i^{th} removed or inserted integer, please output $ans[0], ans[1], \ldots, ans[q-1]$.

Input Format

- line 1: $n \;\; q$
- line 2: $a[0] \;\; a[1] \;\; \cdots \;\; a[n-1]$
- line 3: $op[0] \;\; op[1] \;\; \cdots \;\; op[q-1]$

Output Format

- line $1+i$ $(0 \leq i \leq q-1)$: $ans[i]$

Constraints

- $0 \leq n \leq 200\,000$.
- $1 \leq q \leq 1\,000\,000$.
- $0 \leq a[i] \leq 10^9$ for $i = 0, 1, \ldots, n-1$.
- $a[i] \neq a[j]$ for $i \neq j$.
- $1 \leq op[i] \leq 3$ for $i = 0, 1, \ldots, q-1$.
- All the inputs are integers.

Subtasks

1. (15 points) $n \leq 1000$; $q \leq 1000$.
2. (15 points) $op[i] \in \{1, 2\}$ for $i = 0, 1, \ldots, q-1$.
3. (30 points) $op[i] \in \{2, 3\}$ for $i = 0, 1, \ldots, q-1$.
4. (30 points) $n = 0$.
5. (10 points) No additional constraints.

No.	Testdata Range	Time Limit (ms)	Memory Limit (KiB)
Samples	1-4	2000	262144
1	1-10	2000	262144
2	11-16	2000	262144
3	17-23	2000	262144
4	24-30	2000	262144
5	1-36	2000	262144

Samples

Sample Input 1

```
5 7
4 8 7 6 3
1 3 2 3 1 3 2
```

This sample input satisfies the constraints of Subtasks 1, 5.

Sample Output 1

```
3
0
8
1
0
0
7
```

Sample Input 2

```
5 5
4 2 0 6 9
1 2 2 1 2
```

This sample input satisfies the constraints of Subtasks 1, 2, 5.

Sample Output 2

```
0
9
6
2
4
```

Sample Input 3

```
6 8
31 4 15 9 26 1000000000
3 2 2 3 3 3 2 3
```

This sample input satisfies the constraints of Subtasks 1, 3, 5.

Sample Output 3

```
0
1000000000
31
1
2
3
26
5
```

Sample Input 4

```
0 4

3 2 3 1
```

This sample input satisfies the constraints of Subtasks 1, 4, 5.

Sample Output 4

```
0
0
0
0
```

Problem I. Classical Data Structure Problem

Problem Description

You are given a set $S = \{a[0], a[1], \ldots, a[n-1]\}$, and you will have to support q operations of three types on this set:

1. Remove the smallest integer from S. It is guaranteed that $S \neq \emptyset$.
2. Remove the largest integer from S. It is guaranteed that $S \neq \emptyset$.
3. Insert $\mathsf{MEX}(S)$ into S.

Define $\mathsf{MEX}(S)$ as the smallest non-negative integer that has not appeared in S.

Implementation Details

You should implement the following procedures:

```
void init(int n, int[] a)
```

- n: the initial size of the set S.
- a: arrays of length n. For $0 \leq i \leq n-1$:
  - $a[i]$ is the i^{th} element in S.
- This procedure is called exactly once, before any calls to `remove_min`, `remove_max`, and `insert_mex` (see below).

```
int remove_min()
```

- This procedure should return the minimum integer in S, $\mathsf{MIN}(S)$.
- It is guaranteed that $S \neq \emptyset$ when calling this function.
- After calling this function, $\mathsf{MIN}(S)$ should be removed from set S.

```
int remove_max()
```

- This procedure should return the maximum integer in S, $\mathsf{MAX}(S)$.
- It is guaranteed that $S \neq \emptyset$ when calling this function.
- After calling this function, $\mathsf{MAX}(S)$ should be removed from set S.

```
int insert_mex()
```

- This procedure should return the minimum non-negative integer not in S, $\mathsf{MEX}(S)$.
- After calling this function, you should insert $\mathsf{MEX}(S)$ into set S.

- The total number of calls to `remove_min`, `remove_max`, and `insert_mex` is exactly q.

Constraints

- $0 \le n \le 200\,000$.
- $1 \le q \le 10\,000\,000$.
- $0 \le a[i] \le 10^9$ for $i = 0, 1, \ldots, n-1$.
- $a[i] \ne a[j]$ for $i \ne j$.

Subtasks

1. (1 point) $q \le 5000$.
2. (9 points) $q \le 1\,000\,000$.
3. (20 points) $q \le 3\,000\,000$.
4. (70 points) No additional constraints.

No.	Testdata Range	Time Limit (ms)	Memory Limit (KiB)
1	1-7	1000	262144
2	1-14	1000	262144
3	1-21	500	262144
4	1-28	500	65536

Examples

Consider the following call:

```
init(5, [4, 8, 7, 6, 3])
```

The initial set $S = \{4, 8, 7, 6, 3\}$.

Let's say $q = 7$, and the grader calls the following functions:

Function Call	S	Return Value
`remove_min()`	$\{4, 8, 7, 6\}$	3
`insert_mex()`	$\{4, 8, 7, 6, 0\}$	0
`remove_max()`	$\{4, 7, 6, 0\}$	8
`insert_mex()`	$\{4, 7, 6, 0, 1\}$	1
`remove_min()`	$\{4, 7, 6, 1\}$	0
`insert_mex()`	$\{4, 7, 6, 1, 0\}$	0
`remove_max()`	$\{4, 6, 1, 0\}$	7

As such, the procedure should return 3, 0, 8, 1, 0, 0, 7.

Sample grader

The sample grader reads the input in the following format:

- line 1: n q
- line 2: $a[0]$ $a[1]$ $\cdots$ $a[n-1]$
- line 3: $op[0]$ $op[1]$ $\cdots$ $op[q-1]$

Where $op[i] \in \{1, 2, 3\}$ ($0 \le i \le q-1$) denotes a call to `remove_min`, `remove_max`, or `insert_mex` respectively.

The sample grader prints your answers in the following format:

- line $1+i$ ($0 \le i \le q-1$): the i^{th} called function name, and the return value of the call.

Notes

- Here is a sample implementation. (Link)
- You should include `"1608.h"` in your program.
- You should **NOT** implement the `main` function.
- You should only submit `1608.cpp` to the Online Judge.
- You should **NOT** read anything from `stdin` or print anything to `stdout`.
- You can use `stderr` for debug. (`std::cerr`)
- You can use `g++ -std=c++17 -O2 -o 1608 1608.cpp grader.cpp` to compile the code, and use `./1608` or `1608.exe` to run the code.

Conventions

The task statements specify signatures using generic type names `void`, `string`, `int`, `int[]` (array), and `bool[][]` (2D array).

In each of the supported programming languages, the graders use appropriate data types or implementations, as listed below

Language	**void**	**string**	**int**	**int[]**	length of array **a**
C++	`void`	`std::string`	`int`	`std::vector<int>`	`a.size()`

A 2D array is a non-empty array of arrays of the same length.

Language	**bool[][]**	#rows in 2D array **a**	#columns in 2D array **a**
C++	`std::vector<std::vector<bool>>`	`a.size()`	`a[0].size()`