Birla Institute of Technology and Science, Pilani
Hyderabad Campus

CS F211 Data Structures and Algorithms
Lab 3: Doubly Linked Lists and Binary Search

Allowed languages: **C**

General Tips

- Indent your code appropriately and use proper variable names. These increase readability and writability of the code. Also, use comments wherever necessary.
- Use a proper IDE or text editors like Sublime Text or VSCode as they help to run and test your code on multiple test-cases easily. However, in the lab you will be using command line interface.
- Try to use functions as much as possible in your code. Functions increase reusability and the pass-by-value feature provides a significant help sometimes. Modularizing your code also helps you to debug efficiently.
- You should not use any additional data structures like linked list (of any kind), array, etc.

Directions for Problems A-E

- Problems A-E are on doubly linked lists.
- You need to implement from scrach the required structure for Node and functions to construct a doubly linked list with the given input before solving the problem. **No template functions have been given for this lab sheet.**

Problem A. Best Playlist Ever

Input file:	`standard input`
Output file:	`standard output`
Time limit:	NA
Memory limit:	NA

Write a program to create an empty song playlist, add songs to it, and to play the songs in the playlist.

Every song is represented by a unique positive integer strictly greater than zero. No two songs will be represented by the same integer. You need not store any other information about a song apart from the integer that represents it.

The playlist should be implemented as a doubly linked list.

- Start by defining a `struct` for one song in the playlist. It should have the following members: an integer (to represent the song), and 2 pointers, one for the previous song and one for the next song.
- Next, define a function to add new songs to the playlist. New songs should get added to the end of the playlist.
- Following that, define a function to return the currently playing song. Songs are played in the order in which they were inserted into the playlist, starting with the 1st song. A song keeps playing on repeat until the user changes it to the previous/next song (if it exists).
- Finally, define two functions - one to play the previous song and one to play the next song. If the previous/next song does not exist, then keep playing the current song.

You will be provided with a sequence of songs initially present in the playlist, followed by a sequence of operations. Each operation will be one of the following:

- Operation 1: Add song x to the playlist (it is guaranteed that song x does not exist in the playlist already)
- Operation 2: Print the currently playing song. Note that the very first selection of this operation will print the first song of the playlist.
- Operation 3: Play the next song
- Operation 4: Play the previous song
- Operation 5: Exit the program

Input

The first line of input contains a single integer n — the number of songs initially present in the playlist.

The second line of input contains n space-separated unique integers — each representing one song initially present in the playlist.

The remaining lines of input contain one operation each. The first integer on each line will contain the Operation number (1, 2, 3, 4 or 5). For Operation 1, the lines will contain an additional integer specifying the new song, x. Operation 5 will be the last line of input.

Output

For every operation of type 2, print the currently playing song on a new line.

Examples

standard input	standard output
5 10 57 27 12 78 1 15 2 4 2 3 3 3 3 3 2 5	10 10 15

Note

In the example, initially, the playlist has the following songs: $10(Current\ song) \leftrightarrow 57 \leftrightarrow 27 \leftrightarrow 12 \leftrightarrow 78$. Note that initially, the first song is the current song.

- For the 1st operation, song 15 gets added to the playlist. Updated playlist : $10(Current\ song) \leftrightarrow 57 \leftrightarrow 27 \leftrightarrow 12 \leftrightarrow 78 \leftrightarrow 15$
- For the 2nd operation, the currently playing song, 10, gets printed.
- For the 3rd operation, since the previous song does not exist, song 10 continues to play.
- For the 4th operation, the currently playing song, 10, gets printed.
- For the 5th operation, the next song, 57, gets played. Updated playlist: $10 \leftrightarrow 57(Current\ song) \leftrightarrow 27 \leftrightarrow 12 \leftrightarrow 78 \leftrightarrow 15$
- For the 6th operation, the next song, 27, gets played. Updated playlist: $10 \leftrightarrow 57 \leftrightarrow 27(Current\ song) \leftrightarrow 12 \leftrightarrow 78 \leftrightarrow 15$
- For the 7th operation, the next song, 12, gets played. Updated playlist: $10 \leftrightarrow 57 \leftrightarrow 27 \leftrightarrow 12(Current\ song) \leftrightarrow 78 \leftrightarrow 15$
- For the 8th operation, the next song, 78, gets played. Updated playlist: $10 \leftrightarrow 57 \leftrightarrow 27 \leftrightarrow 12 \leftrightarrow 78(Current\ song) \leftrightarrow 15$
- For the 9th operation, the next song, 15, gets played. Updated playlist: $10 \leftrightarrow 57 \leftrightarrow 27 \leftrightarrow 12 \leftrightarrow 78 \leftrightarrow 15(Current\ song)$
- For the 10th operation, the currently playing song, 15, gets printed
- For the 11th operation, the playlist gets stopped and the program exits.

