Problem A. Easy Problem

Problem Description

KCW is an algorithm enthusiast. Today, he came up with an easy problem. Here's the problem statement:

There is a grid of size $2^n \times 2^n$, and KCW wants you to fill in the numbers from 1 to 2^{2n} into the cells of the grid.

The numbers are filled into the grid following a specific rule: if the size of the grid is greater than one, divide it into four quadrants: top-left, top-right, bottom-left, and bottom-right. Place the first quarter of the numbers in the top-left quadrant, the next quarter from $\frac{1}{4}$ to $\frac{1}{2}$ in the top-right quadrant, the subsequent quarter from $\frac{1}{2}$ to $\frac{3}{4}$ in the bottom-left quadrant, and the rest in the bottom-right quadrant. If the size of the grid is 1×1, fill it with the corresponding number.

KCW will give you the initial size of the grid, represented by n. He wants to know how the numbers will be placed in the grid in the end.

Input Format

- line 1: n

Output Format

- line i ($1 \leq i \leq 2^n$): the 2^n numbers in the i^{th} row of the grid.

Constraints

- $0 \leq n \leq 10$.
- All the inputs are integers.

Subtasks

1. (10 points) $n \leq 2$.
2. (30 points) $n \leq 5$.
3. (60 points) No additional constraints.

No.	Testdata Range	Time Limit (ms)	Memory Limit (KiB)
Samples	1-2	1000	262144
1	1-3	1000	262144
2	1-6	1000	262144
3	1-11	1000	262144

Samples

Sample Input 1

```
1
```

This sample input satisfies the constraints of all the subtasks.

Sample Output 1

```
1 2
3 4
```

Sample Input 2

```
2
```

This sample input satisfies the constraints of all the subtasks.

Sample Output 2

```
1 2 5 6
3 4 7 8
9 10 13 14
11 12 15 16
```

Problem B. Let's Write Divide and Conquer Happily!

Problem Description

In a digital realm where some of the finest minds in the programming world gather, a group of esteemed individuals has gathered to tackle a complex problem. Among them are **9s6e0i3c5**, **detaomega**, **mastermindccr**, **qwe854896**, **rayray9999**, **roylin506**, **huci0062, mmi366127**, and **SorahISA**. Their mission? To master the art of Divide and Conquer in solving a range of queries concerning an array of integers.

9s6e0i3c5, our resident thinker and problem solver, takes the lead in the discussion. With a sharp mind for algorithms, **9s6e0i3c5** outlines the problem at hand:

"For this challenge, we're dealing with an array x of n integers and a series of q queries. These queries come in four flavors:

1. Update the value at position k to u.
2. Find the sum of values in the range $[ql, qr]$.
3. Discover the maximum prefix sum in the range $[ql, qr]$,
   i.e., $\max(0, \max_{i \in [ql,qr]} \sum_{k=ql}^{i} x[k])$, we denote this value as $\text{pmax}([ql, qr])$.
4. Unearth the maximum suffix sum in the same range $[ql, qr]$,
   i.e., $\max(0, \max_{j \in [ql,qr]} \sum_{k=j}^{qr} x[k])$, we denote this value as $\text{smax}([ql, qr])$.

Now, let's delve into how we can efficiently handle these queries."

detaomega, the pragmatic coder, injects a dose of realism into the conversation:

"Let's begin by considering the second operation to get started - finding the sum of values in the interval $[ql, qr]$, represented as $\text{sum}([ql, qr])$. If we compute the sum for each query naively, we're looking at a time complexity of $\mathcal{O}(n)$. With q queries, the total time complexity becomes $\mathcal{O}(qn)$, which is far from ideal. So, let's explore how we can optimize this."

mastermindccr, known for insightful observations, proposes an elegant solution:

"To enhance the efficiency of range sum queries, perhaps we can employ a Divide and Conquer strategy. For any interval $[l, r]$ and $l \neq r$, we can use $m = \left\lfloor \frac{l + r}{2} \right\rfloor$ to divide the interval into smaller subintervals $[l, m]$ and $[m + 1, r]$, calculate the sum of each one, and then sum them to get the sum of $[l, r]$."

