Special Assignment:

#1. A bracket is considered to be any one of the following characters: (,), {, }, [, or]. Two brackets are considered to be a matched pair if the an opening bracket (i.e., (, [, or {) occurs to the left of a closing bracket (i.e.,),], or }) of the exact same type. There are three types of matched pairs of brackets: [], {},and ().

A matching pair of brackets is not balanced if the set of brackets it encloses are not matched. For example, {[(])} is not balanced because the contents in between { and } are not balanced. The pair of square brackets encloses a single, unbalanced opening bracket, (, and the pair of parentheses encloses a single, unbalanced closing square bracket,].

By this logic, we say a sequence of brackets is balanced if the following conditions are met:

It contains no unmatched brackets.

The subset of brackets enclosed within the confines of a matched pair of brackets is also a matched pair of brackets.

Given n strings of brackets, determine whether each sequence of brackets is balanced. If a string is balanced, return YES. Otherwise, return NO.

SAMPLE INPUT

```
3
{[()]}
{[(])}
{{[[(())]]}}
```

SAMPLE OUTPUT

```
YES
NO
YES
```

#2. You are given a stack of **N** integers such that the first element represents the top of the stack and the last element represents the bottom of the stack. You need to pop at least one element from the stack. At any one moment, you can convert stack into a queue. The bottom of the stack represents the front of the queue. You cannot convert the queue back into a stack. Your task is to remove exactly **K** elements such that the sum of the **K** removed elements is maximised.

SAMPLE INPUT

```
10 5
10 9 1 2 3 4 5 6 7 8
```

SAMPLE OUTPUT

```
40
```

Explanation

Pop two elements from the stack. i.e {10,9}

Then convert the stack into queue and remove first three elements from the queue. i.e {8,7,6}.

The maximum possible sum is 10+9+8+7+6 = 40

#3. You are given a stack of **N** integers. In one operation, you can either pop an element from the stack or push any popped element into the stack. You need to maximize the top element of the stack after performing exactly **K** operations. If the stack becomes empty after performing K operations and there is no other way for the stack to be non-empty, print **-1**.

SAMPLE INPUT
6 4
1 2 4 3 3 5
SAMPLE OUTPUT
4

Explanation
In 3 operations, we remove 1,2,4 and in the fourth operation, we push 4 back into the stack. Hence, 4 is the answer.

#4. The Monk is trying to explain to its users that even a single unit of time can be extremely important and to demonstrate this particular fact he gives them a challenging task.

There are **N** processes to be completed by you, the chosen one, since you're Monk's favorite student. All the processes have a unique number assigned to them from **1 to N**.

Now, you are given two things:

- The **calling** order in which all the processes are called.
- The **ideal** order in which all the processes should have been executed.

Now, let us demonstrate this by an example. Let's say that there are **3 processes**, the calling order of the processes is: **3 - 2 - 1.** The ideal order is: **1 - 3 - 2,** i.e., process number 3 will only be executed after process number 1 has been completed; process number 2 will only be executed after process number 3 has been executed.

- *Iteration #1:* Since the ideal order has process #1 to be executed firstly, the calling ordered is changed, i.e., the first element has to be pushed to the last place. Changing the position of the element takes 1 unit of time. The new calling order is: 2 - 1 - 3. Time taken in step #1: 1.

- *Iteration #2:* Since the ideal order has process #1 to be executed firstly, the calling ordered has to be changed again, i.e., the first element has to be pushed to the last place. The new calling order is: 1 - 3 - 2. Time taken in step #2: 1.

- *Iteration #3:* Since the first element of the calling order is same as the ideal order, that process will be executed. And it will be thus popped out. Time taken in step #3: 1.

- *Iteration #4:* Since the new first element of the calling order is same as the ideal order, that process will be executed. Time taken in step #4: 1.

- *Iteration #5:* Since the last element of the calling order is same as the ideal order, that process will be executed. Time taken in step #5: 1.

Total time taken: 5 units.

PS: Executing a process takes 1 unit of time. Changing the position takes 1 unit of time.

SAMPLE INPUT

```
3
3 2 1
1 3 2
```

SAMPLE OUTPUT

```
5
```