Make All Equal (equal)

Wenn Du bei Sonnenaufgang am längsten Tag des Jahres den prähistorischen Steinkreis von Stonehenge besuchst, werden Dich die Druiden herausfordern, am alten und heiligen *Kieselspiel* teilzunehmen. Zuerst wirst Du die Augen verbunden bekommen, sodass Du nichts sehen kannst.

Dann wird Dir die Häuptlingsdruidin mitteilen, dass sie N Steinhaufen in einer Linie hat, **wobei N eine Zweierpotenz ist** ($N = 2^k$ für einen bestimmten Ganzzahlwert k).

Die Haufen sind von 0 bis $N-1$ indiziert. Jeder Haufen hat eine Höhe H_i, die eine positive Ganzzahl ist, und die Haufen sind von klein nach gross geordnet ($H_0 \leq H_1 \leq \ldots \leq H_{N-1}$). Die Häuptlingsdruidin sagt Dir den Wert von N, aber nicht die Anfangshöhen.

Du kannst dann Aktionen einer der folgenden Typen wählen:

1. Wähle eine positive Zahl X und eine Teilmenge der Haufen S. Die Druiden werden dann X Steine zu jedem Haufen in S hinzufügen und die Haufen anschliessend von klein nach gross neu ordnen.
2. Wähle zwei Haufen i und j. Die Druiden werden Dir dann mitteilen, ob diese beiden Haufen derzeit die gleiche Höhe haben.

Dein Ziel ist es, eine Konfiguration zu erreichen, bei der alle Haufen die gleiche Höhe haben. Du hast dafür höchstens Q_{add} Aktionen des ersten Typs und höchstens Q_{compare} Aktionen des zweiten Typs zur Verfügung (siehe Abschnitt Bewertung).

Implementierung

Du musst eine einzelne `.cpp` Quelldatei einreichen.

> ☞ Unter den Anhängen zu dieser Aufgabe findest Du eine Vorlage `equal.cpp` mit einer Beispielimplementierung.

Du musst die folgende Funktion implementieren:

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

- Die Ganzzahl N stellt die Anzahl der Haufen dar.
- Die Ganzzahl Q_{add} gibt die maximale Anzahl an, wie oft Du `add` aufrufen kannst.
- Die Ganzzahl Q_{compare} gibt die maximale Anzahl an, wie oft Du `compare` aufrufen kannst.

Du kannst die folgenden Funktionen aufrufen:

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

- Der Vektor S muss **unterschiedliche** Ganzzahlen zwischen 0 und $N-1$ (einschliesslich) enthalten.
- Die Ganzzahl X muss zwischen 0 und 10^{12} (einschliesslich) liegen.
- Diese Funktion wird H_i für jedes i in S um X erhöhen. Dann wird sie die Höhen in aufsteigender Reihenfolge sortieren ($H_0 \leq \ldots \leq H_{N-1}$).
- Diese Funktion kann höchstens Q_{add} Mal aufgerufen werden.

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

- i und j müssen zwischen 0 und $N-1$ (einschliesslich) liegen.
- Die Funktion wird `true` zurückgeben, wenn der i-te kleinste Haufen und der j-te kleinste Haufen dieselbe aktuelle Höhe haben (das heisst, wenn $H_i = H_j$), und `false` andernfalls.
- Diese Funktion kann höchstens Q_{compare} Mal aufgerufen werden.

Beispielgrader

Das Verzeichnis der Aufgabe enthält eine vereinfachte Version des Jury-Graders, die Du verwenden kannst, um Deine Lösung lokal zu testen. Der vereinfachte Grader liest die Eingangsdaten aus der Datei `stdin`, ruft die Funktionen auf, die Du implementieren musst, und schreibt schliesslich die Ausgabe in die Datei `stdout`.

Die Eingabe besteht aus zwei Zeilen, die Folgendes enthalten:

- Zeile 1: die Ganzzahlen N, Q_{add} und Q_{compare}, durch Leerzeichen getrennt.
- Zeile 2: die Ganzzahlen H_i, durch Leerzeichen getrennt.

Wenn Dein Programm als **Accepted** bewertet wird, gibt der Beispielgrader `Accepted: add=U, compare=V` aus, wobei U und V die Anzahl der Aufrufe von `add` und `compare` sind.

