A. Maximize the Last Element

```
Input file:     standard input
Output file:    standard output
Time limit:     1 second
Memory limit:   256 megabytes
```

You are given an array a of n integers, where n is **odd**.

In one operation, you will remove two **adjacent** elements from the array a, and then concatenate the remaining parts of the array. For example, given the array $[4, 7, 4, 2, 9]$, we can obtain the arrays $[4, 2, 9]$ and $[4, 7, 9]$ by the operations $[\underline{4, 7}, 4, 2, 9] \to [4, 2, 9]$ and $[4, 7, \underline{4, 2}, 9] \to [4, 7, 9]$ respectively. However, we cannot obtain the array $[7, 2, 9]$ as it requires deleting non-adjacent elements $[\underline{4}, 7, \underline{4}, 2, 9]$.

You will repeatedly perform this operation until exactly one element remains in a.

Find the maximum possible value of the remaining element in a.

Input

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

The first line of each test case contains a single integer n $(1 \le n \le 99$; n is odd) — the length of the array a.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ $(1 \le a_i \le 100)$ — the elements of the array a.

Note that there is no bound on the sum of n over all test cases.

Output

For each test case, output a single integer — the maximum possible value of the remaining element in a.

Standard Input	Standard Output

4
1
6
3
1 3 2
5
4 7 4 2 9
7
3 1 4 1 5 9 2
	6
	2
	9
	5

Note

In the first test case, the array a is $[6]$. Since there is only one element, no operations are needed. The maximum possible value of the remaining element is 6.

In the second test case, the array a is $[1, 3, 2]$. We can remove the first two elements $[\underline{1, 3}, 2] \to [2]$, or remove the last two elements $[1, \underline{3, 2}] \to [1]$. Therefore, the maximum possible value of the remaining element is 2.

In the third test case, the array a is $[4, 7, 4, 2, 9]$. One way to maximize the remaining element is $[4, \underline{7, 4}, 2, 9] \to [\underline{4, 2}, 9] \to [9]$. Therefore, the maximum possible value of the remaining element is 9.

In the fourth test case, the array a is $[3, 1, 4, 1, 5, 9, 2]$. It can be shown that the maximum possible value of the remaining element is 5.

,

B. AND Reconstruction

```
Input file:     standard input
Output file:    standard output
Time limit:     1 second
Memory limit:   256 megabytes
```

You are given an array b of $n - 1$ integers.

An array a of n integers is called *good* if $b_i = a_i \,\&\, a_{i+1}$ for $1 \le i \le n-1$, where $\&$ denotes the bitwise AND operator.

Construct a good array, or report that no good arrays exist.

Input

Each test contains multiple test cases. The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases. The description of test cases follows.

The first line of each test case contains a single integer n ($2 \le n \le 10^5$) — the length of the array a.

The second line of each test case contains $n - 1$ integers $b_1, b_2, \ldots, b_{n-1}$ ($0 \le b_i < 2^{30}$) — the elements of the array b.

It is guaranteed that the sum of n over all test cases does not exceed 10^5.

Output

For each test case, output a single integer -1 if no good arrays exist.

Otherwise, output n space-separated integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i < 2^{30}$) — the elements of a good array a.

If there are multiple solutions, you may output any of them.

Standard Input	Standard Output

4
2
1
3
2 0
4
1 2 3
5
3 5 4 2
	5 3
	3 2 1
	-1
	3 7 5 6 3

Note

In the first test case, $b = [1]$. A possible good array is $a = [5, 3]$, because $a_1 \,\&\, a_2 = 5 \,\&\, 3 = 1 = b_1$.

In the second test case, $b = [2, 0]$. A possible good array is $a = [3, 2, 1]$, because $a_1 \,\&\, a_2 = 3 \,\&\, 2 = 2 = b_1$ and $a_2 \,\&\, a_3 = 2 \,\&\, 1 = 0 = b_2$.

In the third test case, $b = [1, 2, 3]$. It can be shown that no good arrays exist, so the output is -1.

In the fourth test case, $b = [3, 5, 4, 2]$. A possible good array is $a = [3, 7, 5, 6, 3]$.

