Educational Codeforces Round 87 (Rated for Div. 2)

A. Alarm Clock

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

Polycarp has spent the entire day preparing problems for you. Now he has to sleep for at least a minutes to feel refreshed.

Polycarp can only wake up by hearing the sound of his alarm. So he has just fallen asleep and his first alarm goes off in b minutes.

Every time Polycarp wakes up, he decides if he wants to sleep for some more time or not. If he's slept for less than a minutes in total, then he sets his alarm to go off in c minutes after it is reset and spends d minutes to fall asleep again. Otherwise, he gets out of his bed and proceeds with the day.

If the alarm goes off while Polycarp is falling asleep, then he resets his alarm to go off in another c minutes and tries to fall asleep for d minutes again.

You just want to find out when will Polycarp get out of his bed or report that it will never happen.

Please check out the notes for some explanations of the example.

Input

The first line contains one integer t ($1 \le t \le 1000$) — the number of testcases.

The only line of each testcase contains four integers a, b, c, d ($1 \le a, b, c, d \le 10^9$) — the time Polycarp has to sleep for to feel refreshed, the time before the first alarm goes off, the time before every succeeding alarm goes off and the time Polycarp spends to fall asleep.

Output

For each test case print one integer. If Polycarp never gets out of his bed then print -1. Otherwise, print the time it takes for Polycarp to get out of his bed.

Example

input
7 10 3 6 4 11 3 6 4 5 9 4 10 6 5 2 3 1 1 1 1 3947465 47342 338129 123123 234123843 13 361451236 361451000
output
27 27 9 -1 1 6471793 358578060125049

Note

In the first testcase Polycarp wakes up after 3 minutes. He only rested for 3 minutes out of 10 minutes he needed. So after that he sets his alarm to go off in 6 minutes and spends 4 minutes falling asleep. Thus, he rests for 2 more minutes, totaling in $3 + 2 = 5$ minutes of sleep. Then he repeats the procedure three more times and ends up with 11 minutes of sleep. Finally, he gets out of his bed. He spent 3 minutes before the first alarm and then reset his alarm four times. The answer is $3 + 4 \cdot 6 = 27$.

The second example is almost like the first one but Polycarp needs 11 minutes of sleep instead of 10. However, that changes nothing because he gets 11 minutes with these alarm parameters anyway.

In the third testcase Polycarp wakes up rested enough after the first alarm. Thus, the answer is $b = 9$.

In the fourth testcase Polycarp wakes up after 5 minutes. Unfortunately, he keeps resetting his alarm infinitely being unable to rest for even a single minute :(

B. Ternary String

time limit per test: 2 seconds

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

You are given a string s such that each its character is either 1, 2, or 3. You have to choose the shortest contiguous substring of s such that it contains each of these three characters at least once.

A contiguous substring of string s is a string that can be obtained from s by removing some (possibly zero) characters from the beginning of s and some (possibly zero) characters from the end of s.

Input
The first line contains one integer t ($1 \le t \le 20000$) — the number of test cases.

Each test case consists of one line containing the string s ($1 \le |s| \le 200000$). It is guaranteed that each character of s is either 1, 2, or 3.

The sum of lengths of all strings in all test cases does not exceed 200000.

Output
For each test case, print one integer — the length of the shortest contiguous substring of s containing all three types of characters at least once. If there is no such substring, print 0 instead.

Example

input
7 123 12222133333332 112233 332211 12121212 333333 31121
output
3 3 4 4 0 0 4

Note
Consider the example test:

In the first test case, the substring 123 can be used.

In the second test case, the substring 213 can be used.

In the third test case, the substring 1223 can be used.

In the fourth test case, the substring 3221 can be used.

In the fifth test case, there is no character 3 in s.

In the sixth test case, there is no character 1 in s.

In the seventh test case, the substring 3112 can be used.

C1. Simple Polygon Embedding

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

The statement of this problem is the same as the statement of problem C2. The only difference is that, in problem C1, n is always even, and in C2, n is always odd.

You are given a regular polygon with $2 \cdot n$ vertices (it's convex and has equal sides and equal angles) and all its sides have length 1. Let's name it as $2n$-gon.

Your task is to find the square of the minimum size such that you can embed $2n$-gon in the square. Embedding $2n$-gon in the square means that you need to place $2n$-gon in the square in such way that each point which lies inside or on a border of $2n$-gon should also lie inside or on a border of the square.

You can rotate $2n$-gon and/or the square.

Input
The first line contains a single integer T ($1 \le T \le 200$) — the number of test cases.

Next T lines contain descriptions of test cases — one per line. Each line contains single **even** integer n ($2 \le n \le 200$). Don't forget you need to embed $2n$-gon, not an n-gon.

Output
Print T real numbers — one per test case. For each test case, print the minimum length of a side of the square $2n$-gon can be embedded in. Your answer will be considered correct if its absolute or relative error doesn't exceed 10^{-6}.

Example

input
3 2 4 200
output
1.000000000 2.414213562 127.321336469

C2. Not So Simple Polygon Embedding

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

The statement of this problem is the same as the statement of problem C1. The only difference is that, in problem C1, n is always even, and in C2, n is always odd.