Problem B. Play Next

Input file:	`standard input`
Output file:	`standard output`
Time limit:	NA
Memory limit:	NA

This is an extension of Problem A. Add the option to play a song next to the playlist program from Problem A. Given a song x, first, check if it already exists in the playlist. If it does, then rearrange the playlist so that it becomes the next song. If it does not, then add it after the currently playing song in the playlist. In addition to the operations in Problem A, you will need to implement the following operation:

- Operation 6: Play song x next. Note that this operation does not directly play song x next. Rather, it re-arranges the list so that x appears after the current song and x can be played by selecting operation 3 next. See the example for better clarity.

You are not allowed to create a new linked list.

Input

The first line of input contains a single integer n — the number of songs initially present in the playlist.

The second line of input contains n space-separated unique integers — each representing one song initially present in the playlist.

The remaining lines of input contain one operation each. The first integer on each line will contain the Operation number (1, 2, 3, 4, 5, or 6). For Operations 1 and 6, the lines will contain an additional integer specifying the new song, x. Operation 5 will be the last line of input.

Output

For every operation of type 2, print the currently playing song on a new line.

Example

standard input	standard output
5 10 57 27 12 78 1 25 3 2 6 31 3 3 4 2 6 78 3 2 5	57 31 78

Note

In the example, initially the playlist had the following songs: $10(\textit{Current song}) \leftrightarrow 57 \leftrightarrow 27 \leftrightarrow 12 \leftrightarrow 78$

- For the 1st operation, song 25 gets added to the playlist. Updated playlist : $10(\textit{Current song}) \leftrightarrow 57 \leftrightarrow 27 \leftrightarrow 12 \leftrightarrow 78 \leftrightarrow 25$

- For the 2nd operation, the next song, 57, gets played. Updated playlist: $10 \leftrightarrow 57(Current\ song) \leftrightarrow 27 \leftrightarrow 12 \leftrightarrow 78 \leftrightarrow 25$
- For the 3rd operation, the currently playing song, 57, gets printed
- For the 4th operation, since 31 does not exist in the playlist already, it gets added after song 57. Updated playlist: $10 \leftrightarrow 57(Current\ song) \leftrightarrow 31 \leftrightarrow 27 \leftrightarrow 12 \leftrightarrow 78 \leftrightarrow 25$
- For the 5th operation, the next song, 31, gets played. Updated playlist: $10 \leftrightarrow 57 \leftrightarrow 31(Current\ song) \leftrightarrow 27 \leftrightarrow 12 \leftrightarrow 78 \leftrightarrow 25$
- For the 6th operation, the next song, 27, gets played. Updated playlist: $10 \leftrightarrow 57 \leftrightarrow 31 \leftrightarrow 27(Current\ song) \leftrightarrow 12 \leftrightarrow 78 \leftrightarrow 25$
- For the 7th operation, the previous song, 31, gets played. Updated playlist: $10 \leftrightarrow 57 \leftrightarrow 31(Current\ song) \leftrightarrow 27 \leftrightarrow 12 \leftrightarrow 78 \leftrightarrow 25$
- For the 8th operation, the currently playing song, 31, gets printed
- For the 9th operation, since 78 exists in the playlist, the playlist gets rearranged so that 78 becomes the next song. Updated playlist: $10 \leftrightarrow 57 \leftrightarrow 31(Current\ song) \leftrightarrow 78 \leftrightarrow 27 \leftrightarrow 12 \leftrightarrow 25$
- For the 10th operation, the next song, 78, gets played. Updated playlist: $10 \leftrightarrow 57 \leftrightarrow 31 \leftrightarrow 78(Current\ song) \leftrightarrow 27 \leftrightarrow 12 \leftrightarrow 25$
- For the 11th operation, the currently playing song, 78, gets printed
- For the 12th operation, the playlist gets stopped and the program exits

