Problem C. Two Increasing Arrays

Problem Description

In the set D, the equivalence relation $=$ is defined with the following properties:

- Reflexivity: For every $a \in D$, it holds that $a = a$.
- Transitivity: For every $a, b, c \in D$, if $a = b$ and $b = c$, then $a = c$.
- Symmetry: For every $a, b \in D$, if $a = b$, then $b = a$.

In the set D, the total ordering relation $\leq$ is defined with the following properties:

- Completeness: For every $a, b \in D$, either $a \leq b$ or $b \leq a$ holds.
- Transitivity: For every $a, b, c \in D$, if $a \leq b$ and $b \leq c$, then $a \leq c$.
- Antisymmetry: For every $a, b \in D$, if $a \leq b$ and $b \leq a$, then $a = b$.

The strict total ordering relation $<$ associated with the total ordering relation $\leq$ satisfies:

- For every $a, b \in D$, $a < b$ if and only if $a \leq b$ and $a \neq b$.

Consider the following problem: You are given two arrays, a and b, both of length n, and strictly increasing according to a total ordering. Your task is to determine the k^{th} smallest element in the merged array $[a[0], a[1], \ldots, a[n-1], b[0], b[1], \ldots, b[n-1]]$.

The total ordering ensures a well-defined comparison relationship between elements in both arrays. Specifically, for any $0 \leq i, j \leq n-1$, it is guaranteed that $a[i]$ and $b[j]$ are distinct, preventing any ambiguity in their relative positions within the merged array.

You can make several queries to the interaction library, each query in the form of:

- Equivalence Test: Given elements $x, y \in D$, the interaction library returns whether $x = y$ is true or not.
- Partial Order Test: Given elements $x, y \in D$, the interaction library returns whether $x \leq y$ is true or not.

There is a restriction on the number of partial order tests as mentioned in the problem. Your score will depend on the correctness of your answers and the number of partial order tests you perform after each function call.

Implementation Details

You are required to implement the following procedure:

```
Number two_increasing_arrays(int n, int k, Number[] a, Number[] b)
```

- n: the size of the arrays a and b.
- k: the procedure should return the k^{th} smallest element in the merged array $[a[0], a[1], \ldots, a[n-1], b[0], b[1], \ldots, b[n-1]]$.
- a, b: arrays of length n. For $0 \leq i \leq n-2$, the following conditions hold:
  - $a[i] < a[i+1]$
  - $b[i] < b[i+1]$
- This procedure is called exactly t times.

Make sure to include the header file `1622.h`.

In this header file, a data type `Number` is defined, describing elements in D.

The interactive library has overloaded operators for `Number`, defining binary relations as follows:

The <= operator describes the binary relation $\leq$ with the following interface:

```
bool operator <= (const Number &rhs) const;
```

For two variables of type `Number`, say `x` and `y` (corresponding to $x, y \in D$), `x <= y` will return a `bool` variable. It returns `true` if $x \leq y$, otherwise, it returns `false`.

Calling the <= operator incurs a cost of 1 unit.

The == operator describes the binary relation $=$ with the following interface:

```
bool operator == (const Number &rhs) const;
```

For two variables of type `Number`, say `x` and `y` (corresponding to $x, y \in D$), `x == y` will return a `bool` variable. It returns `true` if $x = y$, otherwise, it returns `false`.

Calling the == operator incurs a cost of 0 units.

Additionally, for ease of problem-solving, we have overloaded operators >, <, >=, `!=`, defined as follows:

- `a > b` is equivalent to `!(a <= b)`. Calling this operator incurs a cost of 1 unit.
- `a < b` is equivalent to `!(b <= a)`. Calling this operator incurs a cost of 1 unit.
- `a >= b` is equivalent to `b <= a`. Calling this operator incurs a cost of 1 unit.
- `a != b` is equivalent to `!(a == b)`. Calling this operator incurs a cost of 0 units.

For your debugging convenience, a function `get_value()` is provided. Calling this function will return the member variable `info` of the `Number` class. This method is only included in the sample interactive library to aid your debugging. During evaluation, you cannot obtain the value of the variable `info` by any means, as it will be considered an attack on the interactive library, rendering the scores invalid.

Constraints

- $1 \le t \le 500$.
- $1 \le n \le 1024$.
- $1 \le k \le 2n$.

Subtasks

1. (100 points) No additional constraints.

In subtask 1 you can obtain a partial score. Let q be the maximum number of partial order tests you make across all scenarios. Your score for this subtask is calculated according to the following table:

Maximum tests	Score
$q > 100\,000$	0
$21\,000 < q \le 100\,000$	10
$21 < q \le 21\,000$	$10 + 80 \cdot \log_{1000}\left(\frac{21\,000}{q}\right)$
$11 < q \le 21$	$90 + (21 - q)$
$q \le 11$	100

The score is rounded down to the nearest integer. You can see the scoring function in the "Appendix" section.

