

## Codeforces Round #794 (Div. 2)

### A. Everything Everywhere All But One

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

You are given an array of  $n$  integers  $a_1, a_2, \dots, a_n$ . After you watched the amazing film "Everything Everywhere All At Once", you came up with the following operation.

In one operation, you choose  $n - 1$  elements of the array and replace each of them with their arithmetic mean (which doesn't have to be an integer). For example, from the array  $[1, 2, 3, 1]$  we can get the array  $[2, 2, 2, 1]$ , if we choose the first three elements, or we can get the array  $[\frac{4}{3}, \frac{4}{3}, \frac{4}{3}, 3]$ , if we choose all elements except the third.

Is it possible to make all elements of the array equal by performing a finite number of such operations?

#### Input

The first line of the input contains a single integer  $t$  ( $1 \leq t \leq 200$ ) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer  $n$  ( $3 \leq n \leq 50$ ) — the number of integers.

The second line of each test case contains  $n$  integers  $a_1, a_2, \dots, a_n$  ( $0 \leq a_i \leq 100$ ).

#### Output

For each test case, if it is possible to make all elements equal after some number of operations, output **YES**. Otherwise, output **NO**.

You can output **YES** and **NO** in any case (for example, strings **yEs**, **yes**, **Yes** will be recognized as a positive response).

#### Example

input
4 3 42 42 42 5 1 2 3 4 5 4 4 3 2 1 3 24 2 22
output
YES YES NO NO

#### Note

In the first test case, all elements are already equal.

In the second test case, you can choose all elements except the third, their average is  $\frac{1+2+4+5}{4} = 3$ , so the array will become  $[3, 3, 3, 3]$ .

It's possible to show that it's impossible to make all elements equal in the third and fourth test cases.

### B. Odd Subarrays

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

For an array  $[b_1, b_2, \dots, b_m]$  define its number of inversions as the number of pairs  $(i, j)$  of integers such that  $1 \leq i < j \leq m$  and  $b_i > b_j$ . Let's call array  $b$  **odd** if its number of inversions is odd.

For example, array  $[4, 2, 7]$  is odd, as its number of inversions is 1, while array  $[2, 1, 4, 3]$  isn't, as its number of inversions is 2.

You are given a permutation  $[p_1, p_2, \dots, p_n]$  of integers from 1 to  $n$  (each of them appears exactly once in the permutation). You want to split it into several consecutive subarrays (maybe just one), so that the number of the **odd** subarrays among them is as large as possible.

What largest number of these subarrays may be **odd**?

Input

The first line of the input contains a single integer  $t$  ( $1 \leq t \leq 10^5$ ) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer  $n$  ( $1 \leq n \leq 10^5$ ) — the size of the permutation.

The second line of each test case contains  $n$  integers  $p_1, p_2, \dots, p_n$  ( $1 \leq p_i \leq n$ , all  $p_i$  are distinct) — the elements of the permutation.

The sum of  $n$  over all test cases doesn't exceed  $2 \cdot 10^5$ .

Output

For each test case output a single integer — the largest possible number of **odd** subarrays that you can get after splitting the permutation into several consecutive subarrays.

Example

input
5 3 1 2 3 4 4 3 2 1 2 1 2 2 2 1 6 4 5 6 1 2 3
output
0 2 0 1 1

Note

In the first and third test cases, no matter how we split our permutation, there won't be any odd subarrays.

In the second test case, we can split our permutation into subarrays  $[4, 3]$ ,  $[2, 1]$ , both of which are odd since their numbers of inversions are 1.

In the fourth test case, we can split our permutation into a single subarray  $[2, 1]$ , which is odd.

In the fifth test case, we can split our permutation into subarrays  $[4, 5]$ ,  $[6, 1, 2, 3]$ . The first subarray has 0 inversions, and the second has 3, so it is odd.

C. Circular Local MiniMax

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

You are given  $n$  integers  $a_1, a_2, \dots, a_n$ . Is it possible to arrange them on a circle so that each number is strictly greater than both its neighbors or strictly smaller than both its neighbors?

In other words, check if there exists a rearrangement  $b_1, b_2, \dots, b_n$  of the integers  $a_1, a_2, \dots, a_n$  such that for each  $i$  from 1 to  $n$  at least one of the following conditions holds:

- $b_{i-1} < b_i > b_{i+1}$
- $b_{i-1} > b_i < b_{i+1}$

To make sense of the previous formulas for  $i = 1$  and  $i = n$ , one shall define  $b_0 = b_n$  and  $b_{n+1} = b_1$ .