Problem C. Dyad Aggregate

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

Given a sorted doubly linked list and a target value, determine if there are two distinct nodes in the doubly linked list whose values add up to the target value. It is guaranteed that the values of the nodes in the doubly linked list are unique (i.e., no two distinct nodes have the same value). Note that positive as well as negative integers and zero can be present as the elements of the linked list.

Your solution must use only constant extra space. The expected time complexity of your solution should be *O(n)*, where n is the number of nodes in the doubly linked list.

Input

The first line of input contains two integers k ($-2 \cdot 10^9 \leq k \leq 2 \cdot 10^9$), n ($1 \leq n \leq 10^6$) — the target value and the size of the doubly linked list, respectively.

The second line of input contains n space separated integers $a_1, a_2, \ldots, a_n$ ($-10^9 \leq a_i \leq 10^9$) — the elements of the doubly linked list. Each element is unique.

Output

Print one line containing 1 if there are two distinct nodes in the doubly linked list whose values add up to the target value, 0 otherwise.

Examples

standard input	standard output
10 4 2 7 9 15	0
-1 3 -1 0 3	1

Problem D. Two Teams

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

DHCP has decided to send two teams to Arena '23. The coach of DHCP, decides to send two equally good teams. The coach is given the list of n players and their skill level in the game in the form of a **doubly linked list**. He wants the students at the beginning of the list to be in Team A, and the students at the end of the list in Team B.

Being the Assistant coach, you are asked to make the teams by following a series of operations. In one operation you can either -

- Remove a player from the beginning of the list and add him to Team A. (or)
- Remove a player from the end of the list and add him to Team B.

You have to form the teams such that -

- The total skill level of each team is the same
- The skill level of each team is maximized
- No player can play for more than one team

Note that -

- The number of players in each team need not be equal
- Some players may not be assigned a team
- The i-th player can be assigned to Team A only if the **first** $(i-1)$ players are assigned to Team A
- The i-th player can be assigned to Team B only if **all of** $(i+1)$-th to the n-th player are assigned to Team B

Find the number of players that are in each team. You do not have to print which players are in each team and hence do not have to store the teams in any form of data structure, you only have to print how many players will be assigned to each team.

Input

The first line of the input consists of a single integer n $(1 \leq n \leq 10^6)$ — the number of players.

The second line contains n space separated positive integers $a_1, a_2, ..., a_n$ $(1 \leq a_i \leq 10^9)$ — the skill level of the players.

Use **long** to prevent errors due to int overflow.

Output

Print two integers — the number of players in Team A and the number of players in Team B.

Examples

standard input	standard output
7 14 3 4 3 5 7 2	2 4
8 1 4 2 6 7 8 2 7	3 1

Note

In the first example,

- The first two players will be assigned to Team A - the total skill level is $14 + 3 = 17$
- The last three players will be assigned to Team B - the total skill level is $2 + 7 + 5 = 17$

Player numbers 3 and 4 will not be assigned to any team as assigning them to either Team A or Team B will lead to unequal skill level in both the teams.

In the second example,

- The first three players will be assigned to Team A - the total skill level is $1 + 4 + 2 = 7$
- The last player will be assigned to Team B - the total skill level is 7

It can be shown that the teams cannot be formed with a higher skill level such that all the constraints given in the problem statement are satisfied

Problem E. Rotating Places

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

Your class teacher has decided to set up a rotating seating arrangement, such that the student seated in the last row today will be seated in the first row tomorrow, the student seated in the first row today, will be seated in the second row tomorrow, and so on. The seating arrangement is given in the form of a doubly linked list where each student is represented by his/her class ID.

