



Problem Ramen

Input file `stdin`
Output file `stdout`

There are N friends F_0, \dots, F_{N-1} and N types of ramen R_0, \dots, R_{N-1} at the ramen restaurant. Each friend F_i has a certain affinity A_{ij} for ramen type j — the greater the affinity, the more friend F_i likes ramen R_j . The affinities of each friend are distinct — i.e., $A_{ij} \neq A_{ij'}$ for $j \neq j'$. Of course, it is possible that $A_{ij} < 0$.

Suppose the friends visit the ramen restaurant multiple times. During each visit, no two friends can eat the same type of ramen (the supply would be insufficient). Suppose that the friends take their ramen in order $F_{\pi_0}, \dots, F_{\pi_{N-1}}$, for some permutation π_0, \dots, π_{N-1} of $0, \dots, N-1$. Then, friend F_{π_0} will take their favorite ramen (i.e., the one for which they have the highest affinity), friend F_{π_1} will take their favorite ramen *except the one taken by* F_{π_0} , and so on. In other words, F_{π_i} will take their favourite ramen among those not taken by $F_{\pi_0}, \dots, F_{\pi_{i-1}}$.

The *goodness* of a certain permutation π is the sum of the affinities of the friends for the types of ramen they take. In other words, if friend i takes ramen of type σ_i , then the goodness of π is given by $\sum_{i=0}^{N-1} A_{i, \sigma(i)}$.

Your goal is to find a permutation π with maximum goodness. You can do this by experimenting with different orders in which the friends take their ramen (i.e., different visits to the restaurant). The goal is to find an optimal permutation without needing too many visits to the restaurant.

Interaction protocol

You have to implement the following function:

```
std::vector<int> find_order(int N);
```

Here, N is the number of friends. The function should return a permutation π of $0, \dots, N-1$ with maximum goodness. To implement it, you may repeatedly call the following function at most 750 times:

```
std::vector<std::pair<int, int>> query(const std::vector<int>& order);
```

The function takes as input a permutation π of $0, \dots, N-1$ (denoted by the parameter `order`) and returns a list of pairs $(\sigma(i), A_{i, \sigma(i)})$, where $\sigma(i)$ is the type of ramen that friend i takes if the friends take their ramen in the order given by π .

Restrictions

- $1 \leq N \leq 75$
- $|A_{ij}| \leq 2\,000\,000$
- The model solution of the scientific committee requires at most cN^k queries for some constants $c, k \geq 1$. To not overburden the testing infrastructure, you may call `query` at most 750 times, which generously reflects the practical performance of the model solution.



#	Points	Restrictions
1	5	$N = 5$
2	15	$N = 15$
3	20	$N = 30$
4	20	$N = 45$
5	20	$N = 60$
6	20	$N = 75$

Examples

Competitor behaviour	Committee behaviour
	Calls <code>find_order(2)</code> .
<code>query({0, 1})</code>	
	$\{\{0, 9\}, \{1, 0\}\}$
<code>query({1, 0})</code>	
	$\{\{1, 5\}, \{0, 5\}\}$
<code>find_order(2)</code> returns $\{1, 0\}$.	

In this example, there are $N = 2$ friends, with the following affinities $A_{i,j}$ for the $N = 2$ types of ramen:

A	0	1
0	9	5
1	5	0

The interaction begins with the committee calling your function: `find_order(2)`. Your function then queries the two possible permutations: $\{0, 1\}$ and $\{1, 0\}$. The former achieves a goodness of $0 + 9 = 9$, while the latter achieves a goodness of $5 + 5 = 10$. The function then returns the better of the two, which is $\{1, 0\}$. The returned permutation has the maximum possible goodness, as required.