You are given a regular polygon with $2 \cdot n$ vertices (it's convex and has equal sides and equal angles) and all its sides have length 1. Let's name it as $2n$-gon.

Your task is to find the square of the minimum size such that you can embed $2n$-gon in the square. Embedding $2n$-gon in the square means that you need to place $2n$-gon in the square in such way that each point which lies inside or on a border of $2n$-gon should also lie inside or on a border of the square.

You can rotate $2n$-gon and/or the square.

Input
The first line contains a single integer T ($1 \le T \le 200$) — the number of test cases.

Next T lines contain descriptions of test cases — one per line. Each line contains single **odd** integer n ($3 \le n \le 199$). Don't forget you need to embed $2n$-gon, not an n-gon.

Output
Print T real numbers — one per test case. For each test case, print the minimum length of a side of the square $2n$-gon can be embedded in. Your answer will be considered correct if its absolute or relative error doesn't exceed 10^{-6}.

Example

input
3 3 5 199
output
1.931851653 3.196226611 126.687663595

D. Multiset

time limit per test: 1.5 seconds
memory limit per test: 28 megabytes
input: standard input
output: standard output

Note that the memory limit is unusual.

You are given a multiset consisting of n integers. You have to process queries of two types:

- add integer k into the multiset;
- find the k-th order statistics in the multiset and remove it.

k-th order statistics in the multiset is the k-th element in the sorted list of all elements of the multiset. For example, if the multiset contains elements 1, 4, 2, 1, 4, 5, 7, and $k = 3$, then you have to find the 3-rd element in $[1, 1, 2, 4, 4, 5, 7]$, which is 2. If you try to delete an element which occurs multiple times in the multiset, only one occurence is removed.

After processing all queries, print **any** number belonging to the multiset, or say that it is empty.

Input
The first line contains two integers n and q ($1 \le n, q \le 10^6$) — the number of elements in the initial multiset and the number of queries, respectively.

The second line contains n integers a_1, a_2, ..., a_n ($1 \le a_1 \le a_2 \le \cdots \le a_n \le n$) — the elements of the multiset.

The third line contains q integers k_1, k_2, ..., k_q, each representing a query:

- if $1 \le k_i \le n$, then the i-th query is "insert k_i into the multiset";
- if $k_i < 0$, then the i-th query is "remove the $|k_i|$-th order statistics from the multiset". For this query, it is guaranteed that $|k_i|$ is not greater than the size of the multiset.

Output
If the multiset is empty after all queries, print 0.

Otherwise, print any integer that belongs to the resulting multiset.

Examples

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

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

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

Note
In the first example, all elements of the multiset are deleted.

In the second example, the elements 5, 1, 4, 2 are deleted (they are listed in chronological order of their removal).

In the third example, 6 is not the only answer.

E. Graph Coloring

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

You are given an undirected graph without self-loops or multiple edges which consists of n vertices and m edges. Also you are given three integers n_1, n_2 and n_3.

Can you label each vertex with one of three numbers 1, 2 or 3 in such way, that:

1. Each vertex should be labeled by exactly one number 1, 2 or 3;
2. The total number of vertices with label 1 should be equal to n_1;
3. The total number of vertices with label 2 should be equal to n_2;
4. The total number of vertices with label 3 should be equal to n_3;
5. $|col_u - col_v| = 1$ for each edge (u, v), where col_x is the label of vertex x.

If there are multiple valid labelings, print any of them.

Input
The first line contains two integers n and m ($1 \le n \le 5000$; $0 \le m \le 10^5$) — the number of vertices and edges in the graph.

The second line contains three integers n_1, n_2 and n_3 ($0 \le n_1, n_2, n_3 \le n$) — the number of labels 1, 2 and 3, respectively. It's guaranteed that $n_1 + n_2 + n_3 = n$.

Next m lines contan description of edges: the i-th line contains two integers u_i, v_i ($1 \le u_i, v_i \le n$; $u_i \ne v_i$) — the vertices the i-th edge connects. It's guaranteed that the graph doesn't contain self-loops or multiple edges.

Output
If valid labeling exists then print "YES" (without quotes) in the first line. In the second line print string of length n consisting of 1, 2 and 3. The i-th letter should be equal to the label of the i-th vertex.

If there is no valid labeling, print "NO" (without quotes).

Examples

input
6 3 2 2 2 3 1 5 4 2 5
output
YES 112323

input
5 9 0 2 3 1 2 1 3 1 5 2 3 2 4 2 5 3 4 3 5 4 5
output
NO

F. Summoning Minions

time limit per test: 6 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

Polycarp plays a computer game. In this game, the players summon armies of magical minions, which then fight each other.

Polycarp can summon n different minions. The initial power level of the i-th minion is a_i, and when it is summoned, all previously summoned minions' power levels are increased by b_i. The minions can be summoned in any order.

Unfortunately, Polycarp cannot have more than k minions under his control. To get rid of unwanted minions after summoning them, he may destroy them. Each minion can be summoned (and destroyed) only once.