SorahISA, the *MASTER TA*, offers his insights:

"Now, let's analyze the time complexity. Assume $T(n)$ is the time complexity of solving one range sum query with a length of n using Divide and Conquer, we can express it as follows:

$$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + \mathcal{O}(1)$$

By applying the Master Theorem (~~qwe854896 Theorem~~), we conclude that $T(n) = \mathcal{O}(n)$, it won't be any better!" Since SorahISA is an algorithm master, you can always believe what he said.

mmi366127 adds his expertise to the discussion:

"Maybe..., we can store the results for these subintervals, allowing us to query them again without recomputing them? ~~Memory is always free.~~"

rayray9999, the memory wizard, raises a valid concern:

"However, there are $\mathcal{O}(n^2)$ possible subintervals, and memorizing them might lead to a **Memory Limit Exceeded**. We need to be more clever about this."

roylin506, the guy who is always curious about everything, offers a thoughtful suggestion:

"Is there a compromise solution that allows us to avoid memorizing too many answers while still providing enough information to quickly assemble an answer?"

SorahISA delves into a creative idea (since he is always a master):

"We don't necessarily have to divide an interval right at the midpoint. We can choose other points to divide and still obtain the correct answer in the end. For instance:

- $\forall m \in [l, r-1],\ \text{sum}([l, r]) = \text{sum}([l, m]) + \text{sum}([m+1, r])$.

And it's not limited to just one cut, for example:

- $\text{sum}([l_1, r_1]) + \text{sum}([l_2, r_2]) + \ldots + \text{sum}([l_k, r_k])$ such that $l_i = r_{i-1} + 1$.

Perhaps we only need to memorize the answers encountered in one specific query, and then for subsequent queries, we can partition them into intervals that match the recorded answers perfectly!

If we're only recording the answer from one specific query, it would make sense to remember all the intervals encountered during the calculation of $\text{sum}([1, n])$ because all queried intervals are encompassed within $[1, n]$!"

huci0062 inquires:

"Nice idea, but how can we find the specific intervals to combine them to the answer of any $[ql, qr]$?"

qwe854896 provides further insight:

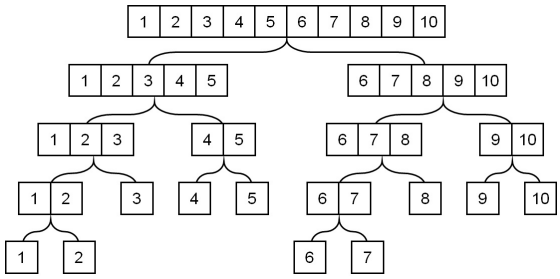
"Instead of thinking about how to divide $[ql, qr]$ into intervals we've memorized, let's consider how the memorized answers can assist us in calculating the answer for $[ql, qr]$.

Specifically, you can initially use Divide and Conquer to calculate the answer for $[1, n]$ and record the answers encountered during the recursion."

SorahISA elaborates on the process:

"Then, to calculate the answer for $[ql, qr]$, you can start from $[1, n]$ and employ Divide and Conquer once more. During the recursion, for the current interval $[l, r]$ and the queried interval $[ql, qr]$, there are three situations:

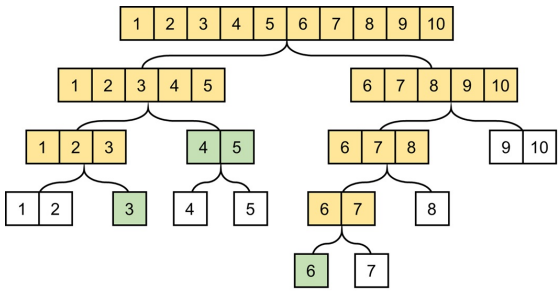
1. $[l, r]$ is entirely inside $[ql, qr]$. In this case, we can directly add $\text{sum}([l, r])$ to our answer.
2. $[l, r]$ is entirely outside $[ql, qr]$. In this scenario, this interval doesn't affect the answer.
3. Otherwise, we can divide it into two subproblems:
    1. How the answer of the interval $[l, m]$ contributes to $\text{sum}([ql, qr])$.
    2. How the answer of the interval $[m + 1, r]$ contributes to $\text{sum}([ql, qr])$."



mastermindccr seeks clarification with an example:

"Could you provide some examples? Suppose we want to find the answer for $[3, 6]$ within $[1, 10]$. How can we use these intervals to achieve that?"

detaomega responds with a visual aid:



"Similar to the image above, you can utilize these green intervals (situation 1): $[3, 3], [4, 5], [6, 6]$ to determine $\text{sum}([3, 6])$. As for the yellow intervals (situation 3), you can't predict their contribution, so you divide them. For the uncolored interval (situation 2), it won't contribute in any way."

roylin506 inquires again:

"So what is the time complexity of implementing this?"

qwe854896 adds valuable information:

"We can observe that situation 3, where $[l, r]$ is neither entirely inside nor outside of $[ql, qr]$, will always reduce to either situation 1 or situation 2 in the next recursion layer.

