A. Puzzle From the Future

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

In the 2022 year, Mike found two binary integers a and b of length n (both of them are written only by digits 0 and 1) that can have leading zeroes. In order not to forget them, he wanted to construct integer d in the following way:

- he creates an integer c as a result of bitwise summing of a and b *without transferring carry*, so c may have one or more 2-s. For example, the result of bitwise summing of 0110 and 1101 is 1211 or the sum of 011000 and 011000 is 022000;
- after that Mike replaces equal consecutive digits in c by one digit, thus getting d. In the cases above after this operation, 1211 becomes 121 and 022000 becomes 020 (so, d won't have equal consecutive digits).

Unfortunately, Mike lost integer a before he could calculate d himself. Now, to cheer him up, you want to find **any binary** integer a of length n such that d will be **maximum possible as integer**.

Maximum possible as integer means that $102 > 21$, $012 < 101$, $021 = 21$ and so on.

Input

The first line contains a single integer t ($1 \le t \le 1000$) — the number of test cases.

The first line of each test case contains the integer n ($1 \le n \le 10^5$) — the length of a and b.

The second line of each test case contains binary integer b of length n. The integer b consists only of digits 0 and 1.

It is guaranteed that the total sum of n over all t test cases doesn't exceed 10^5.

Output

For each test case output one **binary** integer a of length n. Note, that a or b may have leading zeroes but must have the same length n.

Example

input
5 1 0 3 011 3 110 6 111000 6 001011
output
1 110 100 101101 101110

Note

In the first test case, $b = 0$ and choosing $a = 1$ gives $d = 1$ as a result.

In the second test case, $b = 011$ so:

- if you choose $a = 000$, c will be equal to 011, so $d = 01$;
- if you choose $a = 111$, c will be equal to 122, so $d = 12$;
- if you choose $a = 010$, you'll get $d = 021$.
- If you select $a = 110$, you'll get $d = 121$.

We can show that answer $a = 110$ is optimal and $d = 121$ is maximum possible.
In the third test case, $b = 110$. If you choose $a = 100$, you'll get $d = 210$ and it's the maximum possible d.

In the fourth test case, $b = 111000$. If you choose $a = 101101$, you'll get $d = 212101$ and it's maximum possible d.

In the fifth test case, $b = 001011$. If you choose $a = 101110$, you'll get $d = 102121$ and it's maximum possible d.

B. Different Divisors

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

Positive integer x is called *divisor* of positive integer y, if y is divisible by x without remainder. For example, 1 is a divisor of 7 and 3 is not divisor of 8.

We gave you an integer d and asked you to find **the smallest** positive integer a, such that

- a has at least 4 divisors;
- difference between any two divisors of a is at least d.

Input
The first line contains a single integer t ($1 \le t \le 3000$) — the number of test cases.

The first line of each test case contains a single integer d ($1 \le d \le 10000$).

Output
For each test case print one integer a — the answer for this test case.

Example

input
2 1 2
output
6 15

Note
In the first test case, integer 6 have following divisors: $[1, 2, 3, 6]$. There are 4 of them and the difference between any two of them is at least 1. There is no smaller integer with at least 4 divisors.

In the second test case, integer 15 have following divisors: $[1, 3, 5, 15]$. There are 4 of them and the difference between any two of them is at least 2.

The answer 12 is INVALID because divisors are $[1, 2, 3, 4, 6, 12]$. And the difference between, for example, divisors 2 and 3 is less than $d = 2$.

C. Array Destruction

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

You found a useless array a of $2n$ positive integers. You have realized that you actually don't need this array, so you decided to throw out all elements of a.

It could have been an easy task, but it turned out that you should follow some rules:

1. In the beginning, you select any positive integer x.
2. Then you do the following operation n times:
    - select two elements of array with sum equals x;
    - remove them from a and replace x with maximum of that two numbers.

For example, if initially $a = [3, 5, 1, 2]$, you can select $x = 6$. Then you can select the second and the third elements of a with sum $5 + 1 = 6$ and throw them out. After this operation, x equals 5 and there are two elements in array: 3 and 2. You can throw them out on the next operation.

