Hack! (hack)

Cela fait une heure que le concours Codeforces dure, et vous remarquez qu'un autre participant a résolu un problème en utilisant un `unordered_set`. Il est temps de hacker!

Vous savez qu'un `unordered_set` utilise une table de hachage avec n cases, numérotées de 0 à $n-1$. Malheureusement, vous ne connaissez pas la valeur de n et vous voulez la trouver.

Quand un entier x est inséré dans la table, il est inséré dans la case numéro $(x \bmod n)$. S'il y avait b éléments dans cette case avant, cela provoquera b collisions.

En entrant k entiers distincts $x[0], x[1], \ldots, x[k-1]$ au juge, il vous retourne le nombre total de collisions provoquées lors de l'insertion des k entiers dans un `unordered_set` vide. Par contre, entrer ces k entiers lors d'une requête aura un coût valant k.

Par exemple, si $n = 5$, donner au juge $x = [2, 15, 7, 27, 8, 30]$ provoquera 4 collisions au total:

Opération	Nouvelles Collisions	Cases
Début	$-$	$[], [], [], [], []$
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	$[15, 30], [], [2, 7, 27], [8], []$

Notez que le juge crée la table de hachage en insérant les éléments l'un après l'autre dans un `unordered_set` préalablement vide, et un nouvel emsemble sera créé pour chaque requête. Pour résumer, les requêtes sont indépendantes.

Votre but est de trouver la valeur de n, avec un coût total inférieur ou égal à $1\ 000\ 000$.

Détails d'implémentation

Vous devez implémenter la procédure:

```
int hack()
```

- La procédure doit retourner un entier – la valeur de n.
- Pour chaque test, le juge peut appeler cette procédure plus d'une fois. Chaque appel doit être considéré comme un nouveau scénario.

Dans cette procédure, vous pouvez appeler la procédure suivante:

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

- x: un tableau d'entiers distincts, où $1 \leq x[i] \leq 10^{18}$ pour tout i.
- La fonction retourne le nombre de collisions créées par l'insertion de x dans un `unordered_set`.
- Cette procédure peut être appelée plusieurs fois. La somme des longueurs de x pour tous les appels durant un appel de `hack()` ne doit pas dépasser $1\ 000\ 000$.

Note: **Vu que la procédure `hack()` risque d'être appelée plus d'une fois, faites attention à l'impact des données restantes de l'appel précédent sur l'appel actuel, en particulier les valeurs des variables globales.**

La limite du coût inférieur ou égal à $1\ 000\ 000$ s'applique à chaque test. En général, si t appels sont effectués à `hack()`, vous pouvez utiliser un coût total de $t \times 1\ 000\ 000$, avec chaque appel à `hack()` utilisant un coût d'au plus $1\ 000\ 000$.

Le juge n'est pas adaptatif, i.e.les valeurs de n sont fixées avant le début de l'interaction.

Exemple

Supposer, qu'il y a 2 multitests. Le juge va effectuer l'appel suivant:

```
hack()
```

Supposons que dans la procédure, vous effectuez les appels suivants :

Appel	Valeur retournée
`collisions([2, 15, 7, 27, 8, 30])`	4
`collisions([1, 2, 3])`	0
`collisions([10, 20, 30, 40, 50])`	10

Après cela, si vous concluez que la valeur de n est 5, la procédure `hack()` doit retourner 5.

Le juge va effectuer un autre appel:

```
hack()
```

Supposons que dans la procédure, vous effectuez les appels suivants :

Appel	Valeur retournée
`collisions([1, 3])`	1
`collisions([2, 4])`	1

Le seul n satisfaisant les requêtes est 2. Donc, la procédure `hack()` retourne 2.

Contraintes

- $1 \leq t \leq 10$, où t est le nombre de multitests.
- $2 \leq n \leq 10^9$
- $1 \leq x[i] \leq 10^{18}$ pour chaque appel à `collisions()`.

Sous-taches

1. (8 points) $n \leq 500\ 000$
2. (17 points) $n \leq 1\ 000\ 000$
3. (75 points) Pas de contraintes additionnelles.

Dans la dernière sous-tâche, vous pouvez obtenir un score partiel. Soit q le coût total maximal parmi tous les appels de `hack()` parmi tous les test de la sous-tâche. Votre score pour cette sous-tâche est calculé comme-suit :

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

Si, dans n'importe lequel des tests, les appels à la procédure `collisions()` ne sont pas conformes au contraintes décrites dans les Détails d'implémentation, ou bien que le nombre retourné par `hack()` est incorrect, le score de votre solution pour cette sous-tâche sera 0.

Evaluateur d'exemple (grader)

L'évaluateur d'exemple lit l'entrée au format suivant:

- ligne 1: t

Ensuite, t lignes suivent, chacune contenant une valeur de n:

- ligne 1: n

Pour chaque test, soit m la valeur retournée par `hack()`, et c le coût total. L'évaluateur affiche votre réponse au format suivant:

- ligne 1: m c