I've also noticed an interesting property: for intervals at the same recursion depth, situation 3 occurs at most twice!"

rayray9999 calculates the implications:

"This insight implies that the number of cases we need to consider is bounded by a constant multiplied by the depth of recursion. There, the time complexity is determined by the depth of recursion, which is $\mathcal{O}(\log n)$!"

huci0062 continues with a follow-up question:

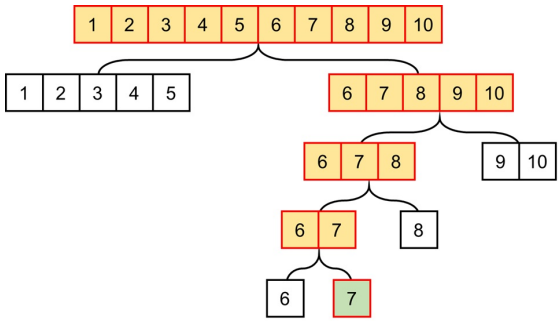
"Now that we can efficiently solve the range summation operation, how can we update the value at position k to u?"

mmi366127 expands on the idea:

"We can treat updates as a range query of $[k, k]$. The path of the range query traverses through intervals whose values will be affected if the k^{th} number changes to u, and we can handle this efficiently."

huci0062 responds with a detailed explanation and an illustrative image:

"So when we want to update 7^{th} number to 63, we first locate the interval $[7, 7]$ by following the path $[1, 10] \to [6, 10] \to [6, 8] \to [6, 7] \to [7, 7]$, and update all the answers along this path as shown in the image below:"



"Once $[7, 7]$ is updated to 63, we move back to $[6, 7]$ using the answer for $[6, 6]$ and the updated answer for $[7, 7]$ to compute the new answer for $[6, 7]$. We then proceed to $[6, 8]$ using the updated answer for $[6, 7]$ and the answer for $[8, 8]$ to calculate the new answer for $[6, 8]$, and so on, working our way back until we've updated $[1, 10]$."

roylin506 confirms:

"That's correct! Now we can also update answers in $\mathcal{O}(\log n)$!"

SorahISA seeks further understanding:

"So the last question is how to calculate the maximum prefix sum? It seems like a challenging aspect."

9s6e0i3c5 provides an answer:

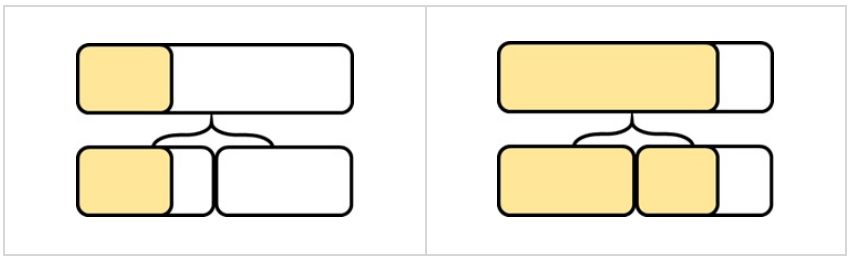
"Certainly, to calculate the maximum prefix sum, denoted as $\text{pmax}([ql, qr])$, we can apply a similar Divide and Conquer approach. The key is to modify the way you merge the answer from the left side and the right side.

For each interval with a length greater than or equal to 2, the maximum prefix sum can only happen in the following two cases:

In the first case, we can obtain the candidate answer by considering $\text{pmax}(\text{left interval})$.

On the other hand, we can get the other candidate answer by adding $\text{sum}(\text{left interval})$ and $\text{pmax}(\text{right interval})$.

By considering the maximum over these two possible cases, we can merge and determine the answer!"



And so, these brilliant minds continue their journey to Divide and Conquer this intricate problem, learning and growing together in their quest to master Divide and Conquer, all with smiles on their faces. As for you, the task of finding solutions for maximum prefix sum and maximum suffix sum and implementing all of this is assigned to you.