Note, that you choose x before the start and can't change it as you want between the operations.

Determine how should you behave to throw out all elements of a.

Input
The first line contains a single integer t ($1 \le t \le 1000$) — the number of test cases.

The first line of each test case contains the single integer n ($1 \le n \le 1000$).

The second line of each test case contains $2n$ integers $a_1, a_2, \ldots, a_{2n}$ ($1 \le a_i \le 10^6$) — the initial array a.

It is guaranteed that the total sum of n over all test cases doesn't exceed 1000.

Output
For each test case in the first line print YES if it is possible to throw out all elements of the array and NO otherwise.

If it is possible to throw out all elements, print the initial value of x you've chosen. Print description of n operations next. For each operation, print the pair of integers you remove.

Example

input
4 2 3 5 1 2 3 1 1 8 8 64 64 2 1 1 2 4 5 1 2 3 4 5 6 7 14 3 11
output
YES 6 1 5 2 3 NO NO YES 21 14 7 3 11 5 6 2 4 3 1

Note
The first test case was described in the statement.

In the second and third test cases, we can show that it is impossible to throw out all elements of array a.

D. Cleaning

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

During cleaning the coast, Alice found n piles of stones. The i-th pile has a_i stones.

Piles i and $i + 1$ are neighbouring for all $1 \leq i \leq n - 1$. If pile i becomes empty, piles $i - 1$ and $i + 1$ **doesn't** become neighbouring.

Alice is too lazy to remove these stones, so she asked you to take this duty. She allowed you to do only the following operation:

- Select two **neighboring** piles and, if both of them are not empty, remove one stone from each of them.

Alice understands that sometimes it's impossible to remove all stones with the given operation, so she allowed you to use the following superability:

- Before the start of cleaning, you can select two **neighboring** piles and swap them.

Determine, if it is possible to remove all stones using the superability **not more than once**.

Input
The first line contains a single integer t ($1 \leq t \leq 10^4$) — the number of test cases.

The first line of each test case contains the single integer n ($2 \leq n \leq 2 \cdot 10^5$) — the number of piles.

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

It is guaranteed that the total sum of n over all test cases doesn't exceed $2 \cdot 10^5$.

Output
For each test case, print YES or NO — is it possible to remove all stones using the superability **not more than once** or not.

Example

input
5 3 1 2 1 3 1 1 2 5

```
2 2 2 1 3
5
2100 1900 1600 3000 1600
2
2443 2445
```

output

```
YES
YES
YES
YES
NO
```

Note

In the first test case, you can remove all stones without using a superability: $[1, 2, 1] \to [1, 1, 0] \to [0, 0, 0]$.

In the second test case, you can apply superability to the second and the third piles and then act like in the first testcase.

In the third test case, you can apply superability to the fourth and the fifth piles, thus getting $a = [2, 2, 2, 3, 1]$.

In the fourth test case, you can apply superability to the first and the second piles, thus getting $a = [1900, 2100, 1600, 3000, 1600]$.

E. What Is It?

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

Lunar rover finally reached planet X. After landing, he met an obstacle, that contains permutation p of length n. Scientists found out, that to overcome an obstacle, the robot should make p an identity permutation (make $p_i = i$ for all i).

Unfortunately, scientists can't control the robot. Thus the only way to make p an identity permutation is applying the following operation to p multiple times:

- Select two indices i and j ($i \neq j$), such that $p_j = i$ and swap the values of p_i and p_j. It takes robot $(j - i)^2$ seconds to do this operation.

Positions i and j are selected by the robot (scientists can't control it). He will apply this operation while p isn't an identity permutation. We can show that the robot will make no more than n operations regardless of the choice of i and j on each operation. Scientists asked you to find out the maximum possible time it will take the robot to finish making p an identity permutation (i. e. worst-case scenario), so they can decide whether they should construct a new lunar rover or just rest and wait. They won't believe you without proof, so you should build an example of p and robot's operations that maximizes the answer.

For a better understanding of the statement, read the sample description.

