Heuristic Knapsack

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

In computer science, the 0/1 Knapsack problem is a classic challenge: given a set of items, each with a weight and a value, choose a subset of items so that the total weight does not exceed a given limit, and the total value is as large as possible.

This problem is NP-hard. When the range of weights is relatively small, it can be solved exactly using the well-known dynamic programming algorithm. However, when both the weight and value ranges are very large, up to 10^9 for example, heuristic approaches are needed to obtain good (though not necessarily optimal) solutions.

Two researchers, Alice and Bob, are studying simple and fast "heuristic" algorithms that give good results for such large-scale knapsack instances. Each of them uses a greedy strategy:

- **Alice's Strategy: Lightest First**. Alice wants to keep the knapsack as light as possible. She sorts all items by weight in **ascending order**. If two items have the same weight, she breaks ties by their original index in ascending order. Then she takes a prefix of this sorted list: she picks items one by one until the next item in her order does not fit into the remaining capacity.
- **Bob's Strategy: Best Value First**. Bob wants to get the most value. He sorts all items by value in **descending order**. If two items have the same value, he breaks ties by their original index in ascending order. Then he goes through this list. For each item, if it fits in the remaining capacity, he takes it; otherwise, he skips it and continues.

Now, Alice and Bob are given a dataset of n items indexed from 1 to n and a knapsack with capacity W. However, some data is missing. For each item, its weight w_i and value r_i are given, but **at most one** of these two values might be unknown.

Your task is to assign a positive integer to each unknown value so that the set of items chosen by Alice is **exactly the same** as the set of items chosen by Bob (the order does not matter).

Input

The first line contains an integer T ($1 \le T \le 100$), denoting the number of test cases.

For each of the test cases, the first line inputs two integers n, W ($1 \le n \le 3000$, $1 \le W \le 10^9$), denoting the number of items and the knapsack capacity.

The second line contains n integers $w_1, w_2, \ldots, w_n$ ($0 \le w_i \le 10^9$), denoting the weights of the items. If $w_i = 0$, this parameter is unknown.

The third line contains n integers $r_1, r_2, \ldots, r_n$ ($0 \le r_i \le 10^9$), denoting the values of the items. If $r_i = 0$, this parameter is unknown. It is guaranteed that for each item, at most one of the two values is unknown.

It is guaranteed that $\sum n \le 3000$ over all test cases.

Output

For each test case, if there exists a valid assignment of positive integers to the unknown values that makes Alice's and Bob's chosen item sets identical, print `Yes` in the first line. Then print all w_i ($1 \le w_i \le 10^9$) in the second line separated by spaces, and print all r_i ($1 \le r_i \le 10^9$) in the third line separated by spaces, which denotes a possible valid assignment. Otherwise, print `No` in a single line.

Note that the maximum limits of w_i and r_i are both 10^9, **as same as** the input limits. Either `Yes` or `No` is case-insensitive, which means you can print `YeS`, `yEs`, `nO`, etc.

Example

standard input

```
6
3 5
0 2 3
2 4 0
3 5
2 3 0
4 0 2
3 1000000000
0 0 1000000000
2 3 1
5 1
0 1 1 1 0
3 8 0 7 9
5 1000000000
0 0 0 1 0
1 9 3 9 10
3 1000000000
500000001 500000000 0
1000000000 999999998 1
```

standard output

```
Yes
3 2 3
2 4 1
Yes
2 3 2
4 1 2
Yes
1000000000 999999999 1000000000
2 3 1
Yes
1000000000 1 1 1 1000000000
3 8 1 7 9
Yes
1000000000 1000000000 1000000000 1 999999999
1 9 3 9 10
No
```

Note

For the first test case of the example, Alice takes the second item and then the first; Bob also takes the second and then the first.

For the second test case, Alice and Bob both take the first and the third items.

For the last test case, there is no solution.