Souvenirs

Amaru is buying souvenirs in a foreign shop. There are N **types** of souvenirs. There are infinitely many souvenirs of each type available in the shop.

Each type of souvenir has a fixed price. Namely, a souvenir of type i ($0 \leq i < N$) has a price of $P[i]$ coins, where $P[i]$ is a positive integer.

Amaru knows that souvenir types are sorted in decreasing order by price, and that the souvenir prices are distinct. Specifically, $P[0] > P[1] > \cdots > P[N-1] > 0$. Moreover, he was able to learn the value of $P[0]$. Unfortunately, Amaru does not have any other information about the prices of the souvenirs.

To buy some souvenirs, Amaru will perform a number of transactions with the seller.

Each transaction consists of the following steps:

1. Amaru hands some (positive) number of coins to the seller.
2. The seller puts these coins in a pile on the table in the back room, where Amaru cannot see them.
3. The seller considers each souvenir type $0, 1, \ldots, N-1$ in that order, one by one. Each type is considered **exactly once** per transaction.
    - When considering souvenir type i, if the current number of coins in the pile is at least $P[i]$, then
        - the seller removes $P[i]$ coins from the pile, and
        - the seller puts one souvenir of type i on the table.
4. The seller gives Amaru all the coins remaining in the pile and all souvenirs on the table.

Note that there are no coins or souvenirs on the table before a transaction begins.

Your task is to instruct Amaru to perform some number of transactions, so that:

- in each transaction he buys **at least one** souvenir, and
- overall he buys **exactly** i souvenirs of type i, for each i such that $0 \leq i < N$. Note that this means that Amaru should not buy any souvenir of type 0.

Amaru does not have to minimize the number of transactions and has an unlimited supply of coins.

Implementation Details

You should implement the following procedure.

```
void buy_souvenirs(int N, long long P0)
```

- N: the number of souvenir types.
- $P0$: the value of $P[0]$.
- This procedure is called exactly once for each test case.

The above procedure can make calls to the following procedure to instruct Amaru to perform a transaction:

```
std::pair<std::vector<int>, long long> transaction(long long M)
```

- M: the number of coins handed to the seller by Amaru.
- The procedure returns a pair. The first element of the pair is an array L, containing the types of souvenirs that have been bought (in increasing order). The second element is an integer R, the number of coins returned to Amaru after the transaction.
- It is required that $P[0] > M \geq P[N-1]$. The condition $P[0] > M$ ensures that Amaru does not buy any souvenir of type 0, and $M \geq P[N-1]$ ensures that Amaru buys at least one souvenir. If these conditions are not met, your solution will receive the verdict `Output isn't correct: Invalid argument`. Note that contrary to $P[0]$, the value of $P[N-1]$ is not provided in the input.
- The procedure can be called at most 5000 times in each test case.

The behavior of the grader is **not adaptive**. This means that the sequence of prices P is fixed before `buy_souvenirs` is called.

Constraints

- $2 \leq N \leq 100$
- $1 \leq P[i] \leq 10^{15}$ for each i such that $0 \leq i < N$.
- $P[i] > P[i+1]$ for each i such that $0 \leq i < N-1$.

Subtasks

Subtask	Score	Additional Constraints
1	4	$N = 2$
2	3	$P[i] = N - i$ for each i such that $0 \leq i < N$.
3	14	$P[i] \leq P[i + 1] + 2$ for each i such that $0 \leq i < N - 1$.
4	18	$N = 3$
5	28	$P[i + 1] + P[i + 2] \leq P[i]$ for each i such that $0 \leq i < N - 2$. $P[i] \leq 2 \cdot P[i + 1]$ for each i such that $0 \leq i < N - 1$.
6	33	No additional constraints.

Example

Consider the following call.

```
buy_souvenirs(3, 4)
```

There are $N = 3$ types of souvenirs and $P[0] = 4$. Observe that there are only three possible sequences of prices P: $[4, 3, 2]$, $[4, 3, 1]$, and $[4, 2, 1]$.

Assume that `buy_souvenirs` calls `transaction(2)`. Suppose the call returns $([2], 1)$, meaning that Amaru bought one souvenir of type 2 and the seller gave him back 1 coin. Observe that this allows us to deduce that $P = [4, 3, 1]$, since:

- For $P = [4, 3, 2]$, `transaction(2)` would have returned $([2], 0)$.
- For $P = [4, 2, 1]$, `transaction(2)` would have returned $([1], 0)$.

