

# Segment Tree Lazy

Enzo Vivallo

enzovivallo@estudiante.uc.cl



Se te entrega una lista de enteros  $a$  de tamaño  $n$  ( $1 \leq n \leq 10^5$ ), y cada elemento cumple  $1 \leq a_i \leq 10^5$ . Nos piden responder  $q$  ( $1 \leq q \leq 10^5$ ) consultas del tipo:

- ¿Cuál es el mínimo en el rango  $[left, right]$ ?
- Suma en todos los elementos del rango  $[left, right]$  el valor  $x$



# Solución con Segment Tree

Hacemos un Segment Tree y por cada query de actualización iteramos sobre el rango  $[left, right]$ , por cada índice  $i$  del rango actualizamos esa posición sumándole  $x$ . En el peor caso el rango es de tamaño  $n$ , entonces la complejidad es  $O(n \log n)$  por actualización.

El Segment Tree Lazy es una variación que permite reducir la complejidad de las actualizaciones a  $O(\log n)$ .



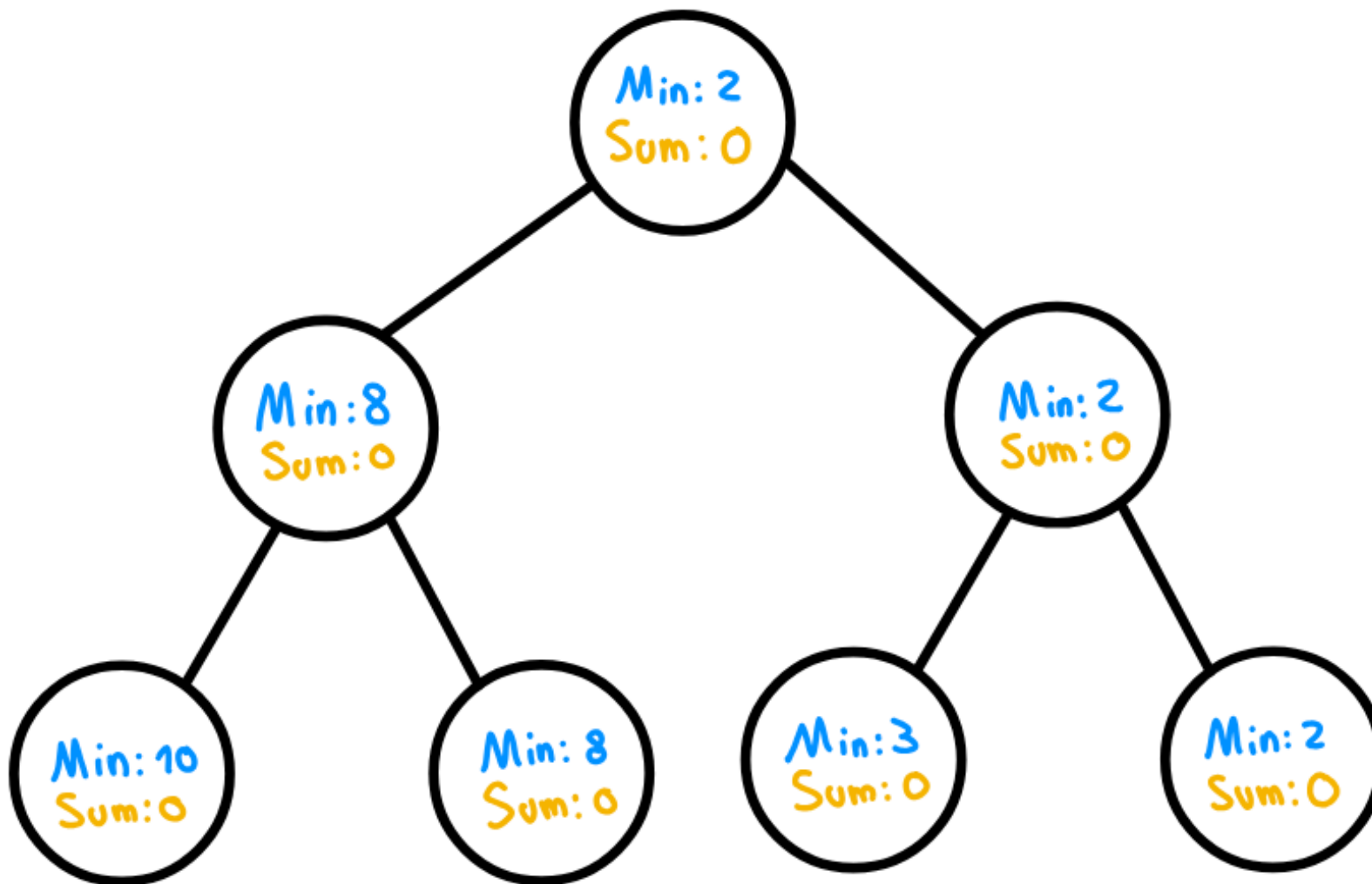
# Segment Tree Lazy

La idea de esta modificación es posponer las actualizaciones hasta que sea necesario. En vez de actualizar inmediatamente los nodos del **Segment Tree**, guardamos la información de las actualizaciones en un arreglo adicional llamado *lazy*, que nos indica el valor de la actualización pendiente para cada nodo, cuando se accede al nodo en el **Segment Tree** se aplica la actualización y se propaga a los hijos. También marcamos los hijos como pendientes de actualización con otro arreglo adicional.



