Problem A. Min Stack

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 MB

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the `MinStack` class:

- `MinStack()` initializes the stack object. This is the class constructor.
- `void push(int val)` pushes the element *val* onto the stack.
- `void pop()` removes the element on the top of the stack.
- `int top()` gets the top element of the stack.
- `int getMin()` retrieves the minimum element in the stack.

You must implement a solution with $O(1)$ time complexity for each function. Solution of higher time complexity will not be considered.

Allowed Language(s): C++

Allowed Header(s): `iostream`
You should not use any other header files.

Input

The first line of input contains n ($1 \leq n \leq 10^6$) — the number of elements initially present in the `MinStack`.

The second line of input contains n space separated integers — the elements initially present in the `MinStack`, from bottom to top.

The third line of input contains m ($1 \leq m \leq 10^6$) — the number of operations that you have to perform on the `MinStack`.

The following m lines contain one operation each. An operation can be one of the following:

- 1 x: push x onto the stack
- 2: pop the the top element of the stack
- 3: print the top element of the stack
- 4: print the minimum element in the stack

It is guaranteed that the stack will be non-empty whenever operations 2, 3 or 4 is asked to be performed. If multiple elements in the stack have the minimum value, print the minimum value only once for operation 4.

Output

For every operation of type 3 or 4, print the required element on a new line.

Example

standard input	standard output
5	2
5 3 -6 2 3	-6
8	3
2	0
3	
4	
2	
2	
4	
1 0	
4	

Problem B. Ikea Bad

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 MB

Your hostel assistant, Ikea, has no work and too much time on his hands. So, he decided to come up with a new hostel rule. Your hostel has n rooms. They are numbered from 1 to n such that when going in the clockwise order, room number $(i + 1)$ comes immediately after room number i for $(1 \leq i < n)$ and room number 1 comes immediately after room number n. Every room is assigned to exactly one student.

As a part of the new rule introduced by Ikea, he has assigned an integer to every student. To leave the college premises, a student has to get an outstation pass from another student who has been assigned an integer strictly greater than him. A previous rule introduced by Ikea forbids the students from traveling in an anti-clockwise manner when going from one room to another.

You are given an array a of n integers, where a_i is the integer assigned to the student in room number i. Determine the minimum number of rooms every student will have to cross in order to get his outstation pass. The numbers assigned to the students need not be unique. Also, a student can give passes to multiple students.

Assume 1-based indexing.

The time complexity of your solution should be $O(n)$. Solution of higher time complexity will not be considered.

Allowed Language(s): C, C++

Allowed Header(s): `iostream`, `stack`

Input

The first line of input contains n $(1 \leq n \leq 10^6)$ — the number of rooms in your hostel.

The second line of input contains n space separated integers $a_1, a_2, \ldots, a_n$ $(1 \leq a_i \leq 10^6)$ — the integers assigned to the students.

Output

Print n space separated integers, the i-th integer should be the minimum number of rooms that the student living in room number i will have to cross in order to get an outstation pass. If a student cannot get his pass from any other student, print -1.

Examples

standard input	standard output
5 1 -2 3 4 3	2 1 1 -1 4

Explanation

In the given example,

- Student in room 1 can get his pass from the student in room 3. He will have to cross 2 rooms, room 2 and room 3.
- Student in room 2 can get his pass from the student in room 3. He will have to cross 1 room, room 3.
- Student in room 3 can get his pass from the student in room 4. He will have to cross 1 room, room 4.

- Student in room 4 cannot get his pass from any other student.
- Student in room 5 can get his pass from the student in room 4. He will have to cross 4 rooms, room 1, room 2, room 3 and room 4.

In all of the above cases, it can be shown that the students can not get their pass by crossing fewer than the mentioned number of rooms.

Problem C. How did the exam go?

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 MB

Ram Prasad is teaching a course for the first time this semester. He recently received the histogram of marks scored by the students in the mid-semester exam of his course. The marks range from 1 to n, with a_i students scoring exactly i marks. Ram Prasad does not want to share the statistics of the exam with the students but since he has to share something, he decided to share the area of the largest rectangle in the histogram.