Input

The first line of the input contains a single integer  $t$  ( $1 \leq t \leq 3 \cdot 10^4$ ) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer  $n$  ( $3 \leq n \leq 10^5$ ) — the number of integers.

The second line of each test case contains  $n$  integers  $a_1, a_2, \dots, a_n$  ( $0 \leq a_i \leq 10^9$ ).

The sum of  $n$  over all test cases doesn't exceed  $2 \cdot 10^5$ .

Output

For each test case, if it is not possible to arrange the numbers on the circle satisfying the conditions from the statement, output **NO**.

You can output each letter in any case.

Otherwise, output **YES**. In the second line, output  $n$  integers  $b_1, b_2, \dots, b_n$ , which are a rearrangement of  $a_1, a_2, \dots, a_n$  and satisfy the conditions from the statement. If there are multiple valid ways to arrange the numbers, you can output any of them.

Example

input
4 3 1 1 2 4 1 9 8 4 4 2 0 2 2 6 1 1 1 11 111 1111
output
NO YES 1 8 4 9 NO YES 1 11 1 111 1 1111

Note

It can be shown that there are no valid arrangements for the first and the third test cases.

In the second test case, the arrangement  $[1, 8, 4, 9]$  works. In this arrangement, 1 and 4 are both smaller than their neighbors, and 8, 9 are larger.

In the fourth test case, the arrangement  $[1, 11, 1, 111, 1, 1111]$  works. In this arrangement, the three elements equal to 1 are smaller than their neighbors, while all other elements are larger than their neighbors.

D. Linguistics

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

Alina has discovered a weird language, which contains only 4 words: **A**, **B**, **AB**, **BA**. It also turned out that there are no spaces in this language: a sentence is written by just concatenating its words into a single string.

Alina has found one such sentence  $s$  and she is curious: is it possible that it consists of precisely  $a$  words **A**,  $b$  words **B**,  $c$  words **AB**, and  $d$  words **BA**?

In other words, determine, if it's possible to concatenate these  $a + b + c + d$  words in some order so that the resulting string is  $s$ . Each of the  $a + b + c + d$  words must be used exactly once in the concatenation, but you can choose the order in which they are concatenated.

Input

The first line of the input contains a single integer  $t$  ( $1 \leq t \leq 10^5$ ) — the number of test cases. The description of the test cases follows.

The first line of each test case contains four integers  $a, b, c, d$  ( $0 \leq a, b, c, d \leq 2 \cdot 10^5$ ) — the number of times that words **A**, **B**, **AB**, **BA** respectively must be used in the sentence.

The second line contains the string  $s$  ( $s$  consists only of the characters **A** and **B**,  $1 \leq |s| \leq 2 \cdot 10^5$ ,  $|s| = a + b + 2c + 2d$ ) — the sentence. Notice that the condition  $|s| = a + b + 2c + 2d$  (here  $|s|$  denotes the length of the string  $s$ ) is equivalent to the fact that  $s$  is as long as the concatenation of the  $a + b + c + d$  words.

The sum of the lengths of  $s$  over all test cases doesn't exceed  $2 \cdot 10^5$ .

Output

For each test case output **YES** if it is possible that the sentence  $s$  consists of precisely  $a$  words **A**,  $b$  words **B**,  $c$  words **AB**, and  $d$  words **BA**, and **NO** otherwise. You can output each letter in any case.

Example

input
8 1 0 0 0 B 0 0 1 0 AB 1 1 0 1 ABAB 1 0 1 1 ABAAB 1 1 2 2

BAABBABBA  
1 1 2 3  
ABABABBAABAB  
2 3 5 4  
AABAABBABAAABABBABBBABB  
1 3 3 10  
BBABABABABBBABABABABABAABABA

output

NO  
YES  
YES  
YES  
YES  
YES  
NO  
YES

Note

In the first test case, the sentence  $s$  is **B**. Clearly, it can't consist of a single word **A**, so the answer is **NO**.

In the second test case, the sentence  $s$  is **AB**, and it's possible that it consists of a single word **AB**, so the answer is **YES**.

In the third test case, the sentence  $s$  is **ABAB**, and it's possible that it consists of one word **A**, one word **B**, and one word **BA**, as  $A + BA + B = ABAB$ .

In the fourth test case, the sentence  $s$  is **ABAAB**, and it's possible that it consists of one word **A**, one word **AB**, and one word **BA**, as  $A + BA + AB = ABAAB$ .

In the fifth test case, the sentence  $s$  is **BAABBABBA**, and it's possible that it consists of one word **A**, one word **B**, two words **AB**, and two words **BA**, as  $BA + AB + B + AB + BA + A = BAABBABBA$ .