,

C. Absolute Zero

```
Input file:     standard input
Output file:    standard output
Time limit:     2 seconds
Memory limit:   256 megabytes
```

You are given an array a of n integers.

In one operation, you will perform the following two-step move:

1. Choose an integer x ($0 \le x \le 10^9$).
2. Replace each a_i with $|a_i - x|$, where $|v|$ denotes the absolute value of v.

For example, by choosing $x = 8$, the array $[5, 7, 10]$ will be changed into $[|5 - 8|, |7 - 8|, |10 - 8|] = [3, 1, 2]$.

Construct a sequence of operations to make all elements of a equal to 0 in at most 40 operations or determine that it is impossible. You do **not** need to minimize the number of operations.

Input

Each test contains multiple test cases. The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases. The description of test cases follows.

The first line of each test case contains a single integer n ($1 \le n \le 2 \cdot 10^5$) — the length of the array a.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 10^9$) — the elements of the array a.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output a single integer -1 if it is impossible to make all array elements equal to 0 in at most 40 operations.

Otherwise, output two lines. The first line of output should contain a single integer k ($0 \le k \le 40$) — the number of operations. The second line of output should contain k integers $x_1, x_2, \ldots, x_k$ ($0 \le x_i \le 10^9$) — the sequence of operations, denoting that on the i-th operation, you chose $x = x_i$.

If there are multiple solutions, output any of them.

You do **not** need to minimize the number of operations.

Standard Input	Standard Output

```
5
1
5
2
0 0
3
4 6 8
4
80 40 20 10
5
1 2 3 4 5
```

```
1
5
0

3
6 1 1
7
60 40 20 10 30 25 5
-1
```

Note

In the first test case, we can perform only one operation by choosing $x = 5$, changing the array from $[5]$ to $[0]$.

In the second test case, no operations are needed because all elements of the array are already 0.

In the third test case, we can choose $x = 6$ to change the array from $[4, 6, 8]$ to $[2, 0, 2]$, then choose $x = 1$ to change it to $[1, 1, 1]$, and finally choose $x = 1$ again to change the array into $[0, 0, 0]$.

In the fourth test case, we can make all elements 0 by following the operation sequence $(60, 40, 20, 10, 30, 25, 5)$.

In the fifth test case, it can be shown that it is impossible to make all elements 0 in at most 40 operations. Therefore, the output is -1.

,

D. Prime XOR Coloring

```
Input file:     standard input
Output file:    standard output
Time limit:     2 seconds
Memory limit:   256 megabytes
```

You are given an undirected graph with n vertices, numbered from 1 to n. There is an edge between vertices u and v if and only if $u \oplus v$ is a prime number, where $\oplus$ denotes the bitwise XOR operator.

Color all vertices of the graph using the minimum number of colors, such that no two vertices directly connected by an edge have the same color.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 500$). The description of test cases follows.

The only line contains a single integer n ($1 \le n \le 2 \cdot 10^5$) — the number of vertices in the graph.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output two lines.

The first line should contain a single integer k ($1 \le k \le n$) — the minimum number of colors required.

The second line should contain n integers $c_1, c_2, \ldots, c_n$ ($1 \le c_i \le k$) — the color of each vertex.

If there are multiple solutions, output any of them.

Standard Input	Standard Output

6
1
2
3
4
5
6
	1
	1
	2
	1 2
	2
	1 2 2
	3
	1 2 2 3
	3
	1 2 2 3 3
	4
	1 2 2 3 3 4

Note

In the first test case, the minimum number of colors is 1, because there is only one vertex.

In the second test case, the minimum number of colors is 2, because there is an edge connecting 1 and 2 ($1 \oplus 2 = 3$, which is a prime number).

In the third test case, the minimum number of colors is still 2, because 2 and 3 can be colored the same since there is no edge between 2 and 3 ($2 \oplus 3 = 1$, which is not a prime number).

In the fourth test case, it can be shown that the minimum number of colors is 3.

In the fifth test case, it can be shown that the minimum number of colors is 3.

In the sixth test case, it can be shown that the minimum number of colors is 4.