Given an array a of n integers, where a_i is the number of students scoring i marks, find the area of the largest rectangle in the histogram for Ram Prasad. Take the width of each bar in the histogram as 1 unit and height of the i-th bar as a_i units for your calculations. Note that it is not necessary that each and every marks is obtained by some student(s).

Assume 1-based indexing.

The time complexity of your solution should be $O(n)$. Solution of higher time complexity will not be considered.

Allowed Language(s): C, C++

Allowed Header(s): `iostream`, `stack`

Input

The first line of input contains n ($1 \leq n \leq 10^6$).

The second line of input contains n space separated integers $a_1, a_2, \ldots, a_n$ ($0 \leq a_i \leq 10^9$).

Output

Print one line containing the area of the largest rectangle in the histogram.

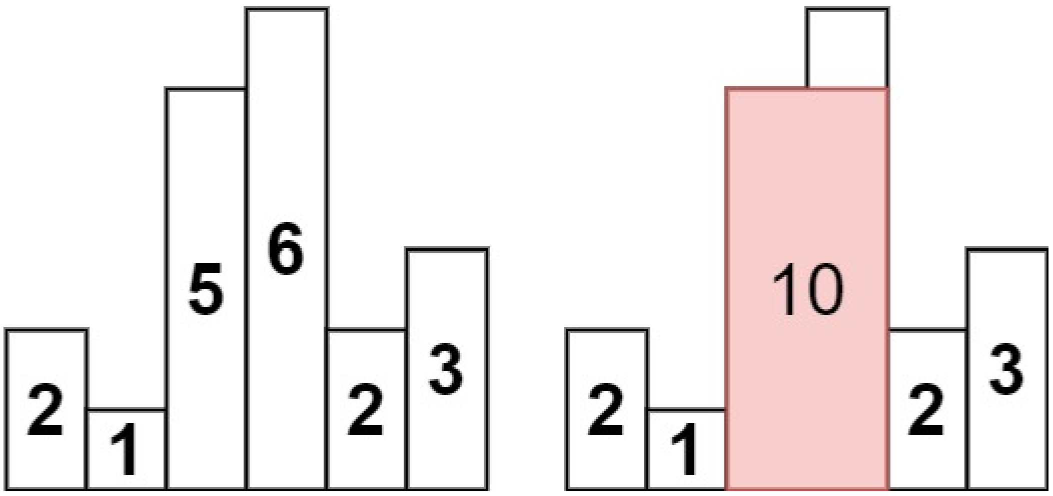
Examples

standard input	standard output

`6`
`2 1 5 6 2 3`
	`10`

Explanation

The following figure illustrates the histogram given in the example:



Problem D. First non-repeating integer

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 MB

Given an array a of n non-negative integers, print the first non-repeating integer in every prefix of a. A prefix of an array is any non-empty, contiguous part of the array starting with the first element of the array.

Assume 1-based indexing.

The time complexity of your solution should be $O(n)$. Solution of higher time complexity will not be considered.

Allowed Language(s): C

Input

The first line of input contains n ($1 \leq n \leq 10^6$).

The second line of input contains n space separated integers $a_1, a_2, \ldots, a_n$ ($0 \leq a_i < n$).

Output

Print n space separated integers, the i-th integer should be the first non-repeating integer in the prefix of a of length i. If there is no non-repeating integer in a prefix, print -1.

Examples

standard input	standard output

`7`
`4 6 4 5 5 6 0`
	`4 4 6 6 6 -1 0`

Explanation

Prefix	**First non-repeating integer**
4	4
4 6	4
4 6 4	6
4 6 4 5	6
4 6 4 5 5	6
4 6 4 5 5 6	-1
4 6 4 5 5 6 0	0

Problem E. Sliding Window Maximum

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 MB

You are given an array a of n integers. There is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. In every iteration, the sliding window moves right by one position.

