A+B

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

This problem will not be graded for Homework 1

You are given two integers a and b. Print $a + b$.

Input

The only line of the input contains integers a and b ($-100 \leq a, b \leq 100$).

Output

Print $a + b$.

Examples

standard input	standard output
7 8	15
-100 100	0
-7 -99	-106

Note

In the first example, $a = 7$ and $b = 8$. Thus, the answer is $a + b = 7 + 8 = 15$.

Introduction to Time Complexity

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

This problem will not be graded for Homework 1

In algorithmic studies, we often focus on understanding how an algorithm works and analyzing its theoretical time complexity. However, in real-world applications, what really matters is the actual size of the data and the time constraints within which the algorithm must execute.

A widely accepted rule of thumb for estimating execution time, particularly on personal computers and online coding platforms, is that approximately 10^8 operations take around 1 second.

So, if an algorithm runs in $O(n)$ time, and $n \leq 10^8$, you can generally expect it to complete in under 1 second. However, this estimate can vary depending on the programming language and the specific hardware running the code.

Given an array of length n, your task is to find and output the maximum sum of any contiguous subarray within it. Note that the empty subarray is considered and has a sum of 0. Once you've implemented your solution, evaluate its time complexity to ensure it meets the real-world time constraints mentioned above.

Hint: For this problem, your grade will depend on the efficiency of your algorithm. If you implement an $O(n^3)$ algorithm, you will receive 60%. If you manage to optimize it to $O(n^2)$, you'll receive 80%. Achieving a solution with $O(n)$ time complexity will earn you 100%. However, this problem is ungraded, so feel free to experiment and try out different approaches without worrying about the final grade.

Input

The first line contains a positive integer n

The second line contains n integers $a_1, \cdots, a_n$. For all i, $-100 \leq a_i \leq 100$.

Output

Output a single integer, indicating the maximum sum. Note that empty subarrays are valid and they have a sum of 0.

Scoring

There are some subtasks in this problem, you will get the percentage of score if you pass the subtask

Subtask	Condition	Score	Additional Limitations
1	$n \leq 10$	20%	None
2	$n \leq 100$	20%	Must pass Subtask 1
3	$n \leq 1000$	20%	Must pass Subtask 1, 2
4	$n \leq 10^4$	20%	Must pass Subtask 1, 2, 3
5	$n \leq 10^6$	20%	Must pass Subtask 1, 2, 3, 4

Example

standard input	standard output
8 -1 3 -2 5 3 -5 2 2	9

Note

The subarray [3, -2, 5, 3] gives the maximum sum, which is 9.

Data Types

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

This problem will not be graded for Homework 1

When implementing algorithms in programming languages like C++ or Java, it's crucial to be aware of the data type ranges you're working with. For example:

- A 32-bit integer, represented as **int** in C++ and Java, has a range from -2^{31} to $2^{31}-1$
- A 64-bit integer, represented as **long long** in C++ and **long** in Java, has a range from -2^{63} to $2^{63}-1$

Understanding these ranges is essential for avoiding issues like integer overflow or underflow. These issues can lead to incorrect or unexpected output, potentially causing your algorithm to fail in real-world applications.

In this exercise, you will be given an array of length n. Your task is to calculate and output the sum of all elements in the array. Be mindful of the data type you choose to ensure that your code does not suffer from overflow or underflow issues.

Note that the default number types in some programming languages such as Python have no overflow issues. You are recommended to learn more about how your language of choice works with numbers.

Input

The first line contains a positive integer n.

The second line contains n integers $a_1, a_2, \cdots, a_n$.

Constraints

- $1 \leq n \leq 10^6$
- $-10^9 \leq a_i \leq 10^9$

Output

Output the sum of the array.

Example

standard input	standard output
5 1000000000 1000000000 1000000000 1000000000 1000000000	5000000000

The Great Box Shuffle

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

In the heart of Algoria, during the annual festival, a thrilling challenge called **The Great Box Shuffle** captivates all who attend. The challenge involves two long rows of enchanted boxes, each containing a collection of mystical items. These boxes are enchanted in such a way that, with a bit of magic, their contents can be shuffled to match each other.

The Great Box Shuffle festival in Algoria

You are the chosen contestant this year, and your task is to determine if specific groups of boxes from the first row can be rearranged to perfectly match groups of boxes from the second row. The magic allows you to rotate the items within any contiguous group of boxes, either to the left or right, as many times as you need.

Throughout the challenge, you will be given several queries. Each query will ask if a particular group of boxes from the first row can be rearranged to match a group of boxes from the second row using the magical rotations. Importantly, the magic can only be applied within the specified subarray or group of boxes.

More formally, you are given two sequences A and B, each of length N. You need to answer q queries. Each query is defined by four integers l_1, r_1, l_2, and r_2. For each query, you must determine if the subarray $A[l_1..r_1]$ can be rearranged to match the subarray $B[l_2..r_2]$ using the magic to rotate the items within the selected segment. The magic can be used an unlimited number of times, but it can only be applied within the specified subarray or group.

Input

- The first line contains a single integer N — the number of boxes in each row.
- The second line contains N integers $A_1, A_2, \ldots, A_N$, where A_i represents the number of items in the i-th box of the first row.
- The third line contains N integers $B_1, B_2, \ldots, B_N$, where B_i represents the number of items in the i-th box of the second row.
- The fourth line contains a single integer q — the number of queries.
- The next q lines each contain four integers l_1, r_1, l_2, and r_2. For each query, you need to determine whether it is possible to shuffle the group of boxes from the first row (from box l_1 to box r_1) to match the corresponding group of boxes from the second row (from box l_2 to box r_2).