No.	Testdata Range	Time Limit (ms)	Memory Limit (KiB)
1	`1-16`	`1000`	`262144`

Example

Consider the following call:

```
two_increasing_arrays(3, 4, [1, 3, 5], [2, 4, 6])
```

The arrays a and b are both of length 3, and strictly increasing according to a total ordering. The goal is to find the 4^{th} smallest element, which is $b[1]$.

The following describes one possible way of the interaction:

Operator Call	Return Value
`a[0] < b[2]`	`true`
`a[2] < b[1]`	`false`
`b[0] >= a[1]`	`false`
`a[1] <= a[2]`	`true`
`a[0] != a[0]`	`false`
`b[1] > a[1]`	`true`
`b[1] == a[1]`	`false`
`b[0] > a[0]`	`true`

The program makes 6 partial order tests in total.

Since we know $a[0] < b[0] < a[1] < b[1] < a[2]$ and $b[1] < b[2]$, we can conclude that $b[1]$ is the 4^{th} smallest element. Thus the procedure should return $b[1]$.

Please note that you will **NOT** be able to access the values of a and b directly.

Sample grader

The sample grader reads the input in the following format:

- line 1: $n \;\; k$
- line 2: $a[0] \;\; a[1] \;\; \ldots \;\; a[n-1]$
- line 3: $b[0] \;\; b[1] \;\; \ldots \;\; b[n-1]$

In order for the sample grader to work properly, your input should satisfy the following conditions:

- $0 \le a[i], b[i] \le 2^{32} - 1$.
- $a[0] < a[1] < \cdots < a[n-1]$.
- $b[0] < b[1] < \cdots < b[n-1]$.
- $a[i] \neq b[j]$ for $0 \le i, j \le n-1$.

The sample grader prints your queries in the following format:

If the answer is correct, the sample grader prints `Accepted: <cost>` in the second line, where `<cost>` is the number of partial order tests you make.

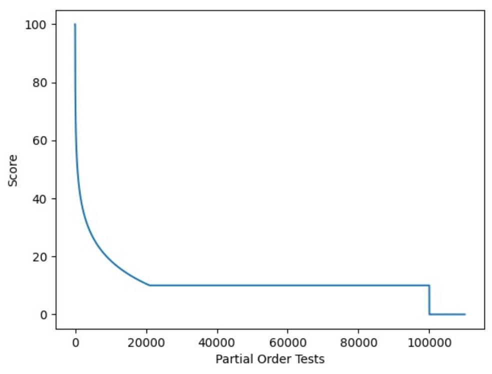
If the answer is incorrect, the sample grader prints `Wrong Answer: <MSG>` in the second line, where `<MSG>` is one of the following:

- `Returned Value is Incorrect`: the return value of `two_increasing_arrays` is incorrect.

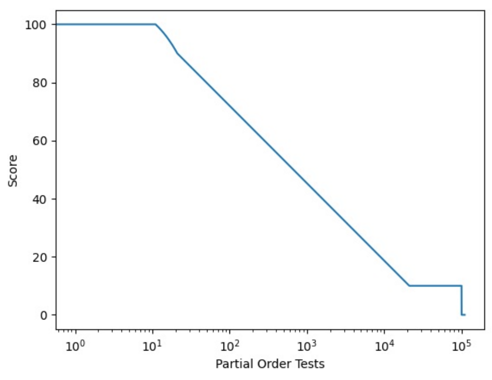
Notes

- Here is a sample implementation. (Link)
- You should include `"1622.h"` in your program.
- You should **NOT** implement the `main` function.
- You should only submit `1622.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 modify the grader as you want.
- **Note that the actual grader will be different from the sample grader, and your solution should NOT rely on the implementation of the grader.**
- You can use `g++ -std=c++17 -O2 -o 1622 1622.cpp grader.cpp` to compile the code, and use `./1622` or `1622.exe` to run the code.

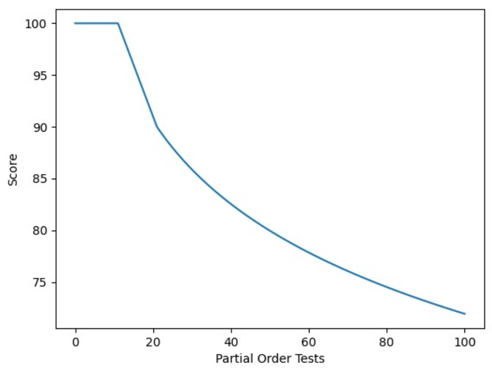
Appendix



The scoring function in linear scale.



The scoring function in logarithm scale.



The scoring function for range $[0, 100]$ in linear scale.