Print the maximum element in the sliding window in every iteration. The elements in the sliding window need not be unique.

The overall time complexity of your solution should be $O(n)$. Solution of higher time complexity will not be considered.

Allowed Language(s): C, C++

Allowed Header(s): `iostream`, `deque`

Input

The first line of input contains two integers, n and k ($1 \leq k \leq n \leq 10^6$).

The second line of input contains n space separated integers $a_1, a_2, \ldots, a_n$ ($-10^9 \leq a_i \leq 10^9$).

Output

Print $(n-k+1)$ space separated integers, the i-th integer should be the maximum element in the sliding window in the i-th iteration.

Examples

standard input	standard output

`8 3`
`1 3 -1 -3 5 3 6 7`
	`3 3 5 5 6 7`

Explanation

Window Position								Max
[1	3	-1]	-3	5	3	6	7	3
1	[3	-1	-3]	5	3	6	7	3
1	3	[-1	-3	5]	3	6	7	5
1	3	-1	[-3	5	3]	6	7	5
1	3	-1	-3	[5	3	6]	7	6
1	3	-1	-3	5	[3	6	7]	7

Problem F. Lexicographic arrangement

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 MB

You are given an array of distinct integers. You have to create a new list from the given array following a sequence of operations. In each operation, you remove the first element of the given array and add it **either to the beginning or the end of the list**. You perform the operations till there are no more elements left in the given array.

For example if the given array is [5,6,3,2,4], then one possible list can be [2,5,6,3,4] obtained by —

- We remove 5 and add it to the list, the list now is [5]
- We remove 6 and add it to the end of the list, the list now is [5,6]
- We remove 3 and add it to the end of the list, the list now is [5,6, 3]
- We remove 2 and add it to the beginning of the list, the list now is [2,5,6,3]
- We remove 4 and add it to the end of the list, the list now is [2,5,6,3,4].

Find the lexicographically smallest possible list that can be obtained by following the above operations.

A list $[x_1, x_2, \ldots, x_n]$ is said to be lexicographically smaller than the sequence $[y_1, y_2, \ldots, y_n]$ if there exists $i \le n$ such that $x_1 = y_1, x_2 = y_2 \ldots, x_{i-1} = y_{i-1}$ and $x_i < y_i$.

For example, the sequence [1,2,4,3] is **smaller** than the sequence [1,2,5,4] because after two matching elements, the third element of the first sequence (4) is smaller than the 3rd element of the second sequence (5).

Likewise, the sequence [1,3,4,3] is **not smaller** than the sequence [1,2,5,4] because after one matching element, the 2nd element of the first sequence (3) is greater than the 2nd element of the second sequence (2).

Implement the above algorithm using Double-ended queue only (You can take the input in an array, but the resulting list should be a doubly-ended queue)

Languages allowed - C

The time complexity of your solution should be $O(n)$. Any solution with a higher time complexity will not be considered.

Input

The first line of input contains n ($1 \le n \le 10^6$) — the size of the given array A.

The second line of input contains n space separated integers $A_0, A_1, \ldots, A_{n-1}$ ($1 \le A_i \le 10^9$) — the elements of the array A. The elements of A are distinct.

Output

Print n space separated integers - the elements of the lexicographically smallest possible list.

Examples

standard input	standard output
5 2 3 5 4 1	1 2 3 5 4
4 3 1 4 2	1 3 4 2

Problem G. Stock Spread

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 MB

You are an analyst in TMI securities. You are given the price of a stock in the last n days. You have to find the **spread** of the stock in each of the n days.

The **spread** of a stock on a given day is the maximum number of consecutive days (including the given day) going backward for which the stock price was less than or equal to the price of the day.

For example if the price of the stock in the last 6 days is [4, 7, 6, 3, 2, 6] the spread of the stock on the 6^{th} day will be 4 as the prices of the stock on the 6^{th}, 5^{th}, 4^{th}, and 3^{rd} days are less than or equal to the price on the 6^{th} day and the price on the 2^{nd} day is greater.

