Codeforces Round #672 (Div. 2)

A. Cubes Sorting

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

For god's sake, you're boxes with legs! It is literally your only purpose! Walking onto buttons! How can you not do the one thing you were designed for?
Oh, that's funny, is it? Oh it's funny? Because we've been at this for twelve hours and you haven't solved it either, so I don't know why you're laughing. You've got one hour! Solve it!

Wheatley decided to try to make a test chamber. He made a nice test chamber, but there was only one detail absent — cubes.

For completing the chamber Wheatley needs n cubes. i-th cube has a volume a_i.

Wheatley has to place cubes in such a way that they would be sorted in a non-decreasing order by their volume. Formally, for each $i > 1$, $a_{i-1} \le a_i$ must hold.

To achieve his goal, Wheatley can exchange two **neighbouring** cubes. It means that for any $i > 1$ you can exchange cubes on positions $i-1$ and i.

But there is a problem: Wheatley is very impatient. If Wheatley needs more than $\frac{n \cdot (n-1)}{2} - 1$ exchange operations, he won't do this boring work.

Wheatly wants to know: can cubes be sorted under this conditions?

Input

Each test contains multiple test cases.

The first line contains one positive integer t ($1 \le t \le 1000$), denoting the number of test cases. Description of the test cases follows.

The first line of each test case contains one positive integer n ($2 \le n \le 5 \cdot 10^4$) — number of cubes.

The second line contains n positive integers a_i ($1 \le a_i \le 10^9$) — volumes of cubes.

It is guaranteed that the sum of n over all test cases does not exceed 10^5.

Output

For each test case, print a word in a single line: "YES" (without quotation marks) if the cubes can be sorted and "NO" (without quotation marks) otherwise.

Example

input

```
3
5
5 3 2 1 4
6
2 2 2 2 2 2
2
2 1
```

output

```
YES
YES
NO
```

Note

In the first test case it is possible to sort all the cubes in 7 exchanges.

In the second test case the cubes are already sorted.

In the third test case we can make 0 exchanges, but the cubes are not sorted yet, so the answer is "NO".

B. Rock and Lever

time limit per test: 1 second

memory limit per test: 256 megabytes
input: standard input
output: standard output

"You must lift the dam. With a lever. I will give it to you.
You must block the canal. With a rock. I will not give the rock to you."

Danik urgently needs rock and lever! Obviously, the easiest way to get these things is to ask Hermit Lizard for them.

Hermit Lizard agreed to give Danik the lever. But to get a stone, Danik needs to solve the following task.

You are given a positive integer n, and an array a of positive integers. The task is to calculate the number of such pairs (i, j) that $i < j$ and $a_i \ \& \ a_j \geq a_i \oplus a_j$, where $\&$ denotes the bitwise AND operation, and $\oplus$ denotes the bitwise XOR operation.

Danik has solved this task. But can you solve it?

Input

Each test contains multiple test cases.

The first line contains one positive integer t ($1 \leq t \leq 10$) denoting the number of test cases. Description of the test cases follows.

The first line of each test case contains one positive integer n ($1 \leq n \leq 10^5$) — length of the array.

The second line contains n positive integers a_i ($1 \leq a_i \leq 10^9$) — elements of the array.

It is guaranteed that the sum of n over all test cases does not exceed 10^5.

Output

For every test case print one non-negative integer — the answer to the problem.

Example

input

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

output

```
1
3
2
0
0
```

Note

In the first test case there is only one pair: $(4, 7)$: for it $4 \ \& \ 7 = 4$, and $4 \oplus 7 = 3$.

In the second test case all pairs are good.

In the third test case there are two pairs: $(6, 5)$ and $(2, 3)$.

In the fourth test case there are no good pairs.

C1. Pokémon Army (easy version)

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

This is the easy version of the problem. The difference between the versions is that the easy version has no swap operations. You can make hacks only if all versions of the problem are solved.

Pikachu is a cute and friendly pokémon living in the wild pikachu herd.

But it has become known recently that infamous team R wanted to steal all these pokémon! Pokémon trainer Andrew decided to help Pikachu to build a pokémon army to resist.

First, Andrew counted all the pokémon — there were exactly n pikachu. The strength of the i-th pokémon is equal to a_i, and all these numbers are distinct.

As an army, Andrew can choose any non-empty subsequence of pokemons. In other words, Andrew chooses some array b from k

indices such that $1 \le b_1 < b_2 < \dots < b_k \le n$, and his army will consist of pokémons with forces $a_{b_1}, a_{b_2}, \dots, a_{b_k}$.