# Actualización en rango

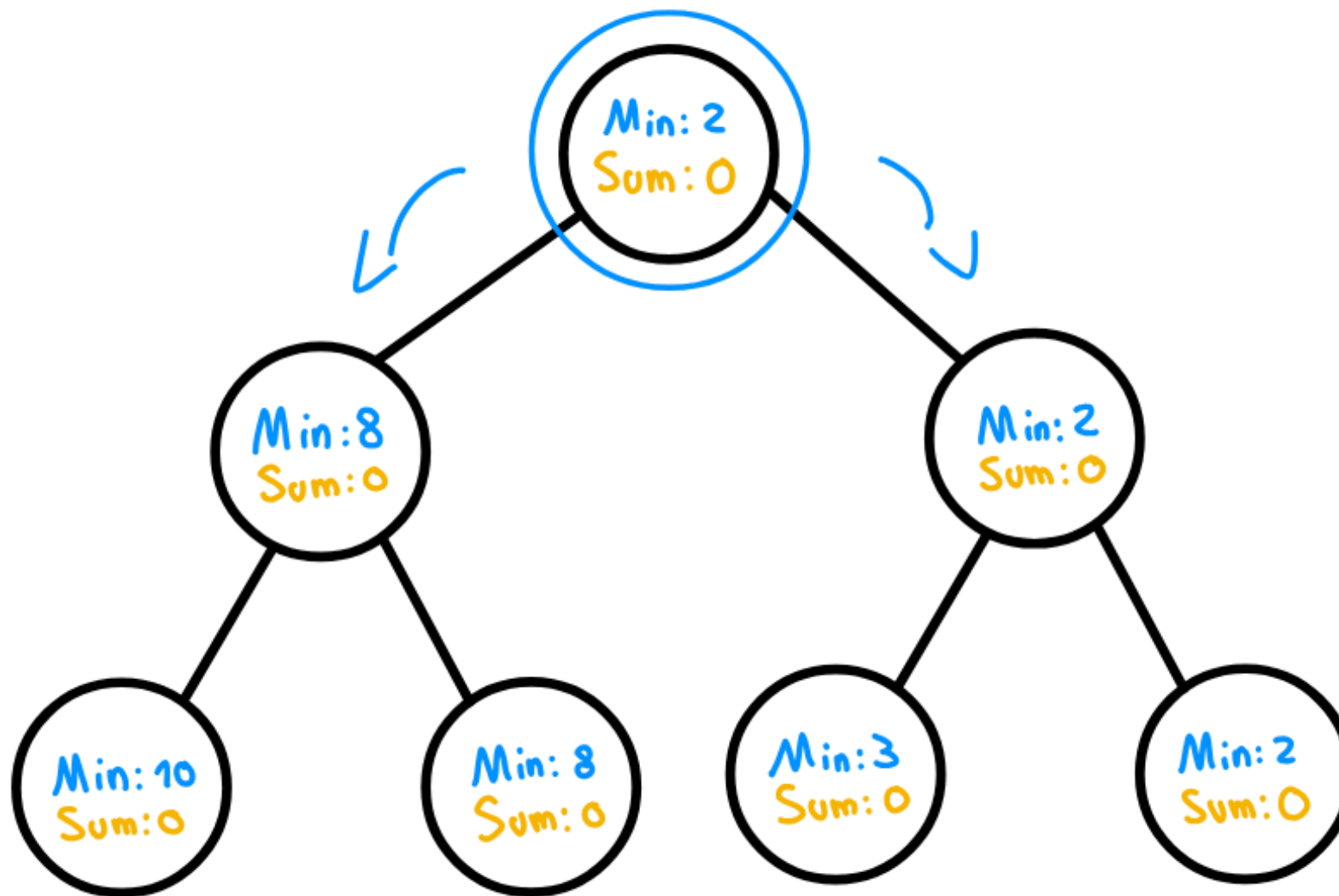
Para  $a = [10, 8, 3, 2]$  y suma de 2 en el rango  $[1, 3]$  para consultas de mínimo:





# Actualización en rango

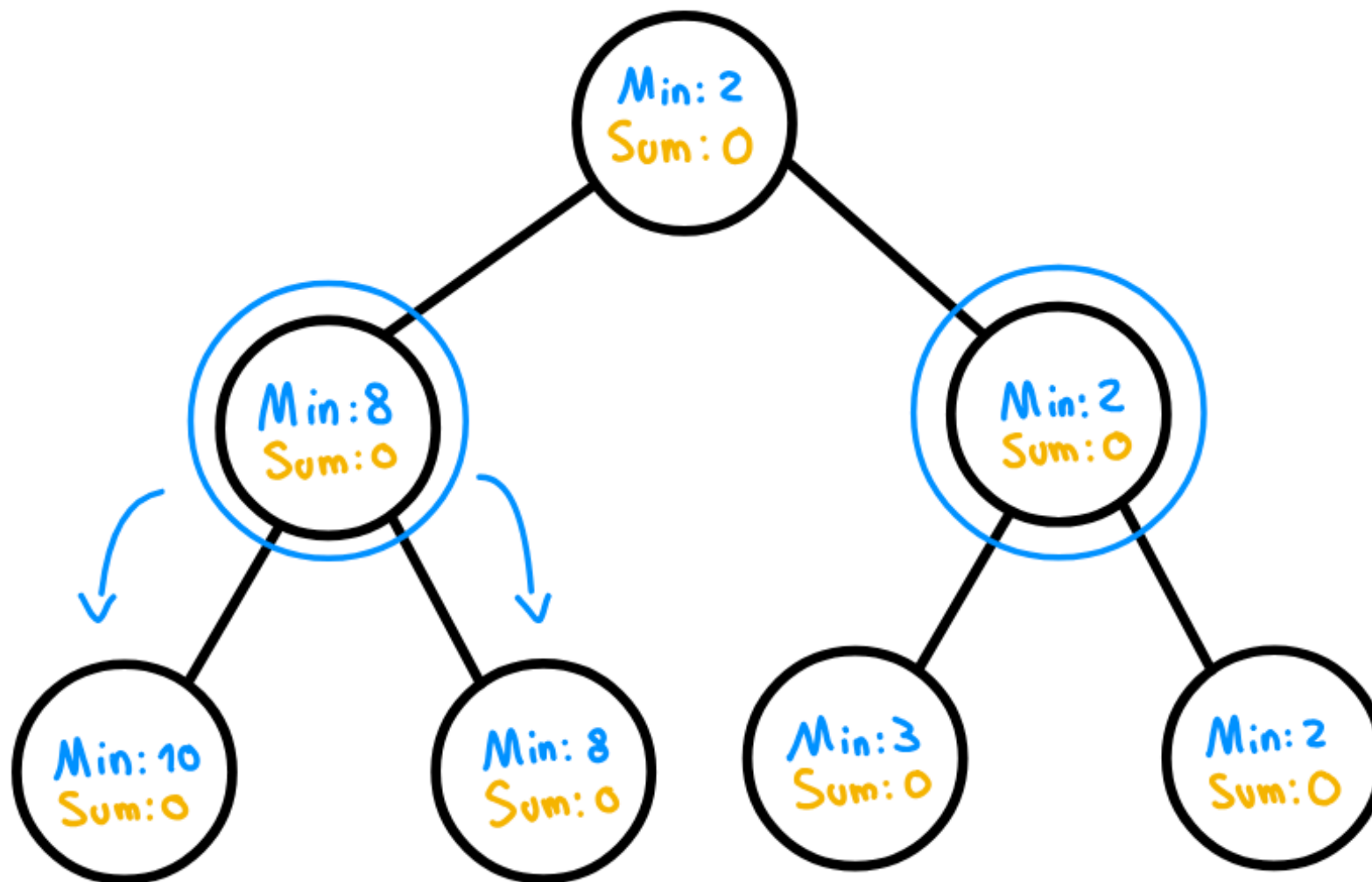
Para  $a = [10, 8, 3, 2]$  y suma de 2 en el rango  $[1, 3]$  para consultas de mínimo:





