Profit Suppression

Congratulations! You're a CEO. Not only that, you're the CEO of a massive logistics empire where efficiency is everything and worker satisfaction is a concept *for sure*. Every day, your employees generate revenue across different departments, and it's your job to manage their earnings. Unfortunately, your superyacht won't pay for itself, and so more budget cuts must be made. Each worker has a maximum earning cap, and under **no** circumstances are they allowed to exceed it. Every raise comes out of your pocket after all. You've decided to use your Advanced Algorithms knowledge to efficiently process company-wide pay adjustments and reporting queries.

Problem Description

You are given:

- An array of worker earnings
- An array of maximum earning capacities

You must process a sequence of operations:

1. **Update:** Increase earnings across a range of workers
2. **Query:** Compute the total earnings in a range

However, any update must respect the earning caps.

For an operation:

`U l r x`

you attempt to add x to every worker in the range $[l, r]$.

But if any worker would exceed their capacity, you must instead compute:

$$\min_{i \in [l,r]} (c[i] - a[i])$$

and only add:

$$\min(x, \text{that minimum})$$

to every worker in the range.

For a query:

`Q l r`

you must output:

$$\sum_{i=l}^{r} a[i]$$

Input

The first line contains an integer N ($1 \leq N \leq 2 \cdot 10^5$), the number of workers.
The second line contains N integers representing the initial earnings.
The third line contains N integers representing the earning capacities.
The fourth line contains an integer Q ($1 \leq Q \leq 2 \cdot 10^5$), the number of operations.
The next Q lines each contain one operation of the form:

- `U l r x`
- `Q l r`

All indices are 0-based and inclusive.

Output

For each query operation, output a single integer on its own line.

Example

Sample Input

```
5
1 2 3 4 5
5 5 8 10 10
6
Q 0 4
U 1 3 2
Q 0 4
U 0 2 10
Q 0 2
Q 2 4
```

Sample Output

```
15
21
15
21
```

Notes

A naive approach will be too slow for large inputs. To solve this efficiently, you should use a Segment Tree with Lazy Propagation that maintains:

- Range sums
- Minimum remaining capacity in each segment
- Pending updates (lazy values)

This allows both updates and queries to run in $O(\log N)$ time.

Remember: your workers might try to earn more, but your system must ensure they **never** succeed.