Input Format

The first input line has two integers n and q: the number of values and queries.

The second line has n integers $x_1, x_2, \ldots, x_n$: the array values.

Finally, there are q lines describing the queries. Each line start with one integer op.

If $op = 1$, then the last two numbers are k and u; otherwise, the last two numbers are ql and qr.

Output Format

Print the result of each query of type 2, 3, and 4.

Constraints

- $1 \leq n, q \leq 200\,000$.
- $-10^9 \leq x_i \leq 10^9$.
- $op \in \{1, 2, 3, 4\}$.
- $1 \leq k \leq n$.
- $-10^9 \leq u \leq 10^9$.
- $1 \leq ql \leq qr \leq n$.

Subtasks

1. (5 points) $n \leq 5000$; $q \leq 5000$.
2. (20 points) $op = 2$ for all queries.
3. (20 points) $op \in \{1, 2\}$ for all queries.
4. (25 points) $op \in \{2, 3, 4\}$ for all queries.
5. (5 points) $ql = 1$; $qr = n$.
6. (25 points) No additional constraints.

No.	Testdata Range	Time Limit (ms)	Memory Limit (KiB)
Samples	1-3	1000	262144
1	1-6	1000	262144
2	7-10	1000	262144
3	7-14	1000	262144
4	7-10,15-18	1000	262144
5	19-22	1000	262144
6	1-26	1000	262144

Samples

Sample Input 1

```
10 4
4 -8 7 -6 3 -4 8 -7 6 -3
2 3 6
2 4 8
1 7 63
2 4 8
```

This sample input satisfies the constraints of Subtasks 1, 3, 6.

Sample Output 1

```
0
-6
49
```

Sample Input 2

```
10 7
4 -8 7 -6 3 -4 8 -7 6 -3
2 1 10
3 1 10
4 1 10
1 7 63
2 1 10
3 1 10
4 1 10
```

This sample input satisfies the constraints of Subtasks 1, 5, 6.

Sample Output 2

```
0
4
4
55
59
59
```

Sample Input 3

```
10 5
4 -8 7 -6 3 -4 8 -7 6 -3
2 3 6
3 4 8
1 7 63
3 4 8
4 2 6
```

This sample input satisfies the constraints of Subtasks 1, 6.

Sample Output 3

```
0
1
56
0
```

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

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

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
$21\,000 < q \leq 100\,000$	10
$21 < q \leq 21\,000$	$10 + 80 \cdot \log_{1000}\left(\frac{21\,000}{q}\right)$
$11 < q \leq 21$	$90 + (21 - q)$
$q \leq 11$	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:

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

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.

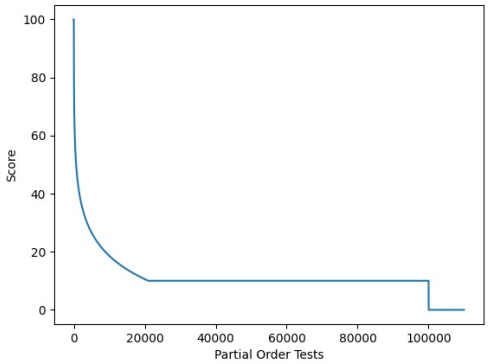
If the answer is incorrect, the sample grader prints `Wrong Answer: <MSG>` in the second line, where `<MSG>` is one of the following:

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

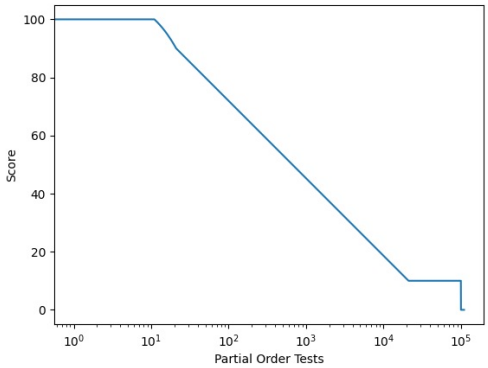
Notes

- 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.

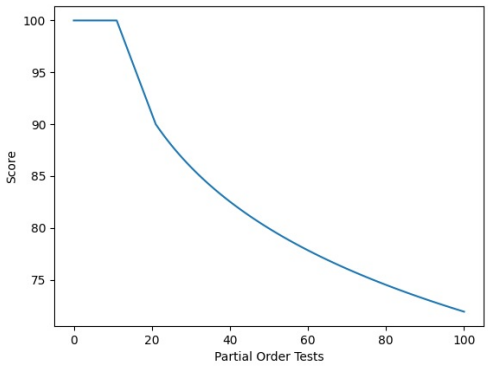
Appendix



The scoring function in linear scale.