Being sick, Kishore will have to miss school for k days and now wants to know where he has to sit once he returns. Given the current arrangement, help Kishore by finding the seating arrangement after k days.

Note that only one student seats in a single row.

Consider the first node of the doubly linked list as the first row and the last node as the last row. You should find the arrangement by modifying the given input doubly linked list. **You are not allowed to change the values stored in the nodes of the doubly linked list or create a new one.**

Input

The first line of input contains two integers n ($1 \leq n \leq 10^6$), k ($1 \leq k \leq 10^6$) — the number of elements in the linked list and the number of days Kishore has to skip school respectively.

The second line of input contains n space-separated integers $a_1, a_2, ..., a_n$ ($1 \leq a_i \leq 10^9$) — the elements of the doubly linked list that represent the class IDs of the students.

Output

Print n space-separated integers — the seating arrangement after k days.

Examples

standard input	standard output
8 3 5 1 3 6 2 4 9 7	4 9 7 5 1 3 6 2

Problem F. Count Distinct

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

Arena '23 is here and lots of students from all over the country are visiting your college. As a member of the FOB, you have been tasked with choosing the colors for each college. Unfortunately, there is no list of colleges participating in the fest, only a list of students participating in the fest.

To make your task easier, other members of the FOB have assigned an integer to every student such that no two students belonging to different colleges have the same integer and no two students belonging to the same college have different integers. They have then sorted this sequence of integers and provided it to you as an array.

Given this array, count the number of different colleges participating in the fest so that you can assign colors to them.

The time complexity of your solution should be *O(k*log(n))*, where k is the number of different colleges and n is the size of the array, respectively.

Input

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

The second line of input contains n space separated integers $a_1, a_2, \ldots, a_n$ ($-10^6 \leq a_i \leq 10^6$) — the elements of the array.

Output

Print one line containing the number of colleges participating in the fest.

Example

standard input	standard output
12 3 3 3 3 5 19 19 19 19 19 1001 1001	4

Problem G. Deductions for Rock

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

Rock, the annual cultural fest of BPCC is coming up. All the students availed the provision of 0% attendance and skipped the GBM regarding deductions for Rock, so as the Student Committee Head you have the discretion to decide the amount to deduct from each student.

You are given a sorted array containing the amount of money in every student's account. There are two restrictions that the amount you decide on for deduction has to satisfy:

- it should not exceed a given deduction limit k
- the amount of money in at least one student's account should be exactly equal to the deduction amount that you decide upon

It is fine if one or more students have less money in the account than the deduction amount. Also, students can have negative balances in their accounts. To organize the biggest edition of Rock ever, you want to deduct the maximum amount possible. Find this amount.

The time complexity of your solution should be *O(logn)*, where n is the size of the array.

Input

The first line of input contains two integers n ($1 \leq n \leq 10^6$), k ($-10^9 \leq k \leq 10^9$) — the size of the array and the deduction limit, respectively.
The second line of input contains n space separated integers $a_1, a_2, \ldots, a_n$ ($-10^9 \leq a_i \leq 10^9$) — the elements of the sorted array.

Output

Print one line containing the maximum possible deduction amount.

Example

standard input	standard output
10 289 -189 -66 -63 171 197 250 273 358 427 499	273

Note

In the example, the deduction limit is 289. The maximum amount in a student's account that does not exceed 289 is 273, hence the maximum possible deduction amount that satisfies both the restrictions is 273.

Problem H. Square Root

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

You were recently taught a programming language called `badLang`. As part of your homework, you are asked to find the square root of a positive integer rounded down to the nearest integer. Incidentally, `badLang` does not have a function to compute the square root of an integer, but luckily for you, you just learned Binary Search in your DSA class and decided to use your knowledge of it to complete the given task. So, given an integer, find its square root using binary search only. **Note that if your implementation uses any means other than binary search, it will not be considered.**

Input

The first and only line of the input consists of a single integer, n ($1 \leq n \leq 10^{18}$)

Use **long** to take in the input.