,

E. Coloring Game

```
Input file:     standard input
Output file:    standard output
Time limit:     2 seconds
Memory limit:   256 megabytes
```

This is an interactive problem.

Consider an undirected connected graph consisting of n vertices and m edges. Each vertex can be colored with one of three colors: 1, 2, or 3. Initially, all vertices are uncolored.

Alice and Bob are playing a game consisting of n rounds. In each round, the following two-step process happens:

1. Alice chooses two **different** colors.
2. Bob chooses an uncolored vertex and colors it with one of the two colors chosen by Alice.

Alice wins if there exists an edge connecting two vertices of the same color. Otherwise, Bob wins.

You are given the graph. Your task is to decide which player you wish to play as and win the game.

Input

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

The first line of each test case contains two integers n, m ($1 \le n \le 10^4$, $n - 1 \le m \le \min(\frac{n\cdot(n-1)}{2}, 10^4)$) — the number of vertices and the number of edges in the graph, respectively.

Each of the next m lines of each test case contains two integers u_i, v_i ($1 \le u_i, v_i \le n$) — the edges of the graph. It is guaranteed that the graph is connected and there are no multiple edges or self-loops.

It is guaranteed that the sum of n and the sum of m over all test cases does not exceed 10^4.

Interaction

For each test case, you need to output a single line containing either "`Alice`" or "`Bob`", representing the player you choose.

Then for each of the following n rounds, the following two-step process happens:

1. Alice (either you or the interactor) will output two integers a and b ($1 \le a, b \le 3$, $a \ne b$) — the colors chosen by Alice.
2. Bob (either you or the interactor) will output two integers i and c ($1 \le i \le n$, $c = a$ or $c = b$) — the vertex and the color chosen by Bob. Vertex i must be a previously uncolored vertex.

If any of your outputs are invalid, the jury will output "`-1`" and you will receive a **Wrong Answer** verdict.

At the end of all n turns, if you have lost the game, the jury will output "`-1`" and you will receive a **Wrong Answer** verdict.

If your program has received a -1 instead of a valid value, it must terminate immediately. Otherwise, you may receive an arbitrary verdict because your solution might be reading from a closed stream.

Note that if you are playing as Alice, and there already exists an edge connected two vertices of the same color, the interactor will not terminate early and you will keep playing all n rounds.

After outputting, do not forget to output end of line and flush the output. Otherwise, you will get `Idleness limit exceeded`. To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `System.out.flush()` in Java;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;
- see documentation for other languages.

In this problem, hacks are disabled.

Standard Input	Standard Output
2	
3 3	
1 2	
2 3	
3 1	Alice
	3 1
3 1	1 2
2 2	2 1
1 1	
4 4	
1 2	
2 3	
3 4	
4 1	Bob
2 3	1 2
1 2	2 1
2 1	4 1
3 1	3 3

Note

Note that the sample test cases are example games and do not necessarily represent the optimal strategy for both players.

In the first test case, you choose to play as Alice.

1. Alice chooses two colors: 3 and 1. Bob chooses vertex 3 and colors it with color 1.
2. Alice chooses two colors: 1 and 2. Bob chooses vertex 2 and colors it with color 2.

3. Alice chooses two colors: 2 and 1. Bob chooses vertex 1 and colors it with color 1.

Alice wins because the edge $(3, 1)$ connects two vertices of the same color.

In the second test case, you choose to play as Bob.

1. Alice chooses two colors: 2 and 3. Bob chooses vertex 1 and colors it with color 2.
2. Alice chooses two colors: 1 and 2. Bob chooses vertex 2 and colors it with color 1.
3. Alice chooses two colors: 2 and 1. Bob chooses vertex 4 and colors it with color 1.
4. Alice chooses two colors: 3 and 1. Bob chooses vertex 3 and colors it with color 3.

Bob wins because there are no edges with vertices of the same color.

,

F. Triangle Formation

```
Input file:     standard input
Output file:    standard output
Time limit:     5 seconds
Memory limit:   256 megabytes
```

You are given n sticks, numbered from 1 to n. The length of the i-th stick is a_i.

You need to answer q queries. In each query, you are given two integers l and r ($1 \le l < r \le n$, $r - l + 1 \ge 6$). Determine whether it is possible to choose 6 **distinct** sticks from the sticks numbered l to r, to form 2 **non-degenerate** triangles*.

