Codeforces Global Round 19

A. Sorting Parts

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

You have an array a of length n. You can **exactly** once select an integer len between 1 and $n-1$ inclusively, and then sort in non-decreasing order the prefix of the array of length len and the suffix of the array of length $n-len$ independently.

For example, if the array is $a = [3, 1, 4, 5, 2]$, and you choose $len = 2$, then after that the array will be equal to $[1, 3, 2, 4, 5]$.

Could it be that after performing this operation, the array will **not** be sorted in non-decreasing order?

Input

There are several test cases in the input data. The first line contains a single integer t ($1 \le t \le 100$) — the number of test cases. This is followed by the test cases description.

The first line of each test case contains one integer n ($2 \le n \le 10^4$) — the length of the array.

The second line of the test case contains a sequence of integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the array elements.

It is guaranteed that the sum of n over all test cases does not exceed 10^4.

Output

For each test case of input data, output "YES" (without quotes), if the array may be **not** sorted in non-decreasing order, output "NO" (without quotes) otherwise. You can output each letter in any case (uppercase or lowercase).

Example

input
3 3 2 2 1 4 3 1 2 1 5 1 2 2 4 4
output
YES YES NO

Note

In the first test case, it's possible to select $len = 1$, then after operation, the array will not be sorted in non-decreasing order and will be equal to $[2, 1, 2]$.

In the second test case, it's possible to select $len = 3$, then after operation, the array will not be sorted in non-decreasing order and will be equal to $[1, 2, 3, 1]$.

In the third test case, the array will be sorted in non-decreasing order for every possible len.

B. MEX and Array

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

Let there be an array $b_1, b_2, \ldots, b_k$. Let there be a partition of this array into segments $[l_1; r_1], [l_2; r_2], \ldots, [l_c; r_c]$, where $l_1 = 1$, $r_c = k$, and for any $2 \le i \le c$ holds that $r_{i-1} + 1 = l_i$. In other words, each element of the array belongs to exactly one segment.

Let's define the *cost* of a partition as

$$c + \sum_{i=1}^{c} \operatorname{mex}(\{b_{l_i}, b_{l_i+1}, \ldots, b_{r_i}\}),$$

where mex of a set of numbers S is the smallest non-negative integer that does not occur in the set S. In other words, the *cost* of a partition is the number of segments plus the sum of MEX over all segments. Let's define the *value* of an array $b_1, b_2, \ldots, b_k$ as the

maximum possible *cost* over all partitions of this array.

You are given an array a of size n. Find the sum of *values* of all its subsegments.

An array x is a subsegment of an array y if x can be obtained from y by deletion of several (possibly, zero or all) elements from the beginning and several (possibly, zero or all) elements from the end.

Input
The input contains several test cases. The first line contains one integer t ($1 \le t \le 30$) — the number of test cases.

The first line for each test case contains one integer n ($1 \le n \le 100$) — the length of the array.

The second line contains a sequence of integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 10^9$) — the array elements.

It is guaranteed that the sum of the values n over all test cases does not exceed 100.

Output
For each test case print a single integer — the answer to the problem.

Example

input
4 2 1 2 3 2 0 1 4 2 0 5 1 5 0 1 1 0 1
output
4 14 26 48

Note
In the second test case:

- The best partition for the subsegment $[2, 0, 1]$: $[2], [0, 1]$. The cost of this partition equals to $2 + \operatorname{mex}(\{2\}) + \operatorname{mex}(\{0, 1\}) = 2 + 0 + 2 = 4$.
- The best partition for the subsegment $[2, 0]$: $[2], [0]$. The cost of this partition equals to $2 + \operatorname{mex}(\{2\}) + \operatorname{mex}(\{0\}) = 2 + 0 + 1 = 3$
- The best partition for the subsegment $[2]$: $[2]$. The cost of this partition equals to $1 + \operatorname{mex}(\{2\}) = 1 + 0 = 1$.
- The best partition for the subsegment $[0, 1]$: $[0, 1]$. The cost of this partition equals to $1 + \operatorname{mex}(\{0, 1\}) = 1 + 2 = 3$.
- The best partition for the subsegment $[0]$: $[0]$. The cost of this partition equals to $1 + \operatorname{mex}(\{0\}) = 1 + 1 = 2$.
- The best partition for the subsegment $[1]$: $[1]$. The cost of this partition equals to $1 + \operatorname{mex}(\{1\}) = 1 + 0 = 1$.

