Make All Equal (`equal`)

Si vous visitez l'enceinte mégalithique préhistorique de Stonehenge à l'aube du plus long jour de l'année, les druides vous mettront au défi de participer à l'ancien et sacré *jeu des galets*. Pour commencer, vos yeux seront bandés de sorte à ce que vous ne puissiez rien voir.

Ensuite, la cheffe druide vous dira qu'elle a N piles de galets sur une ligne, **où N est une puissance de deux** ($N = 2^k$ pour un certain entier k).

Les piles sont numérotées de 0 à $N-1$ inclus. Chaque pile a une hauteur H_i, qui est un entier strictement positif, et les piles sont ordonnées de la plus petite à la plus grande ($H_0 \leq H_1 \leq \ldots \leq H_{N-1}$). La cheffe druide vous donne la valeur de N, mais pas les hauteurs initiales.

Vous pouvez ensuite choisir une action de l'un des types suivants :

1. Choisir un entier positif X et un sous-ensemble de piles S. Les druides ajouteront alors X galets à chacune des piles dans S, puis réordonneront les piles de la plus petite à la plus grande.
2. Choisir deux piles i et j. Les druides vous diront si ces deux piles ont actuellement la même hauteur.

Votre but est d'atteindre une configuration où toutes les piles ont la même hauteur. Vous devez effectuer au plus Q_{add} actions du premier type et au plus Q_{compare} actions du second type (voir section Score).

Implémentation

Vous devrez soumettre une seul fichier source `.cpp`.

> ☞ Parmi les fichiers fournis avec cette tâche, vous trouverez un fichier modèle `equal.cpp` avec une implémentation d'exemple.

Vous devez implémenter la fonction suivante :

C++
```cpp
void make_all_equal(int N, int Q_add, int Q_compare);
```

— L'entier N représente le nombre de piles.
— L'entier Q_{add} représente le nombre maximum d'appels à `add`.
— L'entier Q_{compare} représente le nombre maximum d'appels à `compare`.

Vous pouvez appeler les fonctions suivantes :

C++
```cpp
void add(vector<int> S, long long X);
```

— Le vecteur S doit contenir des entiers **distincts** entre 0 et $N-1$ inclus.
— L'entier X doit être entre 0 et 10^{12} inclus.
— Cette fonction augmentera H_i de X pour tout i dans S. Ensuite, elle triera les hauteurs dans l'ordre croissant ($H_0 \leq \ldots \leq H_{N-1}$).
— Cette fonction peut être appelée au plus Q_{add} fois.

C++
```cpp
bool compare(int i, int j);
```

— i et j doivent être entre 0 et $N-1$ inclus.
— Cette fonction renverra `true` si la i-ème plus petite pile et la j-ème plus petite pile ont la même hauteur (autrement dit, si $H_i = H_j$), et `false` sinon.

— Cette fonction peut être appelée au plus Q_{compare} fois.

Évaluateur

Le dossier de la tâche contient une version simplifiée du grader, qui vous permet de tester votre solutions localement. Ce grader simplifié lit les données d'entrée depuis `stdin`, appelle les fonctions que vous devez implémenter, et finalement écrit le résultat dans `stdout`

L'entrée contient deux lignes, dans le format suivant :

— Ligne 1 : les entiers N, Q_{add} et Q_{compare}, séparés par un espace.

— Ligne 2 : les entiers H_i, séparés par des espaces.

Si votre programme est jugé **Accepted**, l'évaluateur d'exemple affiche `Accepted: add=U, compare=V` où U et V sont le nombre de fois que vous avez appelé `add` et `compare`.

Si votre programme est jugé **Wrong Answer**, l'évaluateur d'exemple affiche `Wrong Answer: MSG`, où `MSG` est l'un des messages suivants :

Message	Signification
`too many calls to add`	Vous avez appelé `add` plus de Q_{add} fois.
`X out of range`	L'entier X donné à `add` n'est pas entre 0 et 10^{12} inclus.
`index in S out of range`	Un élement du vecteur S passé à `add` n'est pas entre 0 et $N-1$ inclus.
`indices in S not distinct`	Il y a deux éléments égaux dans le vecteur S passé à `add`.
`too many calls to compare`	Vous avez appelé `compare` plus de Q_{compare} fois.
`i out of range`	L'entier i passé à `compare` n'est pas entre 0 et $N-1$ inclus.
`j out of range`	L'entier j passé à `compare` n'est pas entre 0 et $N-1$ inclus.
`heights are not equal`	Après avoir appelé `make_all_equal`, les piles n'ont pas toutes la même hauteur.