# Actualización en rango

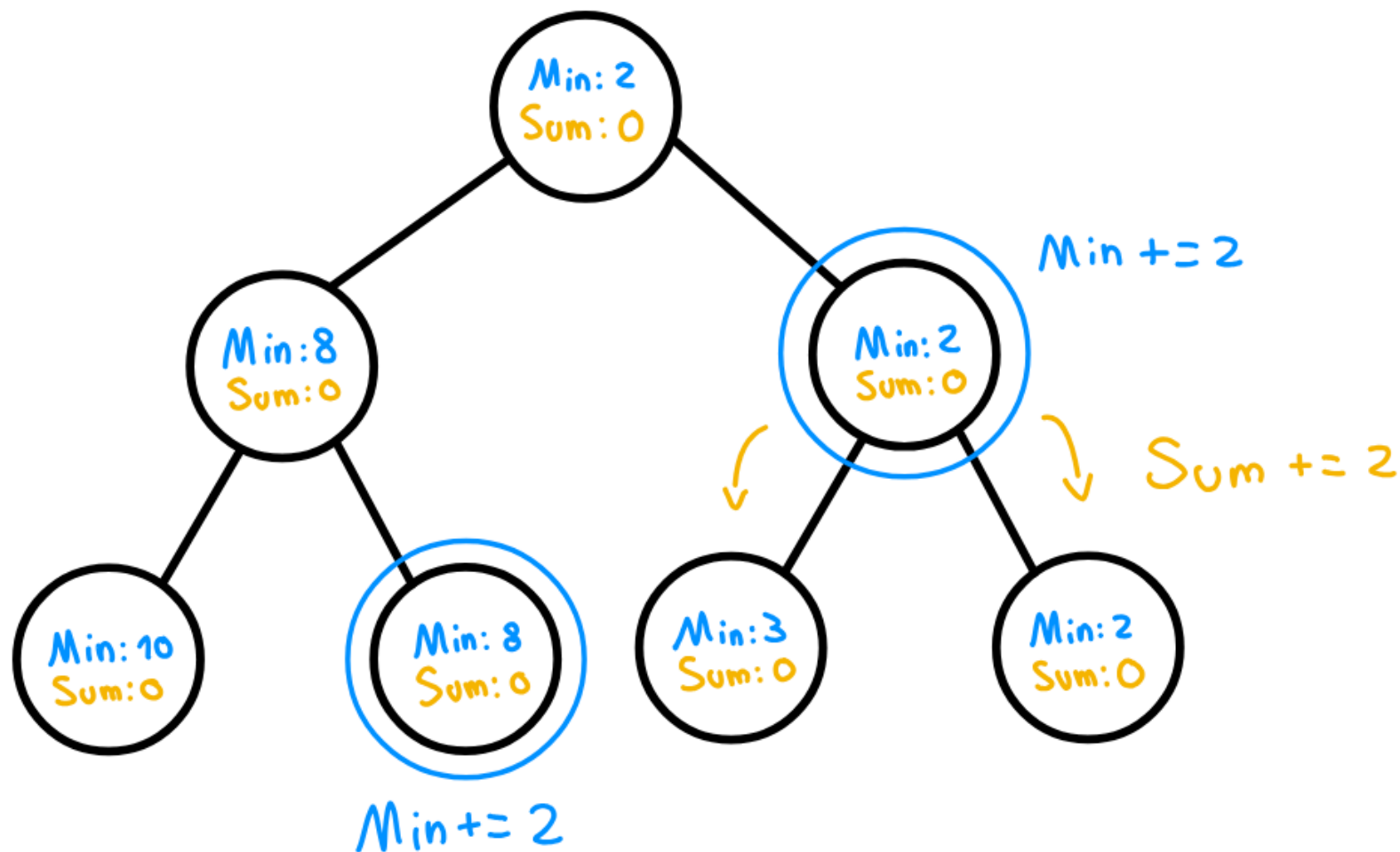
Para  $a = [10, 8, 3, 2]$  y suma de 2 en el rango  $[1, 3]$  para consultas de mínimo:





# Actualización en rango

Para  $a = [10, 8, 3, 2]$  y suma de 2 en el rango  $[1, 3]$  para consultas de mínimo:

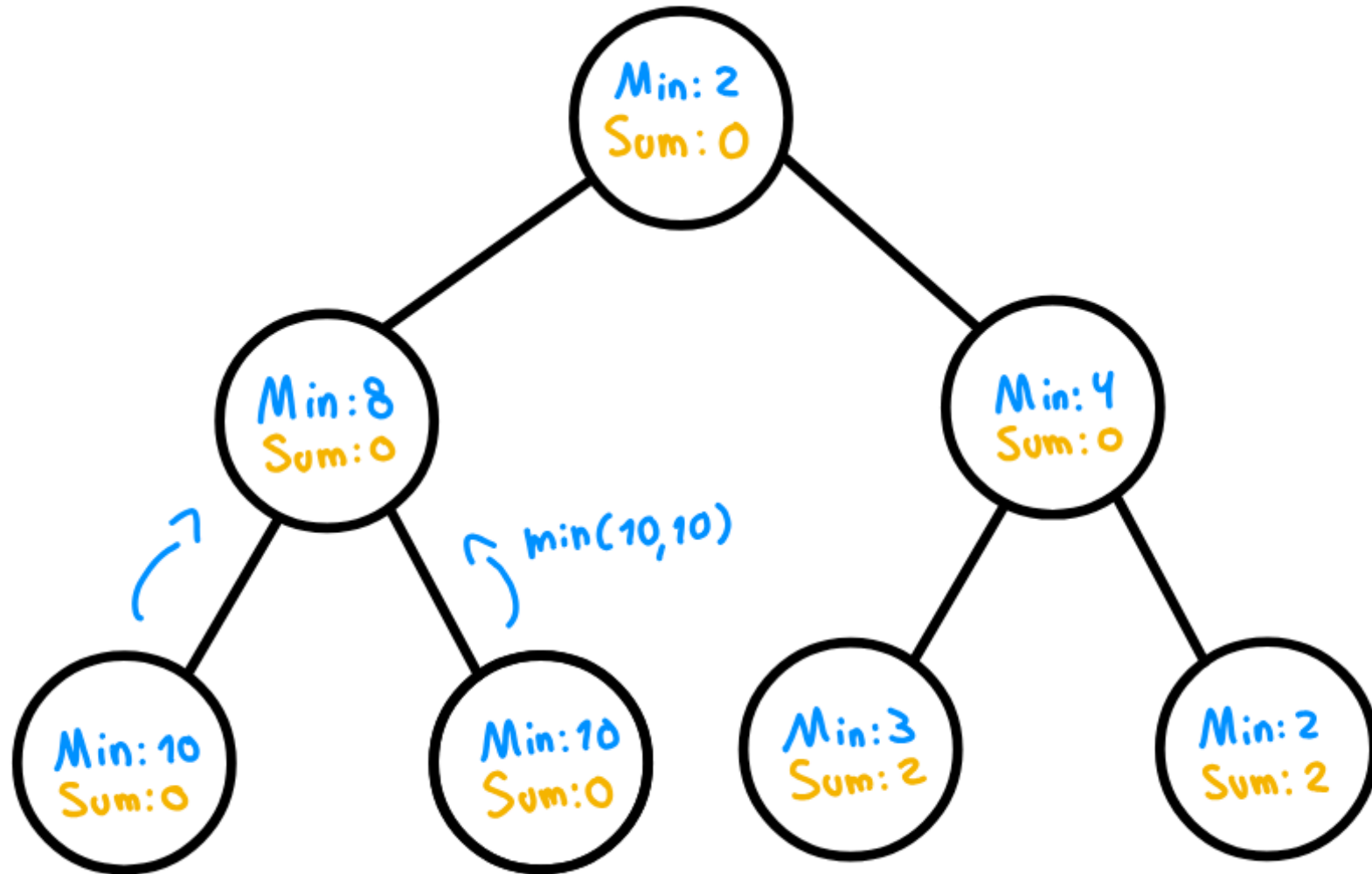






# Actualización en rango

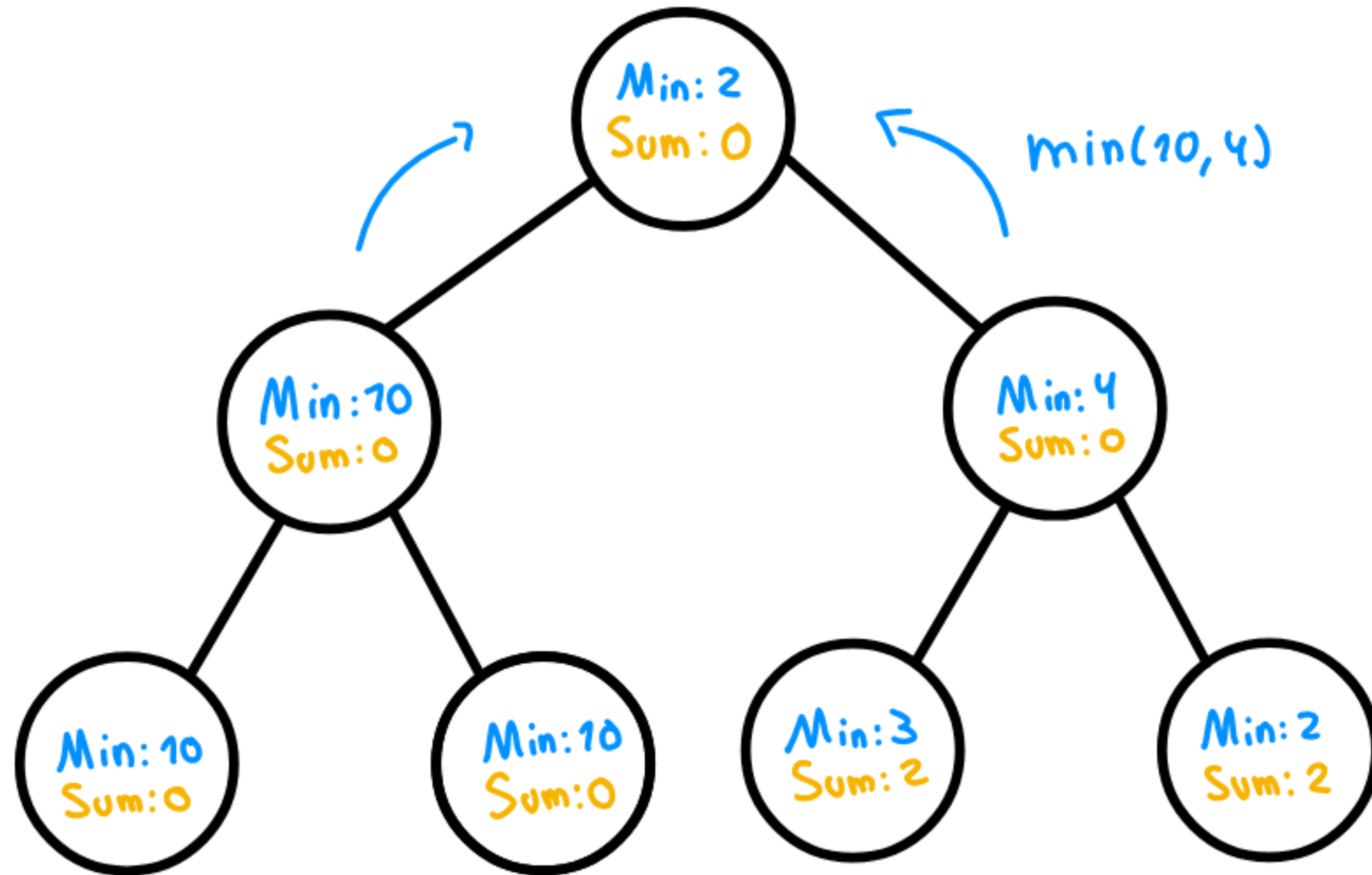
