Retas!

Satu jam telah berlalu dalam suatu kontes Codeforces ketika Anda sadar bahwa ada kontestan lain di ruangan Anda yang menyelesaikan sebuah soal menggunakan `unordered_set`. Saatnya meretas solusi tersebut!

Anda tahu bahwa `unordered_set` menggunakan *hash table* dengan n buah *bucket* yang dinomori dari 0 hingga $n-1$. Sayangnya, Anda tidak mengetahui nilai n dan ingin mencari tahu nilai tersebut.

Saat Anda memasukkan bilangan bulat x ke dalam *hash table*, bilangan bulat tersebut akan dimasukkan ke dalam *bucket* nomor $(x \bmod n)$. Apabila sudah terdapat b elemen di dalam *bucket* tersebut sebelum x dimasukkan, akan terjadi b kali *hash collision*.

Dengan memberikan *array* $x = [x[0], x[1], \ldots, x[k-1]]$ berisi k bilangan bulat berbeda kepada interaktor, Anda bisa menanyakan banyaknya *hash collision* keseluruhan yang terjadi saat membuat `unordered_set` baru dan memasukkan elemen-elemen $x[0], x[1], \ldots, x[k-1]$ secara berurutan. Pertanyaan tersebut membutuhkan biaya sebesar k.

Contohnya, jika $n = 5$, maka memberikan $x = [2, 15, 7, 27, 8, 30]$ kepada interaktor akan mengakibatkan 4 *collision* secara keseluruhan:

Operasi	*Collision* baru	Isi *bucket*
kondisi awal	$-$	$[], [], [], [], []$
memasukkan $x[0] = 2$	0	$[], [], [2], [], []$
memasukkan $x[1] = 15$	0	$[15], [], [2], [], []$
memasukkan $x[2] = 7$	1	$[15], [], [2, 7], [], []$
memasukkan $x[3] = 27$	2	$[15], [], [2, 7, 27], [], []$
memasukkan $x[4] = 8$	0	$[15], [], [2, 7, 27], [8], []$
memasukkan $x[5] = 30$	1	$[15, 30], [], [2, 7, 27], [8], []$

Perhatikan bahwa interaktor membuat *hash table* dengan memasukkan elemen-elemen secara berurutan ke dalam `unordered_set` yang awalnya kosong. Selain itu, untuk setiap pertanyaan, akan dibuat `unordered_set` kosong yang baru. Dengan kata lain, setiap pertanyaan bersifat independen.

Tugas Anda adalah mencari tahu banyaknya *bucket*, n, dengan jumlah biaya seluruh pertanyaan Anda tidak melebihi $1\ 000\ 000$.

Detail Implementasi

Anda harus mengimplementasikan prosedur berikut:

```
int hack()
```

- Prosedur ini harus mengembalikan sebuah bilangan bulat, yakni nilai n yang perlu Anda cari.
- Pada suatu kasus uji, prosedur ini akan dipanggil paling banyak t kali. Setiap pemanggilan prosedur harus Anda anggap sebagai skenario yang terpisah.

Di dalam prosedur di atas, Anda dapat memanggil fungsi berikut:

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

- x: *array* yang berisi bilangan bulat berbeda. Harus terpenuhi bahwa $1 \leq x[i] \leq 10^{18}$ untuk semua i.
- Fungsi ini akan mengembalikan banyaknya *collision* yang terjadi saat memasukkan *array* x secara berurutan ke dalam suatu `unordered_set`.
- Fungsi ini dapat dipanggil berkali-kali. Dalam satu pemanggilan `hack()`, jumlah panjang *array* x yang diberikan ke dalam fungsi ini tidak boleh melebihi $1\ 000\ 000$.

Catatan: **Karena prosedur `hack()` dapat dipanggil lebih dari satu kali, Anda perlu memperhatikan dampak data yang tersisa dari pemanggilan prosedur sebelumnya, khususnya terhadap variabel-variabel global.**

Batasan $1\ 000\ 000$ berlaku untuk setiap pemanggilan `hack()`. Secara umum, jika ada t kali pemanggilan `hack()`, Anda dapat menggunakan biaya keseluruhan tidak lebih dari $t \times 1\ 000\ 000$, dengan masing-masing pemanggilan `hack()` menggunakan jumlah biaya tidak lebih dari $1\ 000\ 000$.

Grader tidak bersifat adaptif: nilai dari n sudah ditetapkan sebelum pemanggilan `hack()` dilakukan.

Contoh

Misalkan terdapat kasus uji yang terdiri dari dua pemanggilan `hack()`. Pertama, *grader* akan melakukan pemanggilan berikut:

```
hack()
```

Misalkan Anda melakukan pemanggilan-pemanggilan berikut di dalam prosedur `hack()`:

Pemanggilan	Nilai kembalian
`collisions([2, 15, 7, 27, 8, 30])`	4
`collisions([1, 2, 3])`	0
`collisions([10, 20, 30, 40, 50])`	10

Setelah itu, jika Anda menemukan bahwa nilai $n = 5$, prosedur `hack()` harus mengembalikan 5.

Lalu, *grader* akan melakukan pemanggilan berikut:

```
hack()
```

Misalkan Anda melakukan pemanggilan-pemanggilan berikut di dalam prosedur `hack()`.

Pemanggilan	Nilai kembalian
`collisions([1, 3])`	1
`collisions([2, 4])`	1

Satu-satunya nilai n yang sesuai dengan semua nilai kembalian adalah 2. Maka, prosedur `hack()` harus mengembalikan 2.

Batasan

- $1 \leq t \leq 10$, dengan t adalah banyaknya pemanggilan `hack()` di dalam setiap kasus uji.
- $2 \leq n \leq 10^9$
- $1 \leq x[i] \leq 10^{18}$ untuk setiap pemanggilan `collisions()`.

Subsoal

1. (8 poin) $n \leq 500\ 000$
2. (17 poin) $n \leq 1\ 000\ 000$
3. (75 poin) Tidak ada batasan tambahan.

Pada subsoal terakhir, Anda bisa mendapatkan nilai parsial. Misalkan q merupakan jumlah biaya maksimum di antara semua pemanggilan prosedur `hack()` dari semua kasus uji pada subsoal ini. Nilai Anda untuk subsoal ini dihitung berdasarkan tabel berikut:

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

Jika pada suatu kasus uji, terdapat pemanggilan prosedur `collisions()` yang tidak sesuai dengan batasan yang tertera pada bagian Detail Implementasi, atau nilai yang dikembalikan oleh prosedur `hack()` tidak tepat, maka nilai Anda untuk subsoal yang mengandung kasus uji tersebut adalah 0.

Contoh *Grader*

Contoh *grader* membaca masukan dengan format berikut:

- Baris 1: t

Setelah itu, terdapat t baris yang masing-masing berisi nilai n untuk setiap pemanggilan `hack()`:

- Baris 1: n

Untuk setiap pemanggilan `hack()`, misalkan m merupakan nilai kembalian `hack()`, dan c merupakan jumlah biaya dari semua pemanggilan `collisions()`. Contoh *grader* akan mengeluarkan jawaban Anda dalam format berikut:

- Baris 1: $m\ c$