Make All Equal (`equal`)

Si visitas el círculo de piedra prehistórico de Stonehenge durante el amanecer del día más largo del año, los druidas te retarán a participar en el vetusto y sagrado *juego de las piedras*. Primero, te vendarán los ojos, de forma que no podrás ver nada.

Entonces, la druida jefe te dirá que tiene N pilas de piedras, **donde N es una potencia de dos** ($N = 2^k$ para algún número entero k).

Las pilas se indexan de 0 a $N-1$. Cada pila tiene una altura H_i, que es un entero positivo, y las pilas se ordenan de menor a mayor altura ($H_0 \leqslant H_1 \leqslant \ldots \leqslant H_{N-1}$). La druida jefe te dice el valor de N, pero no las alturas iniciales.

A continuación puedes elegir realizar acciones de uno de los siguientes tipos:

1. Escoges un número positivo X y un subconjunto de las pilas S. Los druidas añadirán X piedras a cada una de las pilas en S, y después reordenarán las pilas de menor a mayor altura.
2. Escoges dos pilas i y j. Los druidas te dirán si esas dos pilas actualmente tienen la misma altura o no.

Tu objetivo es llegar a una configuración donde todas las pilas tengan la misma altura. Puedes realizar como mucho Q_{add} acciones del primer tipo, y como mucho Q_{compare} acciones del segundo tipo (véase la sección de Puntuación).

Implementación

Deberás presentar un único fichero `.cpp`.

> ☞ Entre los archivos adjuntos encontrarás una plantilla `equal.cpp` con una implementación de ejemplo.

Tienes que implementar la siguiente función:

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

- El entero N representa el número de pilas.
- El entero Q_{add} representa el máximo número de veces que puedes llamar a `add`.
- El entero Q_{compare} representa el máximo número de veces que puedes llamar a `compare`.

Puedes llamar a las siguientes funciones:

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

- El vector S debe contener enteros **distintos** entre 0 y $N-1$ inclusive.
- El entero X debe estar entre 0 y 10^{12} inclusive.
- Esta función incrementará H_i por X por cada i en S. Después, ordenará las alturas en orden creciente ($H_0 \leqslant \ldots \leqslant H_{N-1}$).
- Esta función debe ser llamada como mucho Q_{add} veces.

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

- i y j deben estar entre 0 y $N-1$ inclusive.
- Esta función retornará `true` si la i-ésima pila más pequeña y la j-ésima pila más pequeña tienen la misma altura actualmente (es decir, si $H_i = H_j$), y `false` si no.
- Esta función debe ser llamada como mucho Q_{compare} veces.

Sample Grader

El directorio de la tarea contiene una versión simplificada del grader del jurado, que puedes utilizar para probar tu solución localmente. El grader simplificado lee los datos de entrada de `stdin`, llama a la función que debes implementar, y finalmente escribe la salida a `stdout`.

La entrada se compone de dos líneas, que contienen:

- Línea 1: los enteros N, Q_{add} y Q_{compare}, separados por espacios.
- Línea 2: los enteros H_i, separados por espacios.

Si tu programa es juzgado como **Accepted**, el grader de ejemplo imprime `Accepted: add=U, compare=V` donde U y V son las cantidades de veces que has llamado `add` y `compare`.

Si tu programa es juzgado como **Wrong Answer**, el grader de ejemplo imprime `Wrong Answer: MSG`, donde `MSG` es uno de:

Message	Meaning
`too many calls to add`	Llamaste a `add` más de Q_{add} veces.
`X out of range`	El entero X dado a `add` no está entre 0 y 10^{12} inclusive.
`index in S out of range`	Un elemento del vector S dado a `add` no está entre 0 y $N-1$ inclusive.
`indices in S not distinct`	Hay dos elementos iguales en el vector S dado a `add`.
`too many calls to compare`	Llamaste `compare` más de Q_{compare} veces.
`i out of range`	El entero i dado a `compare` no está entre 0 y $N-1$ inclusive.
`j out of range`	El entero j dado a `compare` no está entre 0 y $N-1$ inclusive.
`heights are not equal`	Tras llamar a `make_all_equal`, hay dos pilas con alturas diferentes.