Para  $a = [10, 8, 3, 2]$  y suma de 2 en el rango  $[1, 3]$  para consultas de mínimo:





# Actualización en rango

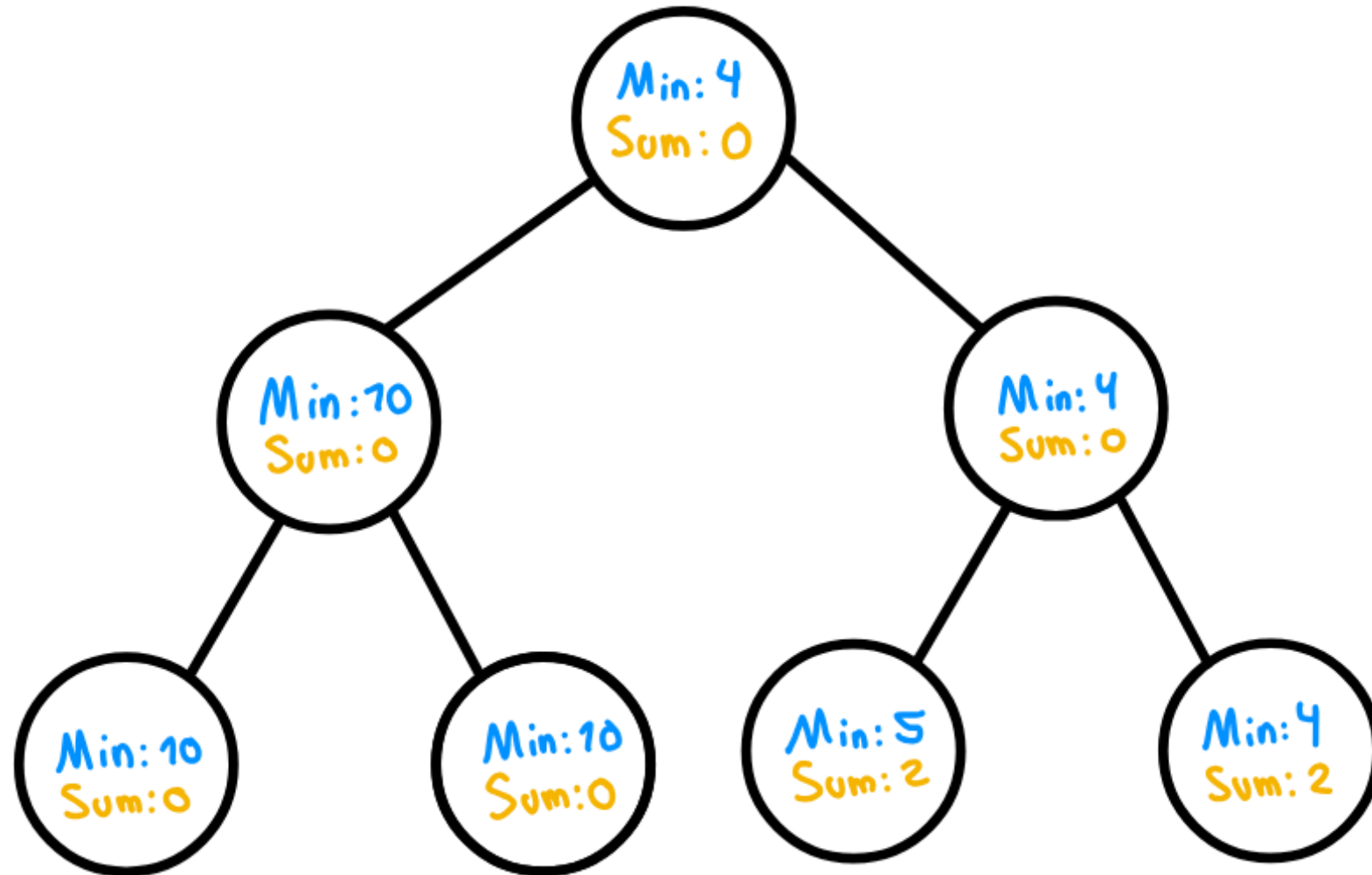
Para  $a = [10, 8, 3, 2]$  y suma de 2 en el rango  $[1, 3]$  para consultas de mínimo:





# Actualización en rango

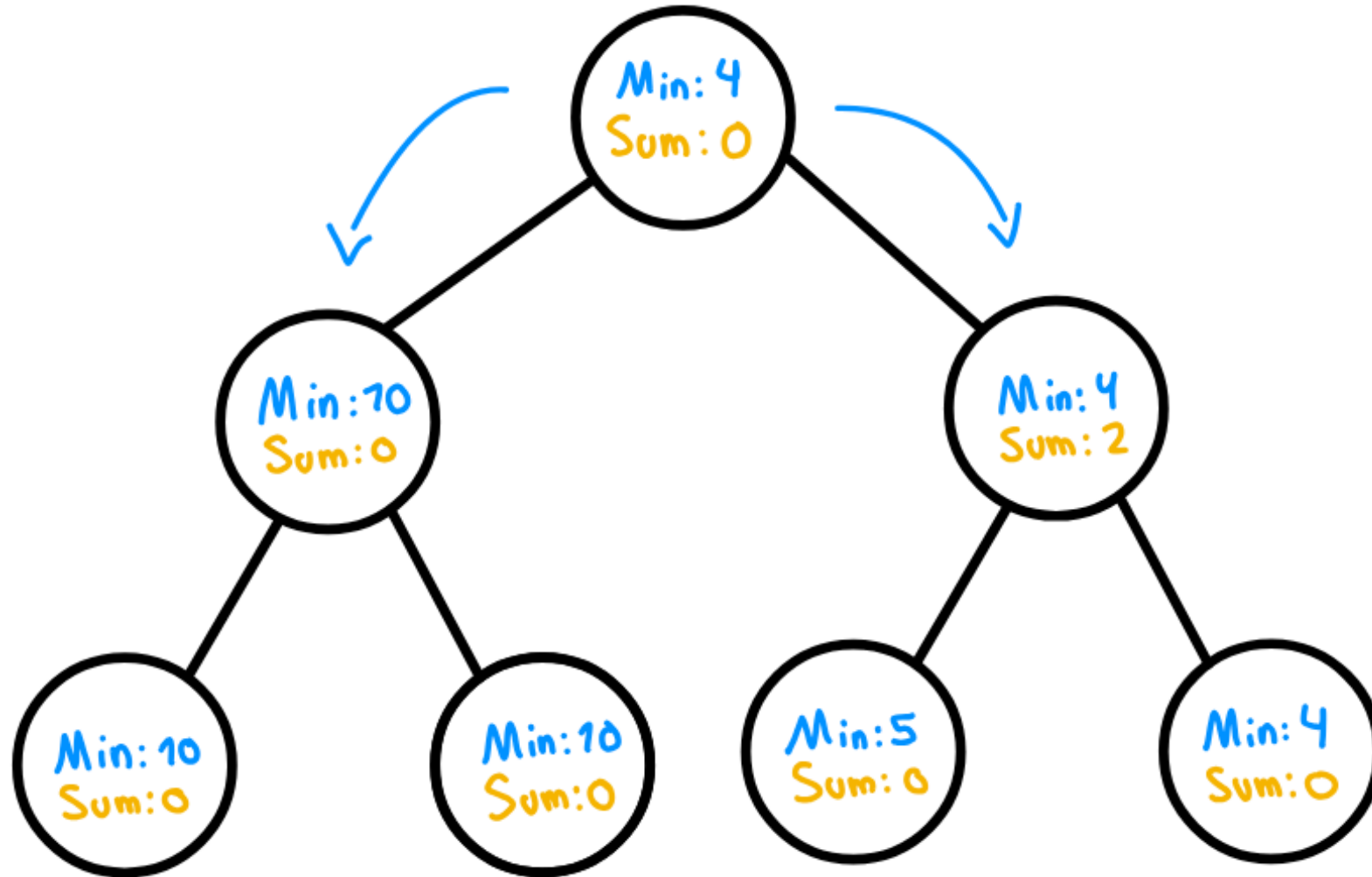
Para  $a = [10, 8, 3, 2]$  y suma de 2 en el rango  $[1, 3]$  para consultas de mínimo:





# Consulta en rango

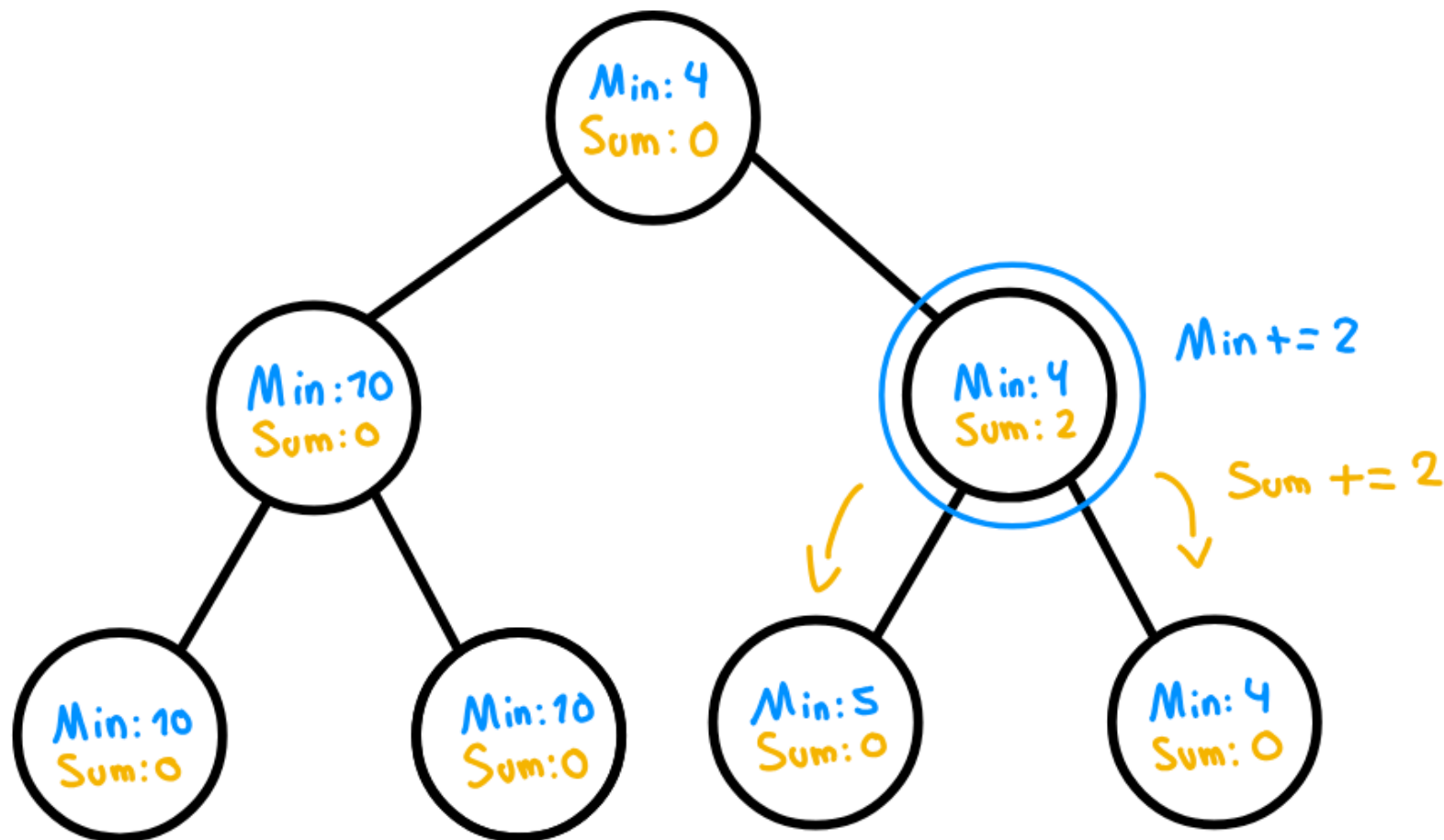
Para  $a = [10, 10, 5, 4]$  y consulta de mínimo en el rango  $[2, 3]$ :





# Consulta en rango

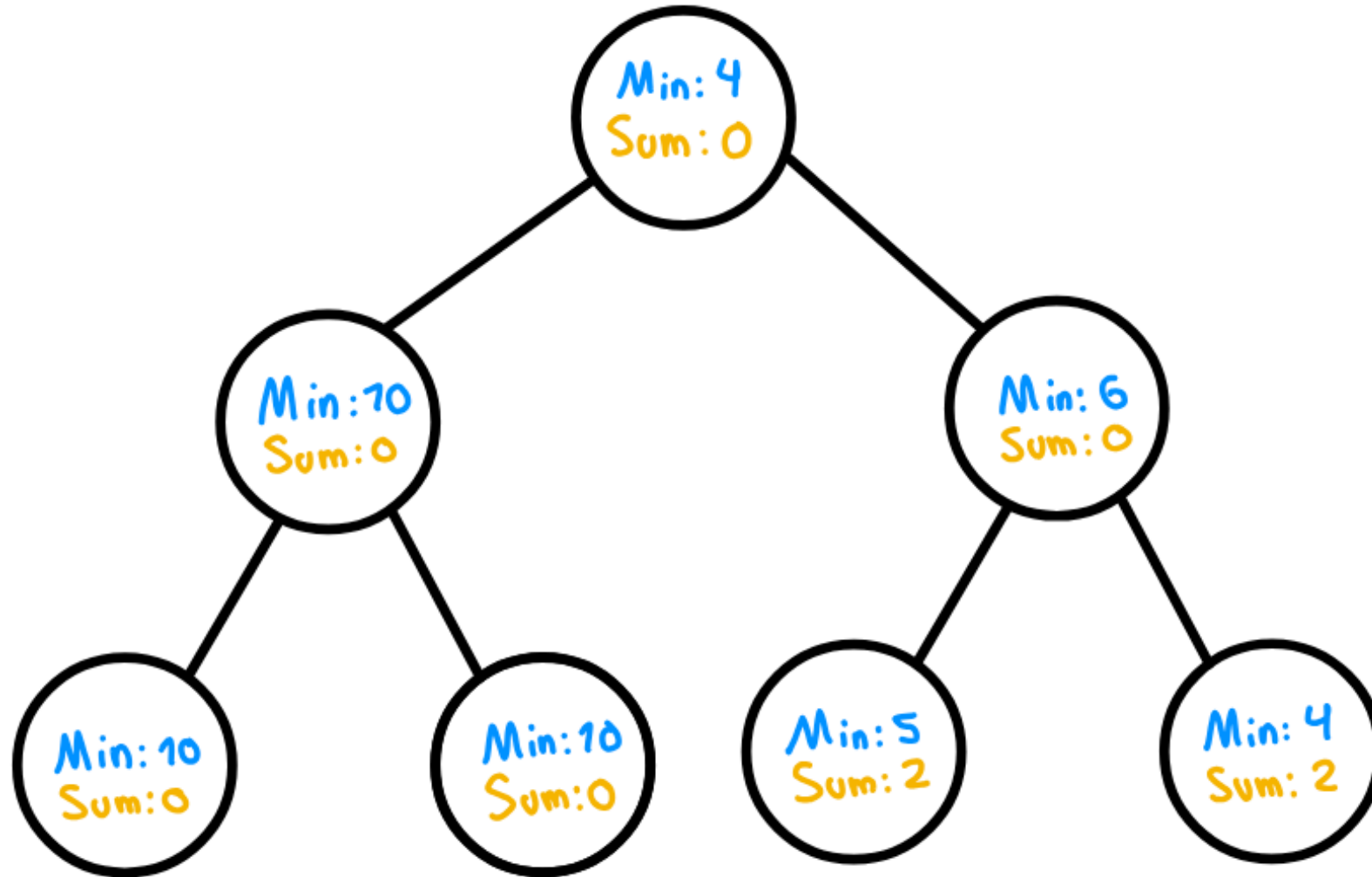
Para  $a = [10, 10, 5, 4]$  y consulta de mínimo en el rango  $[2, 3]$ :





