Distributing Candies

Aunty Khong is preparing \( n \) boxes of candies for students from a nearby school. The boxes are numbered from 0 to \( n - 1 \) and are initially empty. Box \( i \) (\( 0 \leq i \leq n - 1 \)) has a capacity of \( c[i] \) candies.

Aunty Khong spends \( q \) days preparing the boxes. On day \( j \) (\( 0 \leq j \leq q - 1 \)), she performs an action specified by three integers \( l[j] \), \( r[j] \) and \( v[j] \) where \( 0 \leq l[j] \leq r[j] \leq n - 1 \) and \( v[j] \neq 0 \). For each box \( k \) satisfying \( l[j] \leq k \leq r[j] \):

- If \( v[j] > 0 \), Aunty Khong adds candies to box \( k \), one by one, until she has added exactly \( v[j] \) candies or the box becomes full. In other words, if the box had \( p \) candies before the action, it will have \( \min(c[k], p + v[j]) \) candies after the action.
- If \( v[j] < 0 \), Aunty Khong removes candies from box \( k \), one by one, until she has removed exactly \( -v[j] \) candies or the box becomes empty. In other words, if the box had \( p \) candies before the action, it will have \( \max(0, p + v[j]) \) candies after the action.

Your task is to determine the number of candies in each box after the \( q \) days.

Implementation Details

You should implement the following procedure:

\[
\text{int[]} \ \text{distribute_candies(int[]} \ [c, \ [l, \ \text{int[]} \ [r, \ \text{int[]} \ [v])}
\]

- \( c \): an array of length \( n \). For \( 0 \leq i \leq n - 1 \), \( c[i] \) denotes the capacity of box \( i \).
- \( l \), \( r \) and \( v \): three arrays of length \( q \). On day \( j \), for \( 0 \leq j \leq q - 1 \), Aunty Khong performs an action specified by integers \( l[j] \), \( r[j] \) and \( v[j] \), as described above.
- This procedure should return an array of length \( n \). Denote the array by \( s \). For \( 0 \leq i \leq n - 1 \), \( s[i] \) should be the number of candies in box \( i \) after the \( q \) days.

Examples

Example 1

Consider the following call:

\[
\text{distribute_candies([10, 15, 13], [0, 0], [2, 1], [20, -11])}
\]
This means that box 0 has a capacity of 10 candies, box 1 has a capacity of 15 candies, and box 2 has a capacity of 13 candies.

At the end of day 0, box 0 has \( \min(c[0], 0 + v[0]) = 10 \) candies, box 1 has \( \min(c[1], 0 + v[0]) = 15 \) candies and box 2 has \( \min(c[2], 0 + v[0]) = 13 \) candies.

At the end of day 1, box 0 has \( \max(0, 10 + v[1]) = 0 \) candies, box 1 has \( \max(0, 15 + v[1]) = 4 \) candies. Since \( 2 > r[1] \), there is no change in the number of candies in box 2. The number of candies at the end of each day are summarized below:

<table>
<thead>
<tr>
<th>Day</th>
<th>Box 0</th>
<th>Box 1</th>
<th>Box 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>15</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>13</td>
</tr>
</tbody>
</table>

As such, the procedure should return \([0, 4, 13]\).

**Constraints**

- \( 1 \leq n \leq 200000 \)
- \( 1 \leq q \leq 200000 \)
- \( 1 \leq c[i] \leq 10^9 \) (for all \( 0 \leq i \leq n - 1 \))
- \( 0 \leq l[j] \leq r[j] \leq n - 1 \) (for all \( 0 \leq j \leq q - 1 \))
- \( -10^9 \leq v[j] \leq 10^9, v[j] \neq 0 \) (for all \( 0 \leq j \leq q - 1 \))

**Subtasks**

1. (3 points) \( n, q \leq 2000 \)
2. (8 points) \( v[j] > 0 \) (for all \( 0 \leq j \leq q - 1 \))
3. (27 points) \( c[0] = c[1] = \ldots = c[n - 1] \)
4. (29 points) \( l[j] = 0 \) and \( r[j] = n - 1 \) (for all \( 0 \leq j \leq q - 1 \))
5. (33 points) No additional constraints.

**Sample Grader**

The sample grader reads in the input in the following format:

- line 1: \( n \)
- line 2: \( c[0] \ c[1] \ldots \ c[n - 1] \)
- line 3: \( q \)
- line \( 4 + j \) (\( 0 \leq j \leq q - 1 \)): \( l[j] \ r[j] \ v[j] \)

The sample grader prints your answers in the following format:

- line 1: \( s[0] \ s[1] \ldots \ s[n - 1] \)