The scoring function in logarithm scale.



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

Problem D. Monkey

Problem Description

At "Animal World Sanctuary" zoo, there are n monkeys live and work for the zookeeper.

These monkeys are numbered from 1 to n.

They follow certain rules:

1. Each monkey can oversee up to two other monkeys.
2. Each monkey should be overseen by another monkey. However, there's a monkey king who doesn't listen to anyone else.
3. The monkeys with no one to oversee are called "little brother monkeys."

From these certain rules we can find that the monkey relationship can be represented as a binary tree.

To avoid spending too much time on paying bananas as the monkeys' salary, you decide to give all the bananas and the salary list to the monkey king. However, there are two questions bothering you:

- Who is the monkey king?
- To prevent the monkey king from stealing away others' salary, you want to know how many bananas should the "little brother monkeys" have eaten in total.

Can you write a program to find out?

Input Format

Given the monkey relationship with post-order and in-order.

- line 1: n as the number of monkey
- line 2: b_1 b_2 $\ldots$ b_n as the banana given to i^{th} monkey.
- line 3: in_1 in_2 $\ldots$ in_n as the in-order representation of the monkey relationship.
- line 4: $post_1$ $post_2$ $\ldots$ $post_n$ as the post-order representation of the monkey relationship.

Output Format

- line 1: index of the king monkey.
- line 2: the sum of bananas eaten by all of the "little brother monkey."

Constraints

- $1 \leq n \leq 2\,000\,000$.
- $1 \leq b_i \leq 10^9$ for $i = 1, 2, \ldots, n$.
- $\{in_1, in_2, \ldots, in_n\}$ is a permutation of $\{1, 2, \ldots, n\}$.
- $\{post_1, post_2, \ldots, post_n\}$ is a permutation of $\{1, 2, \ldots, n\}$.
- The relationship could be constructed from the in-order and post-order representations.
- All input values are integers.

Subtasks

1. (60 points) Monkey only oversee those whose index is bigger them itself.
2. (40 points) No additional constraints.

No.	Testdata Range	Time Limit (ms)	Memory Limit (KiB)
Samples	1-2	1000	262144
1	3-15	1000	262144
2	1-28	1000	262144

Samples

Sample Input 1

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

This sample input satisfies the constraints of all the subtasks.

Sample Output 1

```
1
23
```

Sample Input 2

```
5
50 20 40 10 30
5 4 3 2 1
1 2 3 4 5
```

This sample input satisfies the constraints of Subtask 2.

Sample Output 2

```
5
50
```

Problem E. Binary Exponentiation

- 2023.10.23 00:30 Update: Fixed typo in Sample Input description.

Problem Description

Given four integers x, y, k, and p, please calculate $\left\lfloor \frac{x^y}{k} \right\rfloor \bmod p$.

There are t testcases.

Input Format

- line 1: t
- line $2+i$ ($0 \leq i \leq t-1$): x y k p

Output Format

- line $1+i$ ($0 \leq i \leq t-1$): the answer for the i^{th} testcase.

Constraints

- $1 \leq t \leq 100\,000$.
- $1 \leq x, y, k, p \leq 10^9$.
- All the inputs are integers.

Subtasks

1. (10 points) $k = 1$; $y \leq 100$.
2. (65 points) $k = 1$.
3. (10 points) $k = 2$.
4. (5 points) $y \leq 100$.
5. (10 points) No additional constraints.

No.	Testdata Range	Time Limit (ms)	Memory Limit (KiB)
Samples	1	1000	262144
1	2	1000	262144
2	2-3	1000	262144
3	4	1000	262144
4	2,5	1000	262144
5	1-6	1000	262144

Samples

Sample Input 1

```
7
2 20 1 998244353
987654321 100 1 1000000000
998244352 100 1 998244353
1000000000 1 1 100
314159265 358979323 1 846264338
314159265 358979323 2 846264338
314159265 358979323 846264338 327950288
```

This sample input satisfies the constraints of Subtask 5.