Contraintes

— $2 \leq N \leq 2048$, et N **est une puissance de deux**.

— Les hauteurs initiales satisfont $1 \leq H_0 \leq \cdots \leq H_{N-1} \leq 1\,000\,000$.

Score

Votre programme sera testé sur un ensemble de cas de test groupés par sous-tâche. Afin d'obtenir les points associés à une sous-tâche, vous devez répondre correctement à tout les cas de test associés.

— **Sous-tâche 1** [**0 points**]: Test d'exemple.

— **Sous-tâche 2** [**5 points**]: $N = 2$, $H_i \leq 4$, $Q_{\text{add}} \geq 3000$ et $Q_{\text{compare}} \geq 4000$.

— **Sous-tâche 3** [**16 points**]: $N = 2$, $H_i \leq 1\,000\,000$, $Q_{\text{add}} \geq 22$ et $Q_{\text{compare}} \geq 1$.

— **Sous-tâche 4** [**15 points**]: $N = 256$, $H_i \leq 10$, $Q_{\text{add}} \geq 3000$ et $Q_{\text{compare}} \geq 255$.

— **Sous-tâche 5** [**18 points**]: $N = 4$, $H_i \leq 1\,000\,000$, $Q_{\text{add}} \geq 45$ et $Q_{\text{compare}} \geq 3$.

— **Sous-tâche 6** [**22 points**]: $N = 2048$, $H_i \leq 1\,000\,000$, $Q_{\text{add}} \geq 298$ et $Q_{\text{compare}} \geq 4000$ **et initialement, il y a au plus deux hauteurs distinctes**. Autrement dit, pour tout i, $H_0 = H_i$ ou $H_i = H_{N-1}$.

— **Sous-tâche 7** [**24 points**]: $N = 2048$, $H_i \leq 1\,000\,000$, $Q_{\text{add}} \geq 298$ et $Q_{\text{compare}} \geq 2047$.

Notez qu'aucune de ces sous-tâches ne contient tous les tests.

Exemples

Entrée	Communication d'exemple			
	Appel	Retour	Hauteurs	Explication
4 45 3 1 4 5 5	`compare(1, 2)`	`false`		La solution demande si $H_1 = H_2$. Vu que $H_1 = 4$ et $H_2 = 5$, c'est faux.
	`compare(2, 3)`	`true`		La solution demande si $H_2 = H_3$. Vu que $H_2 = H_3 = 5$, c'est vrai.
	`add({0, 1}, 2)`		$3, 5, 5, 6$	Après avoir ajouté les galets, les nouvelles hauteurs sont $[\mathbf{3}, \mathbf{6}, 5, 5]$. Après les avoir triées, elles deviennent $H = [3, 5, 5, 6]$.
	`compare(1, 2)`	`true`		La solution demande si $H_1 = H_2$. Vu que $H_1 = H_2 = 5$, c'est vrai.
	`add({0, 3}, 3)`		$5, 5, 6, 9$	Après avoir ajouté les galets, les nouvelles hauteurs sont $[\mathbf{6}, 5, 5, \mathbf{9}]$. Après les avoir triées, elles deviennent $H = [5, 5, 6, 9]$.
	`add({0, 1}, 4)`		$6, 9, 9, 9$	Après avoir ajouté les galets, les nouvelles hauteurs sont $[\mathbf{9}, \mathbf{9}, 6, 9]$. Après les avoir triées, elles deviennent $H = [6, 9, 9, 9]$.
	`add({0}, 3)`		$9, 9, 9, 9$	Après avoir ajouté les galets, les nouvelles hauteurs sont $[\mathbf{9}, 9, 9, 9]$. Après les avoir triées, elles deviennent $H = [9, 9, 9, 9]$.