The sum of values over all subsegments equals to $4 + 3 + 1 + 3 + 2 + 1 = 14$.

C. Andrew and Stones

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

Andrew has n piles with stones. The i-th pile contains a_i stones. He wants to make his table clean so he decided to put every stone either to the 1-st or the n-th pile.

Andrew can perform the following operation any number of times: choose 3 indices $1 \le i < j < k \le n$, such that the j-th pile contains at least 2 stones, then he takes 2 stones from the pile j and puts one stone into pile i and one stone into pile k.

Tell Andrew what is the minimum number of operations needed to move all the stones to piles 1 and n, or determine if it's impossible.

Input
The input contains several test cases. The first line contains one integer t ($1 \le t \le 10\,000$) — the number of test cases.

The first line for each test case contains one integer n ($3 \le n \le 10^5$) — the length of the array.

The second line contains a sequence of integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the array elements.

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

Output
For each test case print the minimum number of operations needed to move stones to piles 1 and n, or print -1 if it's impossible.

Example

input
4 5 1 2 2 3 6 3 1 3 1 3 1 2 1 4 3 1 1 2
output
4 -1 1 -1

Note

In the first test case, it is optimal to do the following:

1. Select $(i, j, k) = (1, 2, 5)$. The array becomes equal to $[2, 0, 2, 3, 7]$.
2. Select $(i, j, k) = (1, 3, 4)$. The array becomes equal to $[3, 0, 0, 4, 7]$.
3. Twice select $(i, j, k) = (1, 4, 5)$. The array becomes equal to $[5, 0, 0, 0, 9]$. This array satisfy the statement, because every stone is moved to piles 1 and 5.

There are 4 operations in total.
In the second test case, it's impossible to put all stones into piles with numbers 1 and 3:

1. At the beginning there's only one possible operation with $(i, j, k) = (1, 2, 3)$. The array becomes equal to $[2, 1, 2]$.
2. Now there is no possible operation and the array doesn't satisfy the statement, so the answer is -1.

In the third test case, it's optimal to do the following:

1. Select $(i, j, k) = (1, 2, 3)$. The array becomes equal to $[2, 0, 2]$. This array satisfies the statement, because every stone is moved to piles 1 and 3.

The is 1 operation in total.
In the fourth test case, it's impossible to do any operation, and the array doesn't satisfy the statement, so the answer is -1.

D. Yet Another Minimization Problem

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

You are given two arrays a and b, both of length n.

You can perform the following operation any number of times (possibly zero): select an index i ($1 \le i \le n$) and swap a_i and b_i.

Let's define the *cost* of the array a as $\sum_{i=1}^{n} \sum_{j=i+1}^{n} (a_i + a_j)^2$. Similarly, the *cost* of the array b is $\sum_{i=1}^{n} \sum_{j=i+1}^{n} (b_i + b_j)^2$.

Your task is to minimize the total *cost* of two arrays.

Input

Each test case consists of several test cases. The first line contains a single integer t ($1 \le t \le 40$) — the number of test cases. The following is a description of the input data sets.

The first line of each test case contains an integer n ($1 \le n \le 100$) — the length of both arrays.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 100$) — elements of the first array.

The third line of each test case contains n integers $b_1, b_2, \ldots, b_n$ ($1 \le b_i \le 100$) — elements of the second array.

It is guaranteed that the sum of n over all test cases does not exceed 100.

Output

For each test case, print the minimum possible total cost.

