ハック (hack)

Codeforces のコンテストに参加して 1 時間が経ったとき，あなたは同じルームの別の参加者が `unordered_set` を使って問題を解いていることに気づいた． ハックの時間だ！

`unordered_set` は 0 から $n-1$ まで番号が付けられている n 個のバケットを持つハッシュテーブルである． 残念なことに， あなたは n の値が分からない． そこで，それを突き止めようと考えた．

このハッシュテーブルに整数 x を挿入するとき，x は $(x \bmod n)$ 番目のバケットに挿入される． もしこのバケットにすでに b 個の要素があるとき，この挿入によりハッシュの衝突が b 回発生する．

k 個の相異なる整数 $x[0], x[1], \ldots, x[k-1]$ をインタラクターに与えることで，あなたは，配列 x の要素を使って `unordered_set` を作ったときに発生するハッシュの衝突回数の合計を知ることができる． ただし，k 個の整数を一度にインタラクタに与えると，そのクエリにはコストが k かかる．

例えば，$n = 5$ で$x = [2, 15, 7, 27, 8, 30]$をインタラクターに与えた時，ハッシュの衝突回数は合計で 4 回である：

操作	衝突回数	バケット
初期状態	$-$	$[], [], [], [], []$
$x[0] = 2$ を挿入	0	$[], [], [2], [], []$
$x[1] = 15$ を挿入	0	$[15], [], [2], [], []$
$x[2] = 7$ を挿入	1	$[15], [], [2, 7], [], []$
$x[3] = 27$ を挿入	2	$[15], [], [2, 7, 27], [], []$
$x[4] = 8$ を挿入	0	$[15], [], [2, 7, 27], [8], []$
$x[5] = 30$ を挿入	1	$[15, 30], [], [2, 7, 27], [8], []$

インタラクタは，空の `unordered_set` に要素を順に挿入することでハッシュテーブルを作ることに注意せよ．また，クエリごとに空の `unordered_set` が用意されることに注意せよ． 即ち，各クエリは独立である．

あなたの目標はコスト $1\,000\,000$ 以下でバケットの個数 n を特定することである．

実装の詳細

あなたは以下の関数を実装する必要がある：

```
int hack()
```

- この関数は隠された値である n を返さなければならない．
- この関数は各テストケースにおいて複数回呼び出されることがある．各呼び出しは，独立のシナリオとして処理を行うこと．

この関数の中で，あなたは以下の関数を呼び出すことができる：

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

- x: 相異なる整数からなる配列． また各 i において $1 \leq x[i] \leq 10^{18}$ である．
- この関数は配列 x の要素を `unordered_set` に挿入したときのハッシュの衝突回数を返す．
- この関数は複数回呼び出すことができる．ただし，一回の `hack()` の呼び出しにおいて配列 x の長さの合計は $1\ 000\ 000$ を超えてはならない．

関数 `hack()` は複数回呼び出されるため，前回の呼び出しのデータが今回の呼び出しのデータに影響を与えることに注意せよ．特に，グローバル変数の状態に注意せよ．

各テストケースにおける呼び出しにおいて，コスト上限は $1\ 000\ 000$ である． すなわち，`hack()` が t 回呼び出されたとき，コストの合計は $t \times 1\ 000\ 000$ を超えてはならない． ただし，各 `hack()` の呼び出しのコストは $1\ 000\ 000$ を超えてはならない．

インタラクタはアダプティブではない．つまり、値 n はインタラクションの開始前に固定されている．

例

仮にマルチテストが 2 つあるときを考えると，採点プログラムは以下のように呼び出す：

```
hack()
```

関数の中で下のようにあなたが関数を呼び出したとする：

呼び出し	戻り値
`collisions([2, 15, 7, 27, 8, 30])`	4
`collisions([1, 2, 3])`	0
`collisions([10, 20, 30, 40, 50])`	10

これによって n が 5 であると判明したら，`hack()` は 5 を返さなければならない．

その後，採点プログラムは次の呼び出しを行う：

```
hack()
```

関数の中で下のようにあなたが関数を呼び出したとする：

呼び出し	戻り値
`collisions([1, 3])`	1
`collisions([2, 4])`	1

この 2 つの条件を満たす n は 2 のみである．そのため，`hack()` は 2 を返さなければならない．

制約

- $1 \leq t \leq 10$． t はマルチテストの数である．
- $2 \leq n \leq 10^9$
- 各 `collisions()` において $1 \leq x[i] \leq 10^{18}$ である．

小課題

1. (8 点) $n \leq 500\ 000$
2. (17 点) $n \leq 1\ 000\ 000$
3. (75 点) 追加の制約はない．

最後の小課題では部分点を得ることができる． すべてのテストケースにおける `hack()` の呼び出しで使用したコストの合計の最大値を q とする． この小課題であなたが得る得点は以下の表に従って q にもとづいて計算される：

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

いずれかのテストケースにおいて `collisions()` の呼び出しが実装の詳細で定められた制約を満たさない場合や，`hack()` の戻り値が正しくない場合はその小課題の得点は 0 点である．

採点プログラムのサンプル

採点プログラムのサンプルは以下の形式で入力を受け取る：

- 1 行目：t

その後，t 行にわたって各行に 1 つずつ n が与えられる：

- 1 行目：n

各テストケースごとに，`hack()` の戻り値を m ，すべてのクエリでかかったコストの合計を c とすると，採点プログラムのサンプルは以下の形式で出力する：

- 1 行目：$m\ c$