E. Bring Balance

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

Alina has a bracket sequence  $s$  of length  $2n$ , consisting of  $n$  opening brackets ' (' and  $n$  closing brackets ')' '. As she likes balance, she wants to turn this bracket sequence into a balanced bracket sequence.

In one operation, she can reverse any substring of  $s$ .

What's the smallest number of operations that she needs to turn  $s$  into a balanced bracket sequence? It can be shown that it's always possible in at most  $n$  operations.

As a reminder, a sequence of brackets is called balanced if one can turn it into a valid math expression by adding characters + and 1. For example, sequences  $(( ))()$ ,  $()$ , and  $(( )(( )))$  are balanced, while  $)()$ ,  $((())$ , and  $(( ))()$  are not.

Input

The first line of the input contains a single integer  $t$  ( $1 \leq t \leq 2 \cdot 10^4$ ) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer  $n$  ( $1 \leq n \leq 10^5$ ).

The second line of each test case contains a string  $s$  of length  $2n$ , consisting of  $n$  opening and  $n$  closing brackets.

The sum of  $n$  over all test cases doesn't exceed  $2 \cdot 10^5$ .

Output

For each test case, in the first line output a single integer  $k$  ( $0 \leq k \leq n$ ) — the smallest number of operations required.

The  $i$ -th of the next  $k$  lines should contain two integers  $l_i, r_i$  ( $1 \leq l_i \leq r_i \leq 2n$ ), indicating that in the  $i$ -th operation, Alina will reverse the substring  $s_{l_i} s_{l_i+1} \dots s_{r_i-1} s_{r_i}$ . Here the numeration starts from 1.

If there are multiple sequences of operations with the smallest length which transform the sequence into a balanced one, you can output any of them.

Example

input
3 2 (()) 5 (())(())( 6 (())(())(
output
0

2  
3 4  
9 10  
1  
2 11

Note

In the first test case, the string is already balanced.

In the second test case, the string will be transformed as follows:  $()()((()))() \rightarrow ()()((()))() \rightarrow ()()((()))()$ , where the last string is balanced.

In the third test case, the string will be transformed to  $((()))((()))$ , which is balanced.

F. Permutation Weight (Easy Version)

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

This is an easy version of the problem. The difference between the easy and hard versions is that in this version, you can output any permutation with the smallest weight.

You are given a permutation  $p_1, p_2, \dots, p_n$  of integers from 1 to  $n$ .

Let's define the weight of the permutation  $q_1, q_2, \dots, q_n$  of integers from 1 to  $n$  as

$$|q_1 - p_{q_2}| + |q_2 - p_{q_3}| + \dots + |q_{n-1} - p_{q_n}| + |q_n - p_{q_1}|$$

You want your permutation to be as lightweight as possible. Find any permutation  $q$  with the smallest possible weight.

Input

The first line of the input contains a single integer  $t$  ( $1 \leq t \leq 100$ ) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer  $n$  ( $2 \leq n \leq 200$ ) — the size of the permutation.

The second line of each test case contains  $n$  integers  $p_1, p_2, \dots, p_n$  ( $1 \leq p_i \leq n$ , all  $p_i$  are distinct) — the elements of the permutation.

The sum of  $n$  over all test cases doesn't exceed 400.

Output

For each test case, output  $n$  integers  $q_1, q_2, \dots, q_n$  ( $1 \leq q_i \leq n$ , all  $q_i$  are distinct) — one of the permutations with the smallest weight.

Example

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

Note

In the first test case, there are two permutations of length 2:  $(1, 2)$  and  $(2, 1)$ . Permutation  $(1, 2)$  has weight  $|1 - p_2| + |2 - p_1| = 0$ , and permutation  $(2, 1)$  has the same weight:  $|2 - p_1| + |1 - p_2| = 0$ . You can output any of these permutations in this version.

In the second test case, the weight of the permutation  $(1, 3, 4, 2)$  is  $|1 - p_3| + |3 - p_4| + |4 - p_2| + |2 - p_1| = |1 - 1| + |3 - 4| + |4 - 3| + |2 - 2| = 2$ . There are no permutations with smaller weights.

In the third test case, the weight of the permutation  $(1, 4, 2, 3, 5)$  is  $|1 - p_4| + |4 - p_2| + |2 - p_3| + |3 - p_5| + |5 - p_1| = |1 - 2| + |4 - 4| + |2 - 3| + |3 - 1| + |5 - 5| = 4$ . There are no permutations with smaller weights.