Languages allowed - C, C++

Allowed Header(s) if using C++: `iostream`, `stack`

The time complexity of your solution should be $O(n)$. Any solution with a higher time complexity will not be considered.

Input

The first line of input contains a single integer, n ($1 \leq n \leq 10^6$) — the number of days for which the price of the stock is given.

The second line of input contains n space-separated integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq 10^6$) — a_i denotes the price of the stock on the i^{th} day

Output

Print n space separated integers - the i^{th} integer denoting the spread of the stock on the i^{th} day.

Examples

standard input	standard output
7 110 85 65 73 65 78 89	1 1 1 2 1 4 6
6 29 14 25 93 117 76	1 1 2 4 5 1

Note

In the first example,

1. The spread of day 1 is 1 since only the 1^{st} day has a price less than or equal to it
2. The spread of day 2 is 1 since only the 2^{nd} day has a price less than or equal to it and the price on the 1^{st} day is greater.
3. The spread of day 3 is 1 since only the 3^{rd} day has a price less than or equal to it and the price on the 2^{nd} day is greater.
4. The spread of day 4 is 2 since the 4^{th} and 3^{rd} days have prices less than or equal to it and the price on the 2^{nd} day is greater.
5. The spread of day 5 is 1 since only the 5^{th} day has a price less than or equal to it and the price on the 4^{th} day is greater.

6. The spread of day 6 is 4 since the 6^{th}, 5^{th}, 4^{th} and 3^{rd} days have prices less than or equal to it and the price on the 2^{nd} day is greater.
7. The spread of day 7 is 6 since the 7^{th}, 6^{th}, 5^{th}, 4^{th}, 3^{rd} and 2^{nd} days have prices less than or equal to it and the price on the 1^{st} day is greater.

Problem H. Circular Winner

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 MB

It's your friend's birthday and everyone is tired of playing the same old games. You suggest a new game called Circular Shout. The rules of the game are as follows (assume n friends attend the party)

1. You all sit in a circle and everyone is numbered 1 to n in clockwise order. The person sitting at the 12 o'clock position is numbered 1. Moving clockwise from the i^{th} person will lead to the $(i+1)^{th}$ person (if $i < n$). Moving clockwise from the n^{th} person will lead to the 1^{st} person.
2. You start from the first person (i.e., number 1) and the person shouts out "1", then you go in clockwise order with the next person shouting the next number and so on till the person who shouts k.
3. The person who shouts out k leaves the circle and is out of the game
4. If there is more than one person left in the game, the game continues with the person next (in clockwise order) to the one that just lost the game shouting out "1" and so on.

Note that since the seating is circular, if at any point the number of people in the game is less than k, some people may end up shouting out more than one number. For example if there are 3 people and k is 5, and we start with the second person, then the game progresses as follows —

1. 2^{nd} person shouts out 1
2. 3^{rd} person shouts out 2
3. 1^{st} person shouts out 3
4. 2^{nd} person shouts out 4
5. 3^{rd} person shouts out 5 and is out of the game.

Since it is your friend's birthday, you want to help him/her win, find the position at which s/he should be seated so that s/he can win the game.

Implement the above algorithm using Circular queue only

Languages allowed - C

The time complexity of your solution should be $O(n * k)$. Any solution with a higher time complexity will not be considered.

Input

The input contains two integers, n and k ($1 \le n, k \le 500$) — the number of people attending the party and the integer which would lead to the person shouting it being out of the game. k can be less than, equal to or greater than n.

Output

A single integer, the position where your friend should be seated such that s/he wins the game.