The strength of the army is equal to the alternating sum of elements of the subsequence; that is, $a_{b_1} - a_{b_2} + a_{b_3} - a_{b_4} + \dots$.

Andrew is experimenting with pokémon order. He performs q operations. In i-th operation Andrew swaps l_i-th and r_i-th pokémon.

Note: $q = 0$ in this version of the task.

Andrew wants to know the maximal stregth of the army he can achieve with the initial pokémon placement. He also needs to know the maximal strength after each operation.

Help Andrew and the pokémon, or team R will realize their tricky plan!

Input

Each test contains multiple test cases.

The first line contains one positive integer t ($1 \le t \le 10^3$) denoting the number of test cases. Description of the test cases follows.

The first line of each test case contains two integers n and q ($1 \le n \le 3 \cdot 10^5, q = 0$) denoting the number of pokémon and number of operations respectively.

The second line contains n distinct positive integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le n$) denoting the strengths of the pokémon.

i-th of the last q lines contains two positive integers l_i and r_i ($1 \le l_i \le r_i \le n$) denoting the indices of pokémon that were swapped in the i-th operation.

It is guaranteed that the sum of n over all test cases does not exceed $3 \cdot 10^5$, and the sum of q over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, print $q + 1$ integers: the maximal strength of army before the swaps and after each swap.

Example

input

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

output

```
3
2
9
```

Note

In third test case we can build an army in such way: [1 2 **5** 4 **3** 6 **7**], its strength will be $5 - 3 + 7 = 9$.

C2. Pokémon Army (hard version)

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

This is the hard version of the problem. The difference between the versions is that the easy version has no swap operations. You can make hacks only if all versions of the problem are solved.

Pikachu is a cute and friendly pokémon living in the wild pikachu herd.

But it has become known recently that infamous team R wanted to steal all these pokémon! Pokémon trainer Andrew decided to help Pikachu to build a pokémon army to resist.

First, Andrew counted all the pokémon — there were exactly n pikachu. The strength of the i-th pokémon is equal to a_i, and all these numbers are distinct.

As an army, Andrew can choose any non-empty subsequence of pokemons. In other words, Andrew chooses some array b from k indices such that $1 \le b_1 < b_2 < \dots < b_k \le n$, and his army will consist of pokémons with forces $a_{b_1}, a_{b_2}, \dots, a_{b_k}$.

The strength of the army is equal to the alternating sum of elements of the subsequence; that is, $a_{b_1} - a_{b_2} + a_{b_3} - a_{b_4} + \dots$.

Andrew is experimenting with pokémon order. He performs q operations. In i-th operation Andrew swaps l_i-th and r_i-th pokémon.

Andrew wants to know the maximal stregth of the army he can achieve with the initial pokémon placement. He also needs to know the maximal strength after each operation.

Help Andrew and the pokémon, or team R will realize their tricky plan!

Input

Each test contains multiple test cases.

The first line contains one positive integer t ($1 \le t \le 10^3$) denoting the number of test cases. Description of the test cases follows.

The first line of each test case contains two integers n and q ($1 \le n \le 3 \cdot 10^5, 0 \le q \le 3 \cdot 10^5$) denoting the number of pokémon and number of operations respectively.

The second line contains n distinct positive integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le n$) denoting the strengths of the pokémon.

i-th of the last q lines contains two positive integers l_i and r_i ($1 \le l_i \le r_i \le n$) denoting the indices of pokémon that were swapped in the i-th operation.

It is guaranteed that the sum of n over all test cases does not exceed $3 \cdot 10^5$, and the sum of q over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, print $q + 1$ integers: the maximal strength of army before the swaps and after each swap.

Example

input

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

output

```
3
4
2
2
2
9
10
10
10
9
11
```

Note

Let's look at the third test case:

Initially we can build an army in such way: [1 2 **5** 4 **3** 6 **7**], its strength will be $5 - 3 + 7 = 9$.

After first operation we can build an army in such way: [**2 1 5** 4 **3** 6 **7**], its strength will be $2 - 1 + 5 - 3 + 7 = 10$.

After second operation we can build an army in such way: [**2 1 5** 4 **3 7** 6], its strength will be $2 - 1 + 5 - 3 + 7 = 10$.

After third operation we can build an army in such way: [**2 1** 4 **5 3 7** 6], its strength will be $2 - 1 + 5 - 3 + 7 = 10$.

After forth operation we can build an army in such way: [1 2 4 **5 3 7** 6], its strength will be $5 - 3 + 7 = 9$.