Then `buy_souvenirs` can call `transaction(3)`, which returns $([1], 0)$, meaning that Amaru bought one souvenir of type 1 and the seller gave him back 0 coins. So far, in total, he has bought one souvenir of type 1 and one souvenir of type 2.

Finally, `buy_souvenirs` can call `transaction(1)`, which returns $([2], 0)$, meaning that Amaru bought one souvenir of type 2. Note that we could have also used `transaction(2)` here. At this point, in total Amaru has one souvenir of type 1 and two souvenirs of type 2, as required.

Sample Grader

Input format:

```
N
P[0] P[1] ... P[N-1]
```

Output format:

```
Q[0] Q[1] ... Q[N-1]
```

Here $Q[i]$ is the number of souvenirs of type i bought in total for each i such that $0 \leq i < N$.

Triple Peaks

The Cordillera Oriental is a mountain range in the Andes that stretches across Bolivia. It consists of a sequence of N mountain peaks, numbered from 0 to $N-1$. The **height** of peak i ($0 \leq i < N$) is $H[i]$, which is an integer between 1 and $N-1$, inclusive.

For any two peaks i and j where $0 \leq i < j < N$, the **distance** between them is defined as $d(i, j) = j - i$.

According to ancient Inca legends, a triple of peaks is **mythical** if it has the following special property: the heights of the three peaks **match** their pairwise distances **ignoring the order**.

Formally, a triple of indices (i, j, k) is mythical if

- $0 \leq i < j < k < N$, and
- the heights $(H[i], H[j], H[k])$ match the pairwise distances $(d(i, j), d(i, k), d(j, k))$ ignoring the order. For example, for indices $0, 1, 2$ the pairwise distances are $(1, 2, 1)$, so the heights $(H[0], H[1], H[2]) = (1, 1, 2)$, $(H[0], H[1], H[2]) = (1, 2, 1)$, and $(H[0], H[1], H[2]) = (2, 1, 1)$ all match them, but the heights $(H[0], H[1], H[2]) = (1, 2, 2)$ do not match them.

This problem consists of two parts, with each subtask associated with either **Part I** or **Part II**. You may solve the subtasks in any order. In particular, you are **not** required to complete all of Part I before attempting Part II.

Part I

Given a description of the mountain range, your task is to count the number of mythical triples.

Implementation Details

You should implement the following procedure.

```
long long count_triples(std::vector<int> H)
```

- H: array of length N, representing the heights of the peaks.
- This procedure is called exactly once for each test case.

The procedure should return an integer T, the number of mythical triples in the mountain range.

Constraints

- $3 \leq N \leq 200\,000$
- $1 \leq H[i] \leq N - 1$ for each i such that $0 \leq i < N$.

Subtasks

Part I is worth a total of 70 points.

Subtask	Score	Additional Constraints
1	8	$N \leq 100$
2	6	$H[i] \leq 10$ for each i such that $0 \leq i < N$.
3	10	$N \leq 2000$
4	11	The heights are non-decreasing. That is, $H[i-1] \leq H[i]$ for each i such that $1 \leq i < N$.
5	16	$N \leq 50\,000$
6	19	No additional constraints.

Example

Consider the following call.

```
count_triples([4, 1, 4, 3, 2, 6, 1])
```

There are 3 mythical triples in the mountain range:

- For $(i, j, k) = (1, 3, 4)$, the heights $(1, 3, 2)$ match the pairwise distances $(2, 3, 1)$.
- For $(i, j, k) = (2, 3, 6)$, the heights $(4, 3, 1)$ match the pairwise distances $(1, 4, 3)$.
- For $(i, j, k) = (3, 4, 6)$, the heights $(3, 2, 1)$ match the pairwise distances $(1, 3, 2)$.

Hence, the procedure should return 3.

Note that the indices $(0, 2, 4)$ do not form a mythical triple, as the heights $(4, 4, 2)$ do not match the pairwise distances $(2, 4, 2)$.

Part II

Your task is to construct mountain ranges with many mythical triples. This part consists of 6 **output-only** subtasks with **partial scoring**.

In each subtask, you are given two positive integers M and K, and you should construct a mountain range with **at most** M peaks. If your solution contains **at least** K mythical triples, you

will receive the full score for this subtask. Otherwise, your score will be proportional to the number of mythical triples your solution contains.