Restricciones

- $2 \leqslant N \leqslant 2048$, y N **es una potencia de 2**.
- Las alturas iniciales satisfacen $1 \leqslant H_0 \leqslant \cdots \leqslant H_{N-1} \leqslant 1\,000\,000$.

Puntuación

Tu programa será probado en un conjunto de casos de prueba agrupados por subtarea. Para obtener la puntuación asociada a una subtarea, debes resolver correctamente todos los casos de prueba que contiene.

- **Subtask 1** [**0 puntos**]: Casos de ejemplo.
- **Subtask 2** [**5 puntos**]: $N = 2$, $H_i \leqslant 4$, $Q_{\text{add}} = 3000$ y $Q_{\text{compare}} = 4000$.
- **Subtask 3** [**16 puntos**]: $N = 2$, $H_i \leqslant 1\,000\,000$, $Q_{\text{add}} = 22$ y $Q_{\text{compare}} = 1$.
- **Subtask 4** [**15 puntos**]: $N = 256$, $H_i \leqslant 10$, $Q_{\text{add}} = 3000$ y $Q_{\text{compare}} = 255$.

- **Subtask 5** [**18 puntos**]: $N = 4$, $H_i \leqslant 1\,000\,000$, $Q_{\text{add}} = 45$ y $Q_{\text{compare}} = 3$ **e inicialmente hay como mucho dos alturas distintas**. Es decir, por cada i, $H_0 = H_i$ o $H_i = H_{N-1}$.
- **Subtask 6** [**22 puntos**]: $N = 2048$, $H_i \leqslant 1\,000\,000$, $Q_{\text{add}} = 298$ y $Q_{\text{compare}} = 4000$.
- **Subtask 7** [**24 puntos**]: $N = 2048$, $H_i \leqslant 1\,000\,000$, $Q_{\text{add}} = 298$ y $Q_{\text{compare}} = 2047$.

Nótese que ninguna de las subtareas contiene todos los casos de prueba.

Ejemplos de entrada/salida

Input	**Sample Communication**			
	Calls	Return	Heights	Explanation
`4 45 3` `1 4 5 5`	`compare(1, 2)`	`false`		La solución pregunta si $H_1 = H_2$. Como $H_1 = 4$ y $H_2 = 5$, es falso.
	`compare(2, 3)`	`true`		La solución pregunta si $H_2 = H_3$. Como $H_2 = H_3 = 5$, es cierto.
	`add({0, 1}, 2)`		$3, 5, 5, 6$	Después de añadir las piedras, las nuevas alturas son $[\mathbf{3}, \mathbf{6}, 5, 5]$. Tras reordenarlas, $H = [3, 5, 5, 6]$.
	`compare(1, 2)`	`true`		La solución pregunta si $H_1 = H_2$. Como $H_1 = H_2 = 5$, es cierto.
	`add({0, 3}, 3)`		$5, 5, 6, 9$	Después de añadir las piedras, las nuevas alturas son $[\mathbf{6}, 5, 5, \mathbf{9}]$. Tras reordenarlas, $H = [5, 5, 6, 9]$.
	`add({0, 1}, 4)`		$6, 9, 9, 9$	Después de añadir las piedras, las nuevas alturas son $[\mathbf{9}, \mathbf{9}, 6, 9]$. Tras reordenarlas, $H = [6, 9, 9, 9]$.
	`add({0}, 3)`		$9, 9, 9, 9$	Después de añadir las piedras, las nuevas alturas son $[\mathbf{9}, 9, 9, 9]$. Tras reordenarlas, $H = [9, 9, 9, 9]$.