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 ($0 \le i < N$) 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 $0 \le i < N$. 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 $0 \leq i < N$.