*A triangle with side lengths a, b, and c is called non-degenerate if:

- $a < b + c$,
- $b < a + c$, and
- $c < a + b$.

Input

The first line contains two integers n and q ($6 \le n \le 10^5$, $1 \le q \le 10^5$) — the number of sticks and the number of queries respectively.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — a_i denotes the length of the i-th stick.

Each of the following q lines contains two integers l and r ($1 \le l < r \le n$, $r - l + 1 \ge 6$) — the parameters of each query.

Output

For each query, output "YES" (without quotes) if it is possible to form 2 triangles, and "NO" (without quotes) otherwise.

You can output the answer in any case (upper or lower). For example, the strings "yEs", "yes", "Yes", and "YES" will be recognized as positive responses.

Standard Input	Standard Output

10 5
5 2 2 10 4 10 6 1 5 3
1 6
2 7
2 8
5 10
4 10
	YES
	NO
	YES
	NO
	YES

Note

In the first query, the lengths of the sticks are $[5, 2, 2, 10, 4, 10]$. Two sets of sticks $[2, 4, 5]$ and $[2, 10, 10]$ can be selected to form 2 non-degenerate triangles.

In the second query, the lengths of the sticks are $[2, 2, 10, 4, 10, 6]$. It can be shown that it is impossible to form 2 non-degenerate triangles.

In the third query, the lengths of the sticks are $[2, 2, 10, 4, 10, 6, 1]$. Two sets of sticks $[1, 2, 2]$ and $[4, 10, 10]$ can be selected to form 2 non-degenerate triangles.

In the fourth query, the lengths of the sticks are $[4, 10, 6, 1, 5, 3]$. It can be shown that it is impossible to form 2 non-degenerate triangles.

In the fifth query, the lengths of the sticks are $[10, 4, 10, 6, 1, 5, 3]$. Two sets of sticks $[1, 10, 10]$ and $[3, 4, 5]$ can be selected to form 2 non-degenerate triangles.

,

G. Grid Reset

```
Input file:     standard input
Output file:    standard output
Time limit:     2 seconds
Memory limit:   256 megabytes
```

You are given a grid consisting of n rows and m columns, where each cell is initially white. Additionally, you are given an integer k, where $1 \le k \le \min(n, m)$.

You will process q operations of two types:

- `H` (horizontal operation) — You choose a $1 \times k$ rectangle completely within the grid, where all cells in this rectangle are white. Then, all cells in this rectangle are changed to black.
- `V` (vertical operation) — You choose a $k \times 1$ rectangle completely within the grid, where all cells in this rectangle are white. Then, all cells in this rectangle are changed to black.

After each operation, if any rows or columns become completely black, all cells in these rows and columns are **simultaneously** reset to white. Specifically, if all cells in the row and column a cell is contained in become black, all cells in both the row and column will be reset to white.

Choose the rectangles in a way that you can perform all given operations, or determine that it is impossible.

Input

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

The first line of each test case contains four integers n, m, k, and q ($1 \le n, m \le 100$, $1 \le k \le \min(n, m)$, $1 \le q \le 1000$) — the number of rows and columns in the grid, the size of the operation rectangle, and the number of operations, respectively.

The second line of each test case contains a string s of length q, consisting only of characters `H` and `V` — the sequence of operation types.

It is guaranteed that the sum of q over all test cases does not exceed 1000.

Output

For each test case, output a single integer -1 if it is impossible to perform all the operations.

Otherwise, output q lines. Each line contains two integers i, j ($1 \le i \le n$, $1 \le j \le m$) — the coordinates of the top-left cell of the operation rectangle.

If there are multiple solutions, output any of them.