Wenn Dein Programm als **Wrong Answer** bewertet wird, gibt der Beispielgrader `Wrong Answer: MSG` aus, wobei `MSG` einer der folgenden ist:

Message	Meaning
`too many calls to add`	Du hast `add` mehr als Q_{add} Mal aufgerufen.
`X out of range`	Die Ganzzahl X, die an `add` übergeben wurde, liegt nicht zwischen 0 und 10^{12} inklusive.
`index in S out of range`	Ein Element des Vektors S, der an `add` übergeben wurde, liegt nicht zwischen 0 und $N-1$ inklusive.
`indices in S not distinct`	Es gibt zwei gleiche Elemente im Vektor S, der an `add` übergeben wurde.
`too many calls to compare`	Du hast `compare` mehr als Q_{compare} Mal aufgerufen.
`i out of range`	Die Ganzzahl i, die an `compare` übergeben wurde, liegt nicht zwischen 0 und $N-1$ inklusive.
`j out of range`	Die Ganzzahl j, die an `compare` übergeben wurde, liegt nicht zwischen 0 und $N-1$ inklusive.
`heights are not equal`	Nach dem Aufruf von `make_all_equal` gibt es zwei Haufen mit unterschiedlichen Höhen.

Einschränkungen

- $2 \leq N \leq 2048$, und N **is a power of two**.
- Die initialen Höhen erfüllen $1 \leq H_0 \leq \cdots \leq H_{N-1} \leq 1\,000\,000$.

Punktevergabe

Dein Programm wird anhand einer Reihe von Testfällen getestet, die nach Teilaufgaben gruppiert sind. Um die Punktzahl zu erhalten, die einer Teilaufgabe zugeordnet ist, musst Du alle darin enthaltenen Testfälle korrekt lösen.

- **Teilaufgabe 1** [**0 Punkte**]: Beispiel Testfall.
- **Teilaufgabe 2** [**5 Punkte**]: $N = 2$, $H_i \leq 4$, $Q_{\text{add}} \geq 3000$ und $Q_{\text{compare}} \geq 4000$.
- **Teilaufgabe 3** [**16 Punkte**]: $N = 2$, $H_i \leq 1\,000\,000$, $Q_{\text{add}} \geq 22$ und $Q_{\text{compare}} \geq 1$.
- **Teilaufgabe 4** [**15 Punkte**]: $N = 256$, $H_i \leq 10$, $Q_{\text{add}} \geq 3000$ und $Q_{\text{compare}} \geq 255$.
- **Teilaufgabe 5** [**18 Punkte**]: $N = 4$, $H_i \leq 1\,000\,000$, $Q_{\text{add}} \geq 45$ und $Q_{\text{compare}} \geq 3$.
- **Teilaufgabe 6** [**22 Punkte**]: $N = 2048$, $H_i \leq 1\,000\,000$, $Q_{\text{add}} \geq 298$ und $Q_{\text{compare}} \geq 4000$ **und zusätzlich gibt es maximal zwei unterschiedliche Höhen**. In anderen Worten, für alle i ist entweder $H_0 = H_i$ oder $H_i = H_{N-1}$.
- **Teilaufgabe 7** [**24 Punkte**]: $N = 2048$, $H_i \leq 1\,000\,000$, $Q_{\text{add}} \geq 298$ und $Q_{\text{compare}} \geq 2047$.

Beachte, dass keine der Teilaufgaben alle Testfälle enthält.

Beispiele

Input	**Beispielkommunikation**			
	Aufrufe	Rückgabe	Höhen	Erklärung
`4 45 3` `1 4 5 5`	`compare(1, 2)`	`false`		Die Lösung fragt, ob $H_1 = H_2$. Da $H_1 = 4$ und $H_2 = 5$ ist, ist dies falsch.
	`compare(2, 3)`	`true`		Die Lösung fragt, ob $H_2 = H_3$. Da $H_2 = H_3 = 5$ ist, ist dies richtig.
	`add({0, 1}, 2)`		$3, 5, 5, 6$	Nach dem Hinzufügen der Steine sind die neuen Höhen $[\mathbf{3}, \mathbf{6}, 5, 5]$. Nach dem Neuordnen werden sie zu $H = [3, 5, 5, 6]$.
	`compare(1, 2)`	`true`		Die Lösung fragt, ob $H_1 = H_2$. Da $H_1 = H_2 = 5$ ist, ist dies richtig.
	`add({0, 3}, 3)`		$5, 5, 6, 9$	Nach dem Hinzufügen der Steine sind die neuen Höhen $[\mathbf{6}, 5, 5, \mathbf{9}]$. Nach dem Neuordnen werden sie zu $H = [5, 5, 6, 9]$.
	`add({0, 1}, 4)`		$6, 9, 9, 9$	Nach dem Hinzufügen der Steine sind die neuen Höhen $[\mathbf{9}, \mathbf{9}, 6, 9]$. Nach dem Neuordnen werden sie zu $H = [6, 9, 9, 9]$.
	`add({0}, 3)`		$9, 9, 9, 9$	Nach dem Hinzufügen der Steine sind die neuen Höhen $[\mathbf{9}, 9, 9, 9]$. Nach dem Neuordnen werden sie zu $H = [9, 9, 9, 9]$.