Hack! (hack)

It has been an hour into a Codeforces contest, when you notice that another contestant in your room has solved a problem using an `unordered_set`. Time to hack!

You know that `unordered_set` uses a hash table with n buckets, which are numbered from 0 to $n-1$. Unfortunately, you do not know the value of n and wish to recover it.

When you insert an integer x into the hash table, it is inserted to the $(x \bmod n)$ -th bucket. If there are b elements in this bucket prior to the insertion, this will cause b hash collisions to occur.

By giving k distinct integers $x[0], x[1], \ldots, x[k-1]$ to the interactor, you can find out the total number of hash collisions that had occurred while creating an `unordered_set` containing the numbers. However, feeding this interactor k integers in one query will incur a cost of k.

For example, if $n = 5$, feeding the interactor with $x = [2, 15, 7, 27, 8, 30]$ would cause 4 collisions in total:

Operation	New collisions	Buckets
initially	$-$	$[], [], [], [], []$
insert $x[0] = 2$	0	$[], [], [2], [], []$
insert $x[1] = 15$	0	$[15], [], [2], [], []$
insert $x[2] = 7$	1	$[15], [], [2, 7], [], []$
insert $x[3] = 27$	2	$[15], [], [2, 7, 27], [], []$
insert $x[4] = 8$	0	$[15], [], [2, 7, 27], [8], []$
insert $x[5] = 30$	1	$[15, 30], [], [2, 7, 27], [8], []$

Note that the interactor creates the hash table by inserting the elements in order into an initially empty `unordered_set`, and a new empty `unordered_set` will be created for each query. In other words, all queries are independent.

Your task is to find the number of buckets n using total cost of at most $1\ 000\ 000$.

Implementation details

You should implement the following procedure:

```
int hack()
```

- This procedure should return an integer – the hidden value of n.
- For each test case, the grader may call this function more than once. Each call should be processed as a separately new scenario.

Within this procedure, you may call the following procedure:

```
long long collisions(std::vector<long long> x)
```

- x: an array of distinct numbers, where $1 \leq x[i] \leq 10^{18}$ for each i.
- This function returns the number of collisions created by inserting the elements of x to an `unordered_set`.
- This procedure can be called multiple times. The sum of length of x over all calls within one call to `hack()` must not exceed $1\ 000\ 000$.

Note: **Since the procedure `hack()` will be called more than once, contestants need to pay attention to the impact of the remaining data of the previous call on the current call, especially the state stored in global variables.**

The cost limit of $1\ 000\ 000$ applies to each test case. In general, if there are t calls to `hack()`, you may use a total cost of no more than $t \times 1\ 000\ 000$, with each individual call to `hack()` using a cost no more than $1\ 000\ 000$.

The interactor is not adaptive, i.e. the values of n are fixed before the start of interaction.

Example

Suppose, there are 2 multitests. The grader will make a following call:

```
hack()
```

Let's say, within the function, you make following calls:

Call	Returned value
`collisions([2, 15, 7, 27, 8, 30])`	4
`collisions([1, 2, 3])`	0
`collisions([10, 20, 30, 40, 50])`	10

After that, if you find that the value of n is 5, the procedure `hack()` should return 5.

Then grader will make another call:

```
hack()
```

Let's say, within the function, you make following calls:

Call	Returned value
`collisions([1, 3])`	1
`collisions([2, 4])`	1

The only n which satisfies the queries is 2. So, the procedure `hack()` should return 2.

Constraints

- $1 \leq t \leq 10$, where t is the number of multitests.
- $2 \leq n \leq 10^9$
- $1 \leq x[i] \leq 10^{18}$ for each call to `collisions()`.

Subtasks

1. (8 points) $n \leq 500\ 000$
2. (17 points) $n \leq 1\ 000\ 000$
3. (75 points) No additional constraints.

In the last subtask, you can get partial score. Let q be the maximum total cost among all invocations of `hack()` over every test case of the subtask. Your score for this subtask is calculated according to the following table:

Condition	Points
$1\ 000\ 000 < q$	0
$110\ 000 < q \leq 1\ 000\ 000$	$75 \cdot \log_{50} \left(\frac{10^6}{x-90000}\right)$
$q \leq 110\ 000$	75

If, in any of the test cases, the calls to the procedure `collisions()` do not conform to the constraints described in Implementation Details, or the number returned by `hack()` is incorrect, the score of your solution for that subtask will be 0.

Sample Grader

The sample grader reads the input in the following format:

- line 1: t

Then, t lines follow, each containing a value of n:

- line 1: n

For each test case, let m be the return value of `hack()`, and c be the total cost of all queries. The sample grader prints your answer in the following format:

- line 1: m c