Note that your solution must consist of a valid mountain range. Specifically, suppose your solution has N peaks (N must satisfy $3 \leq N \leq M$). Then, the height of peak i ($0 \leq i < N$), denoted by $H[i]$, must be an integer between 1 and $N-1$, inclusive.

Implementation Details

There are two methods to submit your solution, and you may use either one for each subtask:

- **Output file**
- **Procedure call**

To submit your solution via an **output file**, create and submit a text file in the following format:

```
N
H[0] H[1] ... H[N-1]
```

To submit your solution via a **procedure call**, you should implement the following procedure.

```
std::vector<int> construct_range(int M, int K)
```

- M: the maximum number of peaks.
- K: the desired number of mythical triples.
- This procedure is called exactly once for each subtask.

The procedure should return an array H of length N, representing the heights of the peaks.

Subtasks and Scoring

Part II is worth a total of 30 points. For each subtask, the values of M and K are fixed and given in the following table:

Subtask	Score	M	K
7	5	20	30
8	5	500	2000
9	5	5000	$50\,000$
10	5	$30\,000$	$700\,000$
11	5	$100\,000$	$2\,000\,000$
12	5	$200\,000$	$12\,000\,000$

For each subtask, if your solution does not form a valid mountain range, your score will be 0 (reported as `Output isn't correct` in CMS).

Otherwise, let T denote the number of mythical triples in your solution. Then, your score for the subtask is:

$$5 \cdot \min\left(1, \frac{T}{K}\right).$$

Sample Grader

Parts I and II use the same sample grader program, with the distinction between the two parts determined by the first line of the input.

Input format for Part I:

```
1
N
H[0] H[1] ... H[N-1]
```

Output format for Part I:

```
T
```

Input format for Part II:

```
2
M K
```

Output format for Part II:

```
N
H[0] H[1] ... H[N-1]
```

Note that the output of the sample grader matches the required format for the output file in Part II.

World Map

Mr. Pacha, a Bolivian archeologist, discovered an ancient document near Tiwanaku that describes the world during the Tiwanaku Period (300-1000 CE). At that time, there were N countries, numbered from 1 to N.

In the document, there is a list of M different pairs of adjacent countries:

$$(A[0], B[0]), (A[1], B[1]), \ldots, (A[M-1], B[M-1]).$$

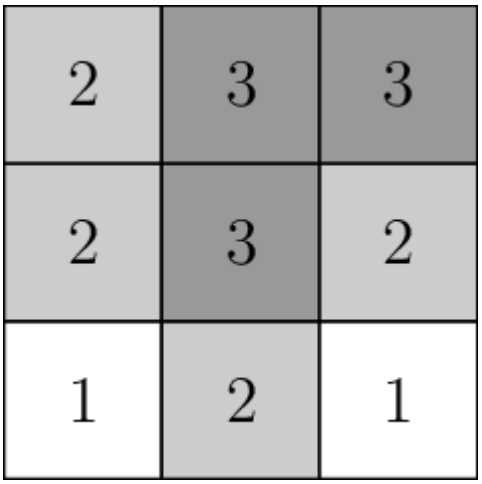
For each i ($0 \leq i < M$), the document states that country $A[i]$ was adjacent to country $B[i]$ and vice versa. Pairs of countries not listed were not adjacent.

Mr. Pacha wants to create a map of the world such that all adjacencies between countries are exactly as they were during the Tiwanaku Period. For this purpose, he first chooses a positive integer K. Then, he draws the map as a grid of $K \times K$ square cells, with rows numbered from 0 to $K-1$ (top to bottom) and columns numbered from 0 to $K-1$ (left to right).

He wants to color each cell of the map using one of N colors. The colors are numbered from 1 to N, and country j ($1 \leq j \leq N$) is represented by color j. The coloring must satisfy all of the following **conditions**:

- For each j ($1 \leq j \leq N$), there is at least one cell with color j.
- For each pair of adjacent countries $(A[i], B[i])$, there is at least one pair of adjacent cells such that one of them is colored $A[i]$ and the other is colored $B[i]$. Two cells are adjacent if they share a side.
- For each pair of adjacent cells with different colors, the countries represented by these two colors were adjacent during the Tiwanaku Period.

For example, if $N = 3$, $M = 2$ and the pairs of adjacent countries are $(1, 2)$ and $(2, 3)$, then the pair $(1, 3)$ was not adjacent, and the following map of dimension $K = 3$ satisfies all the conditions.