Standard Input	Standard Output

1
4 5 3 6
HVVHHV
	1 1
	2 1
	1 1
	2 3
	3 3
	2 2

Note

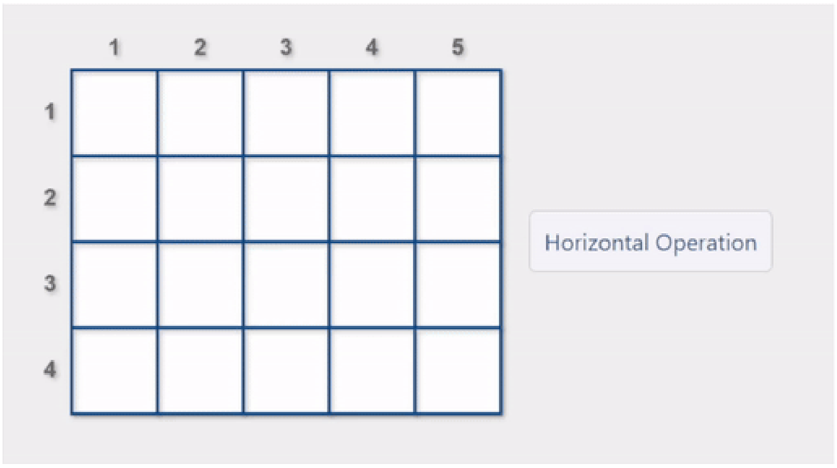

Illustration of example.

The first operation is horizontal. The operation rectangle starts at $(1, 1)$ and is a 1×3 rectangle. After the operation, cells $(1, 1)$, $(1, 2)$, and $(1, 3)$ become black.

The second operation is vertical. The operation rectangle starts at $(2, 1)$ and is a 3×1 rectangle. After the operation, cells $(2, 1)$, $(3, 1)$, and $(4, 1)$ become black. At this point, the first column becomes completely black, so all cells in the first column are reset to white.

The third operation is vertical. The operation rectangle starts at $(1, 1)$ and is a 3×1 rectangle. After the operation, cells $(1, 1)$, $(2, 1)$, and $(3, 1)$ become black.

The fourth operation is horizontal. The operation rectangle starts at $(2, 3)$ and is a 1×3 rectangle. After the operation, cells $(2, 3)$, $(2, 4)$, and $(2, 5)$ become black.

The fifth operation is horizontal. The operation rectangle starts at $(3, 3)$ and is a 1×3 rectangle. After the operation, cells $(3, 3)$, $(3, 4)$, and $(3, 5)$ become black.

The sixth operation is vertical. The operation rectangle starts at $(2, 2)$ and is a 3×1 rectangle. After the operation, cells $(2, 2)$, $(3, 2)$, and $(4, 2)$ become black. At this point, two rows and one column become completely black, so all cells in these rows and the column are reset to white.

,

H. Prime Split Game

```
Input file:     standard input
Output file:    standard output
Time limit:     2 seconds
Memory limit:   256 megabytes
```

Alice and Bob are playing a game with n piles of stones, where the i-th pile has a_i stones. Players take turns making moves, with Alice going first.

On each move, the player does the following three-step process:

1. Choose an integer k ($1 \le k \le \frac{n}{2}$). Note that the value of k can be different for different moves.
2. Remove k piles of stones.
3. Choose another k piles of stones and split each pile into two piles. The number of stones in each new pile must be a prime number.

The player who is unable to make a move loses.

Determine who will win if both players play optimally.

Input

Each test contains multiple test cases. The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases. The description of test cases follows.

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

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 2 \cdot 10^5$) — the number of stones in the piles.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output "`Alice`" (without quotes) if Alice wins and "`Bob`" (without quotes) otherwise.

You can output each letter in any case (upper or lower). For example, the strings "`alIcE`", "`Alice`", and "`alice`" will all be considered identical.

Standard Input	Standard Output

4
2
2 1
3
3 5 7
4
4 6 8 10
5
8 8 8 8 8
	Bob
	Alice
	Alice
	Bob

Note

In the first test case, there are 2 piles of stones with 2 and 1 stones respectively. Since neither 1 nor 2 can be split into two prime numbers, Alice cannot make a move, so Bob wins.

In the second test case, there are 3 piles of stones with 3, 5, and 7 stones respectively. Alice can choose $k = 1$, remove the pile of 7 stones, and then split the pile of 5 stones into two piles of prime numbers of stones, 2 and 3. Then, the piles consist of 3 piles of stones with 3, 2, and 3 stones respectively, leaving Bob with no valid moves, so Alice wins.