Input

The first line of input contains a single integer t ($1 \leq t \leq 10^4$) — the number of test cases.

Each of next t lines contains the single integer n ($2 \leq n \leq 10^5$) – the length of p.

Note, that p is not given to you. You should find the maximum possible time over all permutations of length n.

It is guaranteed, that the total sum of n over all test cases doesn't exceed 10^5.

Output

For each test case in the first line, print how many seconds will the robot spend in the worst case.

In the next line, print the initial value of p that you used to construct an answer.

In the next line, print the number of operations $m \leq n$ that the robot makes in your example.

In the each of next m lines print two integers i and j — indices of positions that the robot will swap on this operation. Note that $p_j = i$ must holds (at the time of operation).

Example

input

```
3
2
3
3
```

output

```
1
2 1
1
2 1
5
2 3 1
2
1 3
```

```
3 2
5
2 3 1
2
1 3
2 3
```

Note

For $n = 2$, p can be either $[1, 2]$ or $[2, 1]$. In the first case p is already identity, otherwise robot will make it an identity permutation in 1 second regardless of choise i and j on the first operation.

For $n = 3$, p can be equals $[2, 3, 1]$.

- If robot will select $i = 3, j = 2$ on the first operation, p will become $[2, 1, 3]$ in one second. Now robot can select only $i = 1, j = 2$ or $i = 2, j = 1$. In both cases, p will become identity in one more second (2 seconds in total).
- If robot will select $i = 1, j = 3$ on the first operation, p will become $[1, 3, 2]$ in four seconds. Regardless of choise of i and j on the second operation, p will become identity in five seconds.

We can show, that for permutation of length 3 robot will always finish all operation in no more than 5 seconds.

F. 1 2 3 4 ...

time limit per test: 5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Igor had a sequence $d_1, d_2, \ldots, d_n$ of integers. When Igor entered the classroom there was an integer x written on the blackboard.

Igor generated sequence p using the following algorithm:

1. initially, $p = [x]$;
2. for each $1 \le i \le n$ he did the following operation $|d_i|$ times:
   - if $d_i \ge 0$, then he looked at the last element of p (let it be y) and appended $y + 1$ to the end of p;
   - if $d_i < 0$, then he looked at the last element of p (let it be y) and appended $y - 1$ to the end of p.

For example, if $x = 3$, and $d = [1, -1, 2]$, p will be equal $[3, 4, 3, 4, 5]$.

Igor decided to calculate the length of the longest increasing subsequence of p and the number of them.

A sequence a is a subsequence of a sequence b if a can be obtained from b by deletion of several (possibly, zero or all) elements.

A sequence a is an increasing sequence if each element of a (except the first one) is strictly greater than the previous element.

For $p = [3, 4, 3, 4, 5]$, the length of longest increasing subsequence is 3 and there are 3 of them: $[\underline{3}, \underline{4}, 3, 4, \underline{5}]$, $[\underline{3}, 4, 3, \underline{4}, \underline{5}]$, $[3, 4, \underline{3}, \underline{4}, \underline{5}]$.

Input

The first line contains a single integer n ($1 \le n \le 50$) — the length of the sequence d.

The second line contains a single integer x ($-10^9 \le x \le 10^9$) — the integer on the blackboard.

The third line contains n integers $d_1, d_2, \ldots, d_n$ ($-10^9 \le d_i \le 10^9$).

Output

Print two integers:

- the first integer should be equal to the length of the longest increasing subsequence of p;
- the second should be equal to the number of them modulo 998244353.

You should print *only the second number* modulo 998244353.

Examples

input
3 3 1 -1 2
output
3 3

input
3 100 5 -3 6
output

9 7

input
3 1 999999999 0 1000000000
output
2000000000 1

input
5 34 1337 -146 42 -69 228
output
1393 3876

Note

The first test case was explained in the statement.

In the second test case $p = [100, 101, 102, 103, 104, 105, 104, 103, 102, 103, 104, 105, 106, 107, 108]$.

In the third test case $p = [1, 2, \ldots, 2000000000]$.