Constraints

- $1 \leq N, Q \leq 200,000$
- $1 \leq l_1 \leq r_1 \leq N$
- $1 \leq l_2 \leq r_2 \leq N$
- $1 \leq A_i, B_i \leq N$

Output

For each query, print `Yes` if you can shuffle the group of boxes from the first row to match the corresponding group from the second row. Otherwise, print `No`.

Scoring

There are some subtasks in this problem, you will get the percentage of score if you pass the subtask

Subtask	Condition	Score	Additional Limitations
1	$n, q \leq 1000$	40%	None
2	No additional constraints	60%	Must pass Subtask 1

Example

standard input	standard output
5 1 2 3 2 4 2 3 1 4 2 4 1 3 1 3 1 2 3 5 1 4 2 5 1 5 1 5	Yes No No Yes

Note

- For the first query, the group of boxes from the first row ($\{1, 2, 3\}$) can be shuffled to match the target group from the second row ($\{2, 3, 1\}$).
- For the second query, the shuffling magic would not allow you to rearrange the group from the first row ($\{1, 2\}$) to match the target group ($\{1, 4, 2\}$).

Hint:

- Can you devise a clever method to compare the groups in $O(1)$ time? Consider using hashing, designing a hash function, or utilizing *prefix sum* or other prefix-based techniques to efficiently compare and process the segments.

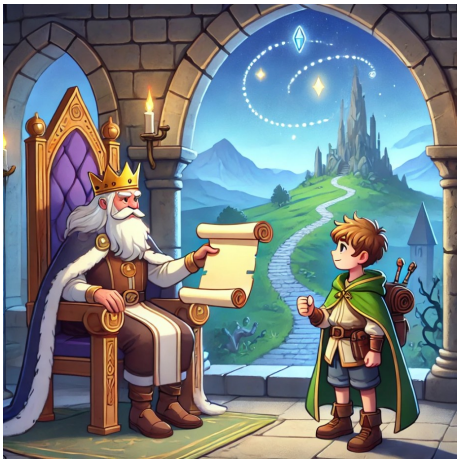
Scroll of Secrets

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 megabytes

This is an interactive problem

In the vast kingdom of DataLand, the wise King Algorithmus has hidden a series of magical scrolls along a secret path. Each scroll contains a number, and the scrolls are arranged in strict order, with each successive scroll bearing a larger number than the last. The kingdom's scholars refer to this path as a *singly linked list*, where each scroll not only holds a number but also points to the next scroll in the sequence.

You, the kingdom's chief seeker, have been tasked with finding the first scroll whose number is not less than a magical number x, which the king has entrusted to you. The challenge lies in the fact that the path is long, and you may only inspect a limited number of scrolls before the path vanishes into the mists of time.



The King hands you the magical scroll

Your mission is to navigate this mysterious path efficiently. Starting from a given scroll, you must determine the smallest number that is greater than or equal to x. If no such number exists, you must return to the king with the news that the quest has failed.

The Path:

- The path consists of n scrolls, each containing a number and a clue (an index) pointing to the next scroll.
- You begin your quest at a scroll indexed as `start`.
- The scrolls are sorted in ascending order, meaning the number on each scroll is less than the number on the next scroll, as long as it exists.

Your Tools:

- You can ask about a specific scroll by its index to reveal the number it holds and the index of the next scroll.
- However, the ancient magic of the path only allows you to ask up to 5000 questions before the path dissolves.

You task is to find and report the smallest number on the path that is greater than or equal to x. If no such number exists, report that the quest has failed.

Input

The number of scrolls n, the index of the first scroll `start`, and the magical number x.

Constraints

- $1 \leq n \leq 10^5$
- $1 \leq \texttt{start} \leq n$
- $0 \leq x \leq 10^9$
- You may ask no more than 5000 questions.

Output

- If you find the desired number, return it to the king. Otherwise, return -1 to signify the quest's failure.

Interaction Protocol

- To make a query about a scroll, print `? i`, where i is the index of the scroll.
- After printing, read two values: `value_i`, which is the number on the scroll, and `next_i`, which is the index of the next scroll in the sequence.
- You may make up to 5000 such queries.
- To submit your final answer, print `! ans`, where `ans` is the smallest number greater than or equal to x. If no such number exists, print `! -1`.
- After the final answer is submitted, your program should terminate.

Scoring

The scoring for this problem depends on the number of queries used:

- If you use $\leq$ 100000 queries, you will receive 50% of the points.
- If you use $\leq$ 60000 queries, you will receive 75% of the points.
- If you successfully use $\leq$ 5000 queries, you will receive full points.

Example

standard input	standard output
5 3 80 97 -1 58 5 16 2 81 1 79 4	81

Note

For the first example, in a quest where $n = 5$, `start = 3`, and $x = 80$, your path may look something like this:

- Scroll 1 holds the number 97 and is the last scroll.
- Scroll 2 holds the number 58 and points to Scroll 5.
- Scroll 3 holds the number 16 and points to Scroll 2.
- Scroll 4 holds the number 81 and points to Scroll 1.
- Scroll 5 holds the number 79 and points to Scroll 4.

A strategy would be to ask 5 queries to get the information for all scrolls, then you would report back to the king with the number 81 as it is the smallest number greater than or equal to 80.

Hints:

- Think about the *nuts and bolts* problem discussed in class. Just as matching each nut to its corresponding bolt required clever strategies, this problem may benefit from a similar approach. Consider using randomization to explore the scrolls efficiently and find the correct number. The key lies in making wise choices about which scrolls to inspect.