Examples

standard input	standard output
7 3	4
9 4	1
423 321	201

Note

In the first example, initially the people in the game are - [1,2,3,4,5,6,7]

1. We start with 1 and 3 will be out, the players now are - [1,2,4,5,6,7]
2. We start with 4 and 6 will be out, the players now are - [1,2,4,5,7]
3. We start with 7 and 2 will be out, the players now are - [1,4,5,7]
4. We start with 4 and 7 will be out, the players now are - [1,4,5]
5. We start with 1 and 5 will be out, the players now are - [1,4]
6. We start with 1 and 1 will be out, the players now are - [4]

We stop now since only 1 player is left, the winning position is 4.

Problem I. Inefficient Paths

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 MB

You are working on a project with your friend. The code for it includes a lot of absolute paths to various directories in your local computer. Your friend was responsible for including some of these paths and s/he made the paths very complicated (though valid) reducing the readability of the code. You decide to simplify the path.

A valid absolute path

- Starts with '/'
- The path may contain directories, period '.' or double period '.'.
- Any two directories are separated by a slash '/' (single or multiple).
- The path will not have whitespace characters.

In a path

- A period '.' refers to the current directory.
- A double period '..' refers to the directory one level up.
- Any multiple consecutive slashes '//' are treated as a single slash '/'.

In a simplified absolute path

- The path starts with a single slash '/'.
- Any two directories are separated by a single slash '/'.
- The path doesn't end with trailing slashes '/'. (Unless it is the only '/' in the path).
- The path only contains the directories on the path from the root directory to the target file or directory (i.e., no period '.' or double period '..')
- The path will not have whitespace characters.

Given a valid absolute path, print the simplified path.

Languages allowed - C, C++

Allowed Header(s) if using C++: `iostream`, `stack`, `string`

The time complexity of your solution should be $O(n)$ where n is the length of the given path. Any solution with a higher time complexity will not be considered.

Input

The input consists of a string s ($1 \le |s| \le 10^5$) — denoting the path.

Output

Print a string representing the simplified path.

Examples

standard input	standard output
`/home/downloads/`	`/home/downloads`
`dir1/./dir2/../../dir3/`	`/dir3`
`/a/b/..//..`	`/`

Note

In the first example, since the simplified path should not end with trailing '/', we remove it.

In the second example,

1. We read dir1 and go to dir1 — We are currently at /dir1
2. We read '.' and stay at dir1 — We are currently at /dir1
3. We read dir2 and go to dir2 — We are currently at /dir1/dir2
4. We read '..', and go back to dir1 — We are currently at /dir1
5. We read '..', and go back to / — We are currently at /
6. We read dir3 and go to dir3 — We are currently at /dir3

In the third example

1. We read a and go to a — We are currently at /a
2. We read b and go to b — We are currently at /a/b
3. We read '..', and go back to a — We are currently at /a
4. We read '..', and go back to / — We are currently at /

Problem J. Make RBS

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 MB

A bracket sequence is any sequence of '(' and ')'. For example ()(), (()),)(are all bracket sequences. A bracket sequence is said to be **regular** if it is possible to obtain correct arithmetic expression by inserting characters + and 1 into this sequence. For example, sequences (())(), () and (()(())) are regular, while)(, (() and (()))(are not.

You are given a bracket sequence. In one operation you can **only add** a bracket (opening or closing) at any position. Find the minimum number of operations you need to perform to make the bracket sequence regular.

Note you are not allowed to remove a bracket or reshuffle the sequence, you can **only add** brackets.

Languages allowed - C, C++

Allowed Header(s) if using C++: `iostream`, `stack`, `string`

The time complexity of your solution should be $O(n)$ where n is the length of the given bracket sequence. Any solution with a higher time complexity will not be considered.

Input

The input consists of a string s ($1 \leq |s| \leq 10^5$) — the bracket sequence.

Output

Print a single integer - the minimum number of operations you need to perform to make the sequence regular.

Example

standard input	standard output
`()(((()`	`2`
`(()))`	`1`
`(()()())`	`0`

Note

In the first example, we can add two closing brackets at the end of the sequence and make it regular.

In the second example, we can add an opening bracket at the beginning of the sequence and make it regular.

In the third example, the given sequence is already regular, so we need not perform any operation on it.