In particular, a country **does not** need to form a connected region on the map. In the map above, country 3 forms a connected region, while countries 1 and 2 form disconnected regions.

Your task is to help Mr. Pacha choose a value of K and create a map. The document guarantees that such a map exists. Since Mr. Pacha prefers smaller maps, in the last subtask your score depends on the value of K, and lower values of K may result in a better score. However, finding the minimum possible value of K is not required.

Implementation Details

You should implement the following procedure:

```
std::vector<std::vector<int>> create_map(int N, int M,
    std::vector<int> A, std::vector<int> B)
```

- N: the number of countries.
- M: the number of pairs of adjacent countries.
- A and B: arrays of length M describing adjacent countries.
- This procedure is called **up to** 50 **times** for each test case.

The procedure should return an array C that represents the map. Let K be the length of C.

- Each element of C must be an array of length K, containing integers between 1 and N inclusive.
- $C[i][j]$ is the color of the cell at row i and column j (for each i and j such that $0 \leq i, j < K$).
- K must be less than or equal to 240.

Constraints

- $1 \leq N \leq 40$
- $0 \leq M \leq \frac{N \cdot (N-1)}{2}$
- $1 \leq A[i] < B[i] \leq N$ for each i such that $0 \leq i < M$.

- The pairs $(A[0], B[0]), \ldots, (A[M-1], B[M-1])$ are distinct.
- There exists at least one map which satisfies all the conditions.

Subtasks and Scoring

Subtask	Score	Additional Constraints
1	5	$M = N - 1, A[i] = i + 1, B[i] = i + 2$ for each $0 \leq i < M$.
2	10	$M = N - 1$
3	7	$M = \frac{N \cdot (N-1)}{2}$
4	8	Country 1 is adjacent to all other countries. Some other pairs of countries may also be adjacent.
5	14	$N \leq 15$
6	56	No additional constraints.

In subtask 6, your score depends on the value of K.

- If any map returned by `create_map` does not satisfy all the conditions, your score for the subtask will be 0.
- Otherwise, let R be the **maximum** value of K / N over all calls to `create_map`. Then, you receive a **partial score** according to the following table:

Limits	Score
$6 < R$	0
$4 < R \leq 6$	14
$3 < R \leq 4$	28
$2.5 < R \leq 3$	42
$2 < R \leq 2.5$	49
$R \leq 2$	56

Example

In CMS, both of the following scenarios are included as part of a single test case.

Example 1

Consider the following call:

```
create_map(3, 2, [1, 2], [2, 3])
```

This is the example from the task description, so the procedure can return the following map.

```
[
[2, 3, 3],
[2, 3, 2],
[1, 2, 1]
]
```

Example 2

Consider the following call:

```
create_map(4, 4, [1, 1, 2, 3], [2, 3, 4, 4])
```

In this example, $N = 4$, $M = 4$ and the country pairs $(1, 2)$, $(1, 3)$, $(2, 4)$, and $(3, 4)$ are adjacent. Consequently, the pairs $(1, 4)$ and $(2, 3)$ are not adjacent.

The procedure can return the following map of dimension $K = 7$, which satisfies all the conditions.

```
[
[2, 1, 3, 3, 4, 3, 4],
[2, 1, 3, 3, 3, 3, 3],
[2, 1, 1, 1, 3, 4, 4],
[2, 2, 2, 1, 3, 4, 3],
[1, 1, 1, 2, 4, 4, 4],
[2, 2, 1, 2, 2, 4, 3],
[2, 2, 1, 2, 2, 4, 4]
]
```

The map could be smaller; for example, the procedure can return the following map of dimension $K = 2$.

```
[
[3, 1],
[4, 2]
]
```

Note that both maps satisfy $K / N \leq 2$.

Sample Grader

The first line of the input should contain a single integer T, the number of scenarios. A description of T scenarios should follow, each in the format specified below.

Input Format:

```
N M
A[0] B[0]
:
A[M-1] B[M-1]
```

Output Format:

```
P
Q[0] Q[1] ... Q[P-1]

C[0][0] ... C[0][Q[0]-1]
:
C[P-1][0] ... C[P-1][Q[P-1]-1]
```

Here, P is the length of the array C returned by `create_map`, and $Q[i]$ ($0 \leq i < P$) is the length of $C[i]$. Note that line 3 in the output format is intentionally left blank.