Example

input
3 1 3 6 4 3 6 6 6 2 7 4 1

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

output

```
0
987
914
```

Note

In the second test case, in one of the optimal answers after all operations $a = [2, 6, 4, 6]$, $b = [3, 7, 6, 1]$.

The cost of the array a equals to $(2 + 6)^2 + (2 + 4)^2 + (2 + 6)^2 + (6 + 4)^2 + (6 + 6)^2 + (4 + 6)^2 = 508$.

The cost of the array b equals to $(3 + 7)^2 + (3 + 6)^2 + (3 + 1)^2 + (7 + 6)^2 + (7 + 1)^2 + (6 + 1)^2 = 479$.

The total cost of two arrays equals to $508 + 479 = 987$.

E. Best Pair

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

You are given an array a of length n. Let cnt_x be the number of elements from the array which are equal to x. Let's also define $f(x, y)$ as $(cnt_x + cnt_y) \cdot (x + y)$.

Also you are given m bad pairs (x_i, y_i). Note that if (x, y) is a bad pair, then (y, x) is also bad.

Your task is to find the maximum value of $f(u, v)$ over all pairs (u, v), such that $u \neq v$, that this pair is not bad, and also that u and v each occur in the array a. It is guaranteed that such a pair exists.

Input

The first line contains a single integer t ($1 \leq t \leq 10\,000$) — the number of test cases.

The first line of each test case contains two integers n and m ($2 \leq n \leq 3 \cdot 10^5$, $0 \leq m \leq 3 \cdot 10^5$) — the length of the array and the number of bad pairs.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq 10^9$) — elements of the array.

The i-th of the next m lines contains two integers x_i and y_i ($1 \leq x_i < y_i \leq 10^9$), which represent a bad pair. It is guaranteed that no bad pair occurs twice in the input. It is also guaranteed that $cnt_{x_i} > 0$ and $cnt_{y_i} > 0$.

It is guaranteed that for each test case there is a pair of integers (u, v), $u \neq v$, that is not bad, and such that both of these numbers occur in a.

It is guaranteed that the total sum of n and the total sum of m don't exceed $3 \cdot 10^5$.

Output

For each test case print a single integer — the answer to the problem.

Example

input

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

output

```
40
14
15
```

Note

In the first test case 3, 6, 7 occur in the array.

- $f(3, 6) = (cnt_3 + cnt_6) \cdot (3 + 6) = (3 + 2) \cdot (3 + 6) = 45$. But $(3, 6)$ is bad so we ignore it.
- $f(3, 7) = (cnt_3 + cnt_7) \cdot (3 + 7) = (3 + 1) \cdot (3 + 7) = 40$.
- $f(6, 7) = (cnt_6 + cnt_7) \cdot (6 + 7) = (2 + 1) \cdot (6 + 7) = 39$.

The answer to the problem is $\max(40, 39) = 40$.

F. Towers

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

You are given a tree with n vertices numbered from 1 to n. The height of the i-th vertex is h_i. You can place any number of towers into vertices, for each tower you can choose which vertex to put it in, as well as choose its efficiency. Setting up a tower with efficiency e costs e coins, where $e > 0$.

It is considered that a vertex x gets a signal if for some pair of towers at the vertices u and v ($u \neq v$, but it is allowed that $x = u$ or $x = v$) with efficiencies e_u and e_v, respectively, it is satisfied that $\min(e_u, e_v) \geq h_x$ and x lies on the path between u and v.

Find the minimum number of coins required to set up towers so that you can get a signal at all vertices.

Input
The first line contains an integer n ($2 \leq n \leq 200\,000$) — the number of vertices in the tree.

The second line contains n integers h_i ($1 \leq h_i \leq 10^9$) — the heights of the vertices.

Each of the next $n - 1$ lines contain a pair of numbers v_i, u_i ($1 \leq v_i, u_i \leq n$) — an edge of the tree. It is guaranteed that the given edges form a tree.