Sample Output 1

```
1048576
409912001
1
0
604903687
725584012
36855733
```

- In the 1^{st} testcase: $2^{20} \bmod 998\,244\,353 = 1\,048\,576$.
- In the 3^{rd} testcase: $998\,244\,352^{100} \bmod 998\,244\,353 = (-1)^{100} \bmod 998\,244\,353 = 1$.
- In the 4^{th} testcase: $1\,000\,000\,000^{1} \bmod 100 = 0$.

Problem F. Bad Sequence

- 2023.11.01 03:40 Update: Strengthened testcases and rejudged solutions.
- Trick or Treat! Added a simpler subtask and Sample 5.

Problem Description

We consider an integer sequence $[c_1, c_2, \ldots, c_m]$, and let the minimum and maximum value of the sequence be x and y, respectively. We call the sequence k-bad if $y - x - m + 1 = k$.

For example, the sequence $[4, 8, 7, 6, 3]$ is 1-bad, and the sequence $[1, 1, 2, 2, 3, 3]$ is (-3)-bad.

Given an array a and an integer k, please count the number of pairs (ℓ, r) $(\ell \leq r)$ such that the sequence $[a_\ell, a_{\ell+1}, \ldots, a_r]$ is k-bad.

Input Format

- line 1: n k
- line 2: a_1 a_2 $\ldots$ a_n

Output Format

- line 1: the number of pairs that is k-bad.

Constraints

- $2 \leq n \leq 500\,000$.
- $0 \leq k \leq n - 2$.
- $1 \leq a_i \leq n$ for $i = 1, 2, \ldots, n$.
- All input values are integers.

Subtasks

1. (5 points) $n \leq 1000$; $\{a_1, a_2, \ldots, a_n\}$ is a permutation of $\{1, 2, \ldots, n\}$.
2. (5 points) $n \leq 1000$.
3. (30 points) $n \leq 80\,000$; $k = 0$; $\{a_1, a_2, \ldots, a_n\}$ is a permutation of $\{1, 2, \ldots, n\}$.
4. (30 points) $n \leq 80\,000$; $\{a_1, a_2, \ldots, a_n\}$ is a permutation of $\{1, 2, \ldots, n\}$.
5. (10 points) $n \leq 80\,000$.
6. (15 points) $\{a_1, a_2, \ldots, a_n\}$ is a permutation of $\{1, 2, \ldots, n\}$.
7. (5 points) No additional constraints.

No.	Testdata Range	Time Limit (ms)	Memory Limit (KiB)
Samples	1-5	2000	262144
1	6-17	2000	262144
2	1-27	2000	262144
3	28-37	2000	262144
4	6-17,28-43	2000	262144
5	1-51	2000	262144
6	6-17,28-43,52-62	2000	262144
7	1-70	2000	262144

Samples

Sample Input 1

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

This sample input satisfies the constraints of Subtasks 1, 2, 4, 5, 6, 7.

Sample Output 1

```
2
```

The sequence $[8, 4, 2]$ and $[6, 1]$ are 4-bad, and there is no 5-bad sequence.

The number of 0, 1, 2, 3, 4, 5, and 6 bad sequences are 11, 6, 8, 9, 2, 0, and 0, respectively.

Sample Input 2

```
5 1
1 1 1 1 1
```

This sample input satisfies the constraints of Subtasks 2, 5, 7.

Sample Output 2

```
0
```

- Every subarray of length 1 is 0-bad.
- Every subarray of length 2 is (-1)-bad.
- Every subarray of length 3 is (-2)-bad.
- Every subarray of length 4 is (-3)-bad.
- Every subarray of length 5 is (-4)-bad.

There is no 1-bad sequence.

Sample Input 3

```
15 0
1 15 2 14 3 13 4 12 5 11 6 10 7 9 8
```

This sample input satisfies the constraints of all the subtasks.

Sample Output 3

```
29
```

- There is 29 0-bad sequences.
- There is $(14 - k)$ k-bad sequences for $k = 1, 2, \ldots, 13$.

Sample Input 4

```
20 8
13 17 6 9 19 13 1 15 19 4 4 10 1 1 16 15 9 8 1 14
```

This sample input satisfies the constraints of Subtasks 2, 5, 7.

Sample Output 4

```
17
```

Sample Input 5

```
9 0
3 4 5 1 2 9 8 7 6
```

This sample input satisfies the constraints of all the subtasks.

Sample Output 5

```
21
```