Hack! (hack)

코드포스 대회가 시작한 지 한시간이 지났을 때 당신은 다른 참가자가 `unordered_set`을 이용하여 문제를 푼 것을 알아냈다. 해킹을 해보자!

`unordered set`은 n개의 버킷으로 된 해시 테이블을 이용한다. 각 버킷은 0부터 $n-1$까지 수로 번호가 매겨져 있다. 불행히도, 당신은 n 값을 알지 못하며 이를 알아내고 싶다.

정수 x를 해시 테이블에 넣었을 때, 이 값은 버킷 $(x \bmod n)$ 에 들어간다. 만약 이 값을 삽입하기 전에 b개의 원소가 이 버킷에 들어 있었다면, b번 해시 충돌이 벌어지게 된다.

k개의 다른 정수값 $x[0], x[1], \ldots, x[k-1]$을 해시 테이블에 삽입할 때, 우리는 이 수들을 저장하면서 충돌이 몇 번 발생했는지 알아낼 수 있다. 단, 한 번에 k개의 정수를 해시 테이블에 삽입할 때마다 비용 k가 발생한다. 충돌 수가 비용이 아니라는데 유의하라.

예를 들어, 만약 $n = 5$이라면, $x = [2, 15, 7, 27, 8, 30]$을 처음에 비어 있는 해시 테이블에 삽입하면 충돌이 모두 4번 발생한다.

작업	새로 발생하는 충돌 수	버킷의 상태
초기 상태	–	$[], [], [], [], []$
$x[0] = 2$ 삽입	0	$[], [], [2], [], []$
$x[1] = 15$ 삽입	0	$[15], [], [2], [], []$
$x[2] = 7$ 삽입	1	$[15], [], [2, 7], [], []$
$x[3] = 27$ 삽입	2	$[15], [], [2, 7, 27], [], []$
$x[4] = 8$ 삽입	0	$[15], [], [2, 7, 27], [8], []$
$x[5] = 30$ 삽입	1	$[15, 30], [], [2, 7, 27], [8], []$

처음 비어있는 해시 테이블에 정수를 순서대로 삽입하면서 해시 테이블을 만든는다는 점에 유의하라. 각 질의마다 새로운 공집합들을 만들어내기 때문에, 모든 질의는 서로 독립적이다.

당신은 버킷의 수 n을 $1\,000\,000$ 이하의 비용으로 알아내야 한다.

Implementation details

다음 함수를 구현해야 한다.

```
int hack()
```

- 이 함수의 리턴값은 숨겨진 정수값 n이다.
- 각 테스트케이스마다, 그레이더는 이 함수를 한 번 이상 호출할 수 있다. 각 호출은 별도의 시나리오로 처리되어야 한다.

이 함수 내부에서, 다음 함수를 호출할 수 있다.

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

- x: 서로 다른 수를 저장하고 있는 배열로, 각각의 i에 대해 $1 \leq x[i] \leq 10^{18}$.
- 이 함수는 x의 원소들을 `unordered set`에 삽입하면서 발생하는 충돌 횟수를 리턴한다.
- 이 함수는 여러번 호출될 수 있다. `hack()`을 한 번 호출하는 동안 이 함수를 호출하면서 주어진 x의 길이의 총합은 $1\ 000\ 000$ 이하여야 한다.

`hack()`은 한 번 이상 호출될 수 있기 때문에, 이전 호출의 결과가 현재 호출에 미칠 영향에 주의해야 한다. 특히 전역변수에 저장된 값들에 주의하라.

각 테스트케이스마다 제약은 $1\ 000\ 000$이다. 즉, `hack()`을 t번 호출했다면, 전체 비용은 $t \times 1\ 000\ 000$ 이하이고, 각 `hack()` 호출의 비용도 $1\ 000\ 000$이어야 한다..

n 값은 적응적(adaptive)하지 않다. 즉, 해시 테이블에 첫번째 삽입 연산을 하기 전에 이미 정해져 있다.

Example

다음과 같은 두 테스트케이스를 고려해보자. 그레이더가 먼저 다음 함수 호출을 한다.

```
hack()
```

이 함수 내부에서 당신은 다음과 같이 함수 호출을 했다.

함수 호출	리턴값
`collisions([2, 15, 7, 27, 8, 30])`	4
`collisions([1, 2, 3])`	0
`collisions([10, 20, 30, 40, 50])`	10

만약 n 값이 5라는 것을 알아내었다면, `hack()`의 리턴값은 5여야 한다.

그레이더가 이제 다음 함수 호출을 했다고 하자.

```
hack()
```

이 함수 내부에서, 당신이 다음과 같이 함수 호출을 했다고 하자.

함수 호출	리턴값
`collisions([1, 3])`	1
`collisions([2, 4])`	1

위 질의를 만족하는 유일한 n 값은 2이다. 따라서, `hack()`의 리턴값은 2여야 한다.

Constraints

- $1 \le t \le 10$, t는 테스트케이스의 수.
- $2 \le n \le 10^9$
- 각각 `collisions()`에 대해 $1 \le x[i] \le 10^{18}$.

Subtasks

1. (8 points) $n \le 500\ 000$
2. (17 points) $n \le 1\ 000\ 000$
3. (75 points) 추가적인 제약 조건이 없다.

마지막 서브태스크에서는 부분 점수를 받을 수 있다. q가 이 서브태스크의 모든 테스트케이스에서 호출한 `hack()` 함수를 호출할 때 발생한 총 비용 중 최대값이라고 하자. 이 서브태스크에서 점수 배점은 다음 표와 같다.

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

만약, 테스트케이스 중 어느 하나에서라도 `collisions()`의 호출이 Implementation Details에서 설명한 제약 조건을 따르지 않았거나, `hack()`의 리턴값이 틀렸다면, 이 서브태스크에서 0점을 받게 된다.

Sample Grader

샘플 그레이더는 다음 양식으로 입력을 읽는다.

- line 1: t

다음 t 줄에는 한 줄에 하나씩 n이 주어진다.

- line 1: n

각 테스트케이스마다, m이 `hack()`의 리턴값이고, c가 모든 질의에 대한 비용의 총합이라고 하자. 샘플 그레이더는 다음 양식으로 출력한다.

- line 1: m c