Polycarp's goal is to summon the strongest possible army. Formally, he wants to maximize the sum of power levels of all minions under his control (those which are summoned and not destroyed).

Help Polycarp to make up a plan of actions to summon the strongest possible army!

Input
The first line contains one integer T ($1 \le T \le 75$) — the number of test cases.

Each test case begins with a line containing two integers n and k ($1 \le k \le n \le 75$) — the number of minions availible for summoning, and the maximum number of minions that can be controlled by Polycarp, respectively.

Then n lines follow, the i-th line contains 2 integers a_i and b_i ($1 \le a_i \le 10^5$, $0 \le b_i \le 10^5$) — the parameters of the i-th minion.

Output
For each test case print the optimal sequence of actions as follows:

Firstly, print m — the number of actions which Polycarp has to perform ($0 \le m \le 2n$). Then print m integers o_1, o_2, ..., o_m, where o_i denotes the i-th action as follows: if the i-th action is to summon the minion x, then $o_i = x$, and if the i-th action is to destroy the minion x, then $o_i = -x$. Each minion can be summoned at most once and cannot be destroyed before being summoned (and, obviously, cannot be destroyed more than once). The number of minions in Polycarp's army should be not greater than k after every action.

If there are multiple optimal sequences, print any of them.

Example

input
3

```
5 2
5 3
7 0
5 0
4 0
10 0
2 1
10 100
50 10
5 5
1 5
2 4
3 3
4 2
5 1
```

output

```
4
2 1 -1 5
1
2
5
5 4 3 2 1
```

Note

Consider the example test.

In the first test case, Polycarp can summon the minion 2 with power level 7, then summon the minion 1, which will increase the power level of the previous minion by 3, then destroy the minion 1, and finally, summon the minion 5. After this, Polycarp will have two minions with power levels of 10.

In the second test case, Polycarp can control only one minion, so he should choose the strongest of them and summon it.

In the third test case, Polycarp is able to summon and control all five minions.

G. Find a Gift

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

This is an interactive problem. Don't forget to flush output after printing queries using `cout.flush()` or `fflush(stdout)` in C++ or similar functions in other programming languages.

There are n gift boxes in a row, numbered from 1 to n from left to right. It's known that exactly k of them contain valuable gifts — other boxes contain just lucky stones. All boxes look the same and differ only in weight. All *boxes with stones have the same weight and are strictly heavier* than boxes with valuable items. But valuable gifts may be different, so the boxes with valuable items may have different weights.

You can ask no more than 50 queries (printing an answer doesn't count). By each query you can compare total weights of two non-intersecting subsets of boxes $a_1, a_2, \ldots, a_{k_a}$ and $b_1, b_2, \ldots, b_{k_b}$. In return you'll get one of four results:

- `FIRST`, if subset $a_1, a_2, \ldots, a_{k_a}$ is strictly **heavier**;
- `SECOND`, if subset $b_1, b_2, \ldots, b_{k_b}$ is strictly **heavier**;
- `EQUAL`, if subsets have equal total weights;
- `WASTED`, if the query is incorrect or the limit of queries is exceeded.

Using such queries (or, maybe, intuition) find the box with a valuable gift with **the minimum index**.

Input

The input consists of several cases. In the beginning, you receive the integer T ($1 \le T \le 500$) — the number of test cases.

At the beginning of each test case, you receive two integers n and k ($2 \le n \le 1000$, $1 \le k \le \frac{n}{2}$) — the number of boxes in a row and the number of boxes with valuable gifts.

It's guaranteed that the order of boxes is fixed beforehand and that the sum of n in one test doesn't exceed 1000.

Output

For each test case print the minimum index among all boxes with a valuable gift in the following format: "`!` x" where x ($1 \le x \le n$) — the index of the box.

Interaction

Print each query in three lines. In the first line print the sizes of subset in the following format: "`?` k_a k_b" where k_a and k_b ($1 \le k_a, k_b \le n$; $k_a + k_b \le n$) — the corresponding sizes.

In the second line print k_a integers $a_1, a_2, \ldots, a_{k_a}$ ($1 \le a_i \le n$; $a_i \ne a_j$ if $i \ne j$) — indexes of boxes in the first subset.

In the third line print k_b integers $b_1, b_2, \ldots, b_{k_b}$ ($1 \le b_i \le n$; $b_i \ne b_j$ if $i \ne j$) — indexes of boxes in the second subset.

The subsets shouldn't intersect, i. e. $a_i \ne b_j$ for all i and j.

You'll receive one of four responses described above. In the case of `WASTED` stop your program to avoid getting random verdict instead of `Wrong Answer`.

Example

input
2 2 1 - - - FIRST - 5 2 - - - FIRST - - - SECOND - - - EQUAL -
output
- - ? 1 1 1 2 - ! 2 - ? 1 1 1 2 - ? 2 3 4 2 1 3 5 - ? 1 1 4 5 - ! 1

Note

Additional separators "–" in the sample are used only to increase the readability of the sample. **Don't print any unnecessary symbols or line breaks in your solution when you send it to the system.**

Hacks are forbidden in this task.