Output
Print one integer — the minimum required number of coins.

Examples

input
3 1 2 1 1 2 2 3
output
4

input
5 1 3 3 1 3 1 3 5 4 4 3 2 3
output
7

input
2 6 1 1 2
output
12

Note
In the first test case it's optimal to install two towers with efficiencies 2 at vertices 1 and 3.

In the second test case it's optimal to install a tower with efficiency 1 at vertex 1 and two towers with efficiencies 3 at vertices 2 and 5.

In the third test case it's optimal to install two towers with efficiencies 6 at vertices 1 and 2.

G. Birthday

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

Vitaly gave Maxim n numbers $1, 2, \ldots, n$ for his 16-th birthday. Maxim was tired of playing board games during the celebration, so he decided to play with these numbers. In one step Maxim can choose two numbers x and y from the numbers he has, throw them away, and add two numbers $x + y$ and $|x - y|$ instead. He wants all his numbers to be equal after several steps and the sum of the numbers to be minimal.

Help Maxim to find a solution. Maxim's friends don't want to wait long, so the number of steps in the solution should not exceed $20n$. It is guaranteed that under the given constraints, if a solution exists, then there exists a solution that makes all numbers equal, minimizes their sum, and spends no more than $20n$ moves.

Input
The first line contains a single integer t ($1 \le t \le 25\,000$) — the number of test cases.

Each test case contains a single integer n ($2 \le n \le 5 \cdot 10^4$) — the number of integers given to Maxim.

It is guaranteed that the total sum of n doesn't exceed $5 \cdot 10^4$.

Output
For each test case print -1 if it's impossible to make all numbers equal.

Otherwise print a single integer s ($0 \le s \le 20n$) — the number of steps. Then print s lines. The i-th line must contain two integers x_i and y_i — numbers that Maxim chooses on the i-th step. The numbers must become equal after all operations.

Don't forget that you not only need to make all numbers equal, but also minimize their sum. It is guaranteed that under the given constraints, if a solution exists, then there exists a solution that makes all numbers equal, minimizes their sum, and spends no more than $20n$ moves.

Example

input
2 2 3
output
-1 3 1 3 2 2 4 0

H. Minimize Inversions Number

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

You are given a permutation p of length n.

You can choose any subsequence, remove it from the permutation, and insert it at the beginning of the permutation keeping the same order.

For every k from 0 to n, find the minimal possible number of inversions in the permutation after you choose a subsequence of length exactly k.

Input
The first line contains a single integer t ($1 \le t \le 50\,000$) — the number of test cases.

The first line of each test case contains one integer n ($1 \le n \le 5 \cdot 10^5$) — the length of the permutation.

The second line of each test case contains the permutation $p_1, p_2, \ldots, p_n$ ($1 \le p_i \le n$).

It is guaranteed that the total sum of n doesn't exceed $5 \cdot 10^5$.

Output
For each test case output $n + 1$ integers. The i-th of them must be the answer for the subsequence length of $i - 1$.

Example

input
3 1 1 4 4 2 1 3 5 5 1 3 2 4
output
0 0 4 2 2 1 4 5 4 2 2 1 5

Note
In the second test case:

- For the length 0: $[4, 2, 1, 3] \to [4, 2, 1, 3]$: 4 inversions.
- For the length 1: $[4, 2, \mathbf{1}, 3] \to [1, 4, 2, 3]$: 2 inversions.
- For the length 2: $[4, \mathbf{2}, \mathbf{1}, 3] \to [2, 1, 4, 3]$, or $[4, 2, \mathbf{1}, \mathbf{3}] \to [1, 3, 4, 2]$: 2 inversions.
- For the length 3: $[4, \mathbf{2}, \mathbf{1}, \mathbf{3}] \to [2, 1, 3, 4]$: 1 inversion.
- For the length 4: $[\mathbf{4}, \mathbf{2}, \mathbf{1}, \mathbf{3}] \to [4, 2, 1, 3]$: 4 inversions.