Output

Print a single integer, the square root of n.

Examples

standard input	standard output
64	8
150	12

Problem I. Bad Search

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

Alice was asked to find the minimum element in an array as homework in her programming class. Being new to programming, she wrote a very inefficient algorithm. The algorithm works as follows (assume the length of the given array is n and 1-based indexing) —

- In the first step, she picks the first element of the array and compares it with the rest of the elements starting with the second element, i.e., she does $(n-1)$ comparisons
- In the second step, she picks the second element and compares it to the 3^{rd} element, 4^{th}, 5^{th} and so on, here she does $(n-2)$ comparisons.
- In general, in the i^{th} step, she compares the i^{th} element with the $(i+1)^{th}$ to the n^{th} element i.e., she does $(n-i)$ comparisons.

Since the algorithm is slow and takes a lot of time, Alice decides to do some other work as the algorithm runs. She wants you to remind her when the algorithm reaches the halfway stage. **The algorithm reaches the halfway stage when half of the total number of comparisons are done.** (The total number of comparisons will be $\frac{n(n-1)}{2}$).

Note that in case the total number of comparisons is odd (say k), then the halfway stage is when $\lfloor \frac{k}{2} \rfloor$ operations are done.

Find the step at which the algorithm reaches the halfway stage (Check the explanation for the samples for better understanding). **Use binary search to solve the problem. If your solution does not use binary search, it will not be considered.**

Input

The input consists of a single integer n ($1 \leq n \leq 10^9$) — the size of the array given to Alice.

Use **long** to prevent errors due to int overflow.

Output

A single integer — the step at which the algorithm reaches the halfway stage.

Examples

standard input	standard output
10	3
14	4

Note

In the first example, the total number of comparisons will be: $\frac{1}{2}(10 * 9) = 45$

Number of comparisons at the halfway stage: $\lfloor \frac{45}{2} \rfloor = 22$

The number of comparisons after the first step: 9

Total number of comparisons after the second step: $9 + 8 = 17$

Total number of comparisons after the third step: $9 + 8 + 7 = 24$

Therefore, the program reaches the halfway stage at step 3.

Note that you should not consider step 2 as the halfway stage because after step 2 only, 17 (not 22) comparisons are complete.

Problem J. Captain's Machine

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

Captain Miller needs k wooden planks of equal length to build a machine, the longer the planks, the bigger the machine. The Captain is given n wooden planks of different sizes (the sizes of the planks are stored in an integer array a containing only positive integers strictly greater than zero), the size of the i^{th} plank being a_i.

The planks can be broken down into smaller ones and then be used. The smallest size of a plank is 1, not lesser than that. However, smaller planks cannot be merged together to create a larger plank.

The Captain wants to build the biggest machine possible using k planks and needs your help in finding the longest possible length of the k wooden planks given the n planks. **Use binary search to solve the problem. If your solution does not use binary search, it will not be considered.**

Input

The first line contains two integers, n and k — the number of given planks and the number of planks required to build the machine respectively.

The second line consists of n space-separated integers — the length of each plank.

Output

A single integer — the longest possible length of the k planks to build the machine. If it is not possible to get k planks from the given input, print 0.

Example

standard input	standard output
5 6 3 6 9 4 8	4
4 10 1 3 3 2	0

Note

Assume 1-based indexing for the explanation.
In the first example, the second plank can be broken down into 2 planks of length **4** and 2
The third plank can be broken down into 3 planks of length **4**, **4** and 1
The fourth plank is used as is.
The fifth plank can be broken down into 2 planks of length **4** and **4**

Of course, there may be other ways in which you can get 6 planks of size 4.

Now finally, we have 6 planks of length 4 and it can be shown that it is not possible to get 6 or more planks of length greater than or equal to 5.

In the second example, it is not possible to obtain 10 planks from the given input. Hence, the output is 0.