After all operations we can build an army in such way: [1 **4 2 5 3 7** 6], its strength will be $4 - 2 + 5 - 3 + 7 = 11$.

D. Rescue Nibel!

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Ori and Sein have overcome many difficult challenges. They finally lit the Shrouded Lantern and found Gumon Seal, the key to the Forlorn Ruins. When they tried to open the door to the ruins... nothing happened.

Ori was very surprised, but Sein gave the explanation quickly: clever Gumon decided to make an additional defence for the door.

There are n lamps with Spirit Tree's light. Sein knows the time of turning on and off for the i-th lamp — l_i and r_i respectively. To open the door you have to choose k lamps in such a way that there will be a moment of time when they all will be turned on.

While Sein decides which of the k lamps to pick, Ori is interested: how many ways there are to pick such k lamps that the door will open? It may happen that Sein may be wrong and there are no such k lamps. The answer might be large, so print it modulo

$998\,244\,353$.

Input

First line contains two integers n and k ($1 \le n \le 3 \cdot 10^5$, $1 \le k \le n$) — total number of lamps and the number of lamps that must be turned on simultaneously.

Next n lines contain two integers l_i ans r_i ($1 \le l_i \le r_i \le 10^9$) — period of time when i-th lamp is turned on.

Output

Print one integer — the answer to the task modulo $998\,244\,353$.

Examples

input

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

output

```
9
```

input

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

output

```
3
```

input

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

output

```
0
```

input

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

output

```
1
```

input

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

output

```
7
```

Note

In first test case there are nine sets of k lamps: $(1, 2, 3)$, $(1, 2, 4)$, $(1, 2, 5)$, $(1, 2, 6)$, $(1, 3, 6)$, $(1, 4, 6)$, $(2, 3, 6)$, $(2, 4, 6)$, $(2, 6, 7)$.

In second test case $k = 1$, so the answer is 3.

In third test case there are no such pairs of lamps.

In forth test case all lamps are turned on in a time 3, so the answer is 1.

In fifth test case there are seven sets of k lamps: $(1, 2)$, $(1, 3)$, $(2, 3)$, $(2, 4)$, $(3, 4)$, $(3, 5)$, $(4, 5)$.

E. Battle Lemmings

time limit per test: 2 seconds

memory limit per test: 512 megabytes

input: standard input
output: standard output

A lighthouse keeper Peter commands an army of n battle lemmings. He ordered his army to stand in a line and numbered the lemmings from 1 to n from left to right. Some of the lemmings hold shields. Each lemming cannot hold more than one shield.

The more protected Peter's army is, the better. To calculate the *protection* of the army, he finds the number of protected pairs of lemmings, that is such pairs that both lemmings in the pair don't hold a shield, but there is a lemming with a shield between them.

Now it's time to prepare for defence and increase the protection of the army. To do this, Peter can give orders. He chooses a lemming with a shield and gives him one of the two orders:

- give the shield to the left neighbor if it exists and doesn't have a shield;
- give the shield to the right neighbor if it exists and doesn't have a shield.

In one second Peter can give exactly one order.

It's not clear how much time Peter has before the defence. So he decided to determine the maximal value of army protection for each k from 0 to $\frac{n(n-1)}{2}$, if he gives no more that k orders. Help Peter to calculate it!

Input

First line contains a single integer n ($1 \le n \le 80$), the number of lemmings in Peter's army.

Second line contains n integers a_i ($0 \le a_i \le 1$). If $a_i = 1$, then the i-th lemming has a shield, otherwise $a_i = 0$.

Output

Print $\frac{n(n-1)}{2} + 1$ numbers, the greatest possible protection after no more than $0, 1, \ldots, \frac{n(n-1)}{2}$ orders.

Examples

input

```
5
1 0 0 0 1
```

output

```
0 2 3 3 3 3 3 3 3 3 3
```

input

```
12
0 0 0 0 1 1 1 1 0 1 1 0
```

output

```
9 12 13 14 14 14 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15
15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15
```

Note

Consider the first example.

The protection is initially equal to zero, because for each pair of lemmings without shields there is no lemmings with shield.

In one second Peter can order the first lemming give his shield to the right neighbor. In this case, the protection is two, as there are two protected pairs of lemmings, $(1, 3)$ and $(1, 4)$.

In two seconds Peter can act in the following way. First, he orders the fifth lemming to give a shield to the left neighbor. Then, he orders the first lemming to give a shield to the right neighbor. In this case Peter has three protected pairs of lemmings — $(1, 3)$, $(1, 5)$ and $(3, 5)$.

You can make sure that it's impossible to give orders in such a way that the protection becomes greater than three.