In the third test case, there are 4 piles of stones with 4, 6, 8, and 10 stones respectively. Alice can choose $k = 2$, removing two piles of 8 and 10 stones. She splits the pile of 4 stones into two piles of prime numbers of stones, 2 and 2, and the pile of 6 stones into two piles of 3 and 3 stones. Then, Bob has no valid moves, so Alice wins.

In the fourth test case, there are 5 piles of stones, each containing 8 stones. It can be shown that if both players play optimally, Bob will win.

,

I. Grid Game

```
Input file:     standard input
Output file:    standard output
Time limit:     5 seconds
Memory limit:   256 megabytes
```

This is an interactive problem.

You are given a grid with n rows and m columns. You need to fill each cell with a unique integer from 1 to $n \cdot m$.

After filling the grid, you will play a game on this grid against the interactor. Players take turns selecting one of the previously unselected cells from the grid, with the interactor going first.

On the first turn, the interactor can choose any cell from the grid. After that, any chosen cell must be orthogonally adjacent to at least one previously selected cell. Two cells are considered orthogonally adjacent if they share an edge. The game continues until all cells have been selected.

Your goal is to let the sum of numbers in the cells selected by you be **strictly** less than the sum of numbers in the cells selected by the interactor.

Input

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

The only line of each test case contains two integers n and m ($4 \le n, m \le 10$) — the number of rows and columns in the grid.

Interaction

First, output n lines, each containing m integers, representing the integers that you filled in the grid. Each integer from 1 to $n \cdot m$ should appear exactly once.

Then, the game begins. The interactor and you take turns outputting coordinates to select a cell for $n \times m$ turns, with the interactor starting first.

On each player's (either you or the interactor) turn, the player will output two integers i and j ($1 \le i \le n$, $1 \le j \le m$), denoting that the player has selected the cell on the i-th row and j-th column. This cell should not have been selected in a previous round. Additionally, if it is not the first turn, the cell must be orthogonally adjacent to at least one previously selected cell.

If any of your outputs are invalid, the jury will output "`-1`" and you will receive a **Wrong Answer** verdict.

At the end of all $n \cdot m$ turns, if the sum of numbers in the cells selected by you is not strictly less than the sum of numbers in the cells selected by the interactor, the jury will output "`-1`" and you will receive a **Wrong Answer** verdict.

If your program has received a **Wrong Answer** verdict, it must terminate immediately. Otherwise, you may receive an arbitrary verdict because your solution might be reading from a closed stream.

After outputting, do not forget to output end of line and flush the output. Otherwise, you will get `Idleness limit exceeded`. To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;

- `System.out.flush()` in Java;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;
- see documentation for other languages.

In this problem, hacks are disabled.

Standard Input	Standard Output
1	
4 4	2 3 4 10
	12 6 11 15
	5 13 16 8
	9 7 1 14
3 4	2 4
4 4	4 3
4 2	3 3
4 1	3 1
1 4	1 3
1 2	1 1
2 2	2 3
2 1	3 2

Note

Note that this is an example game and does not necessarily represent the optimal strategy for both players.

First, we fill a 4×4 grid with unique integers from 1 to 16 in the following way:

2	3	4	10
12	6	11	15
5	13	16	8
9	7	1	14

Next, the game begins.

1. The interactor first selects $(3, 4)$, which is the number 8. For this selection, the interactor could choose any number. From the next selection onwards, each chosen number has to be adjacent to any previously selected number.
2. We select $(2, 4)$, which is the number 15, adjacent to $(3, 4)$.
3. The interactor selects $(4, 4)$, which is the number 14, adjacent to $(3, 4)$.

4. We select $(4, 3)$, which is the number 1, adjacent to $(4, 4)$.
5. ...
6. This is continued until all numbers are selected.

In the end, the numbers we selected were $[15, 1, 16, 5, 4, 2, 11, 13]$, and the numbers selected by the interactor were $[8, 14, 7, 9, 10, 3, 6, 12]$. The sum of the numbers we selected is 67, which is less than the sum of the numbers selected by the interactor 69. Therefore, we have won this game.