# Consulta en rango

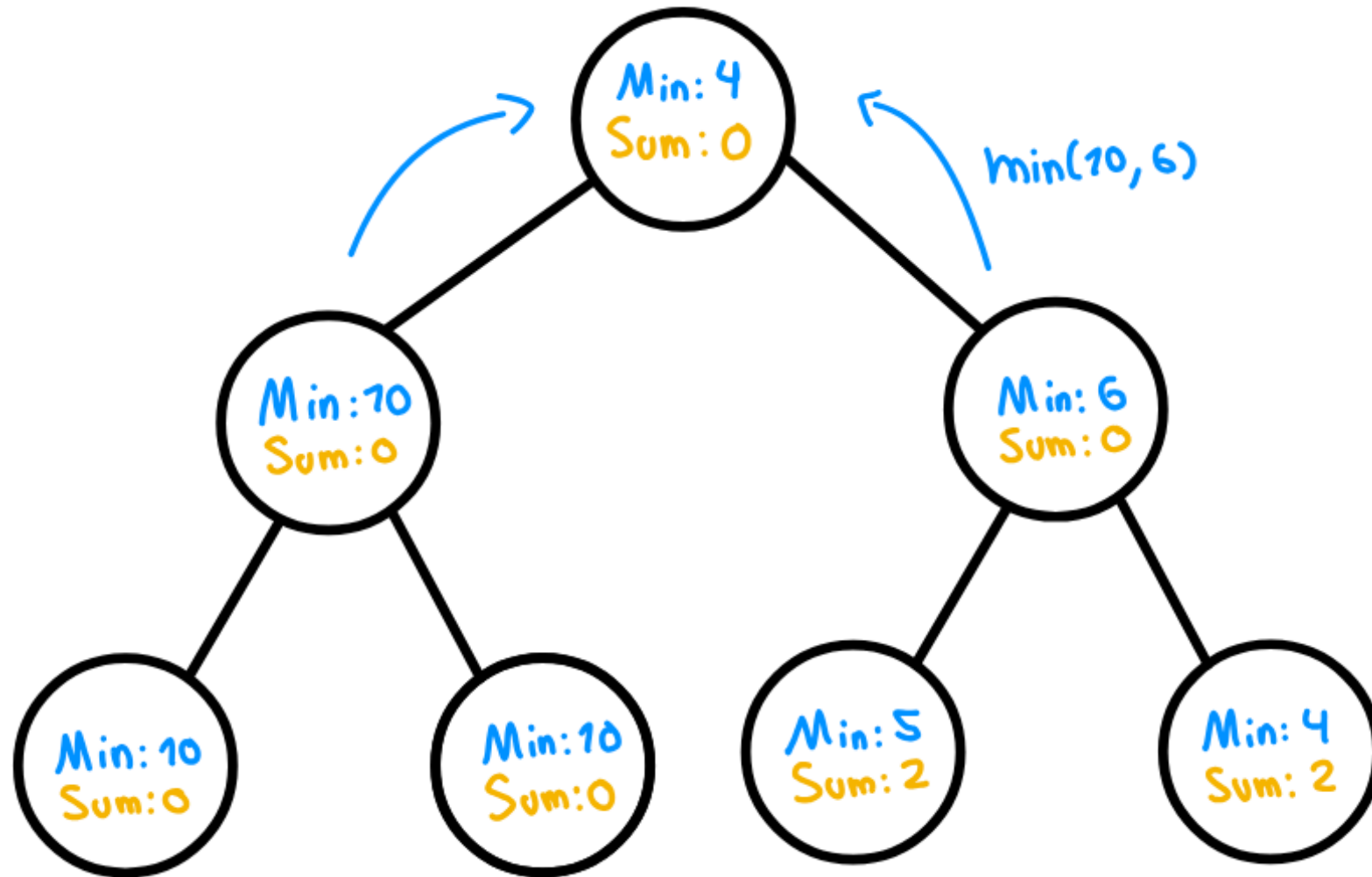
Para  $a = [10, 10, 5, 4]$  y consulta de mínimo en el rango  $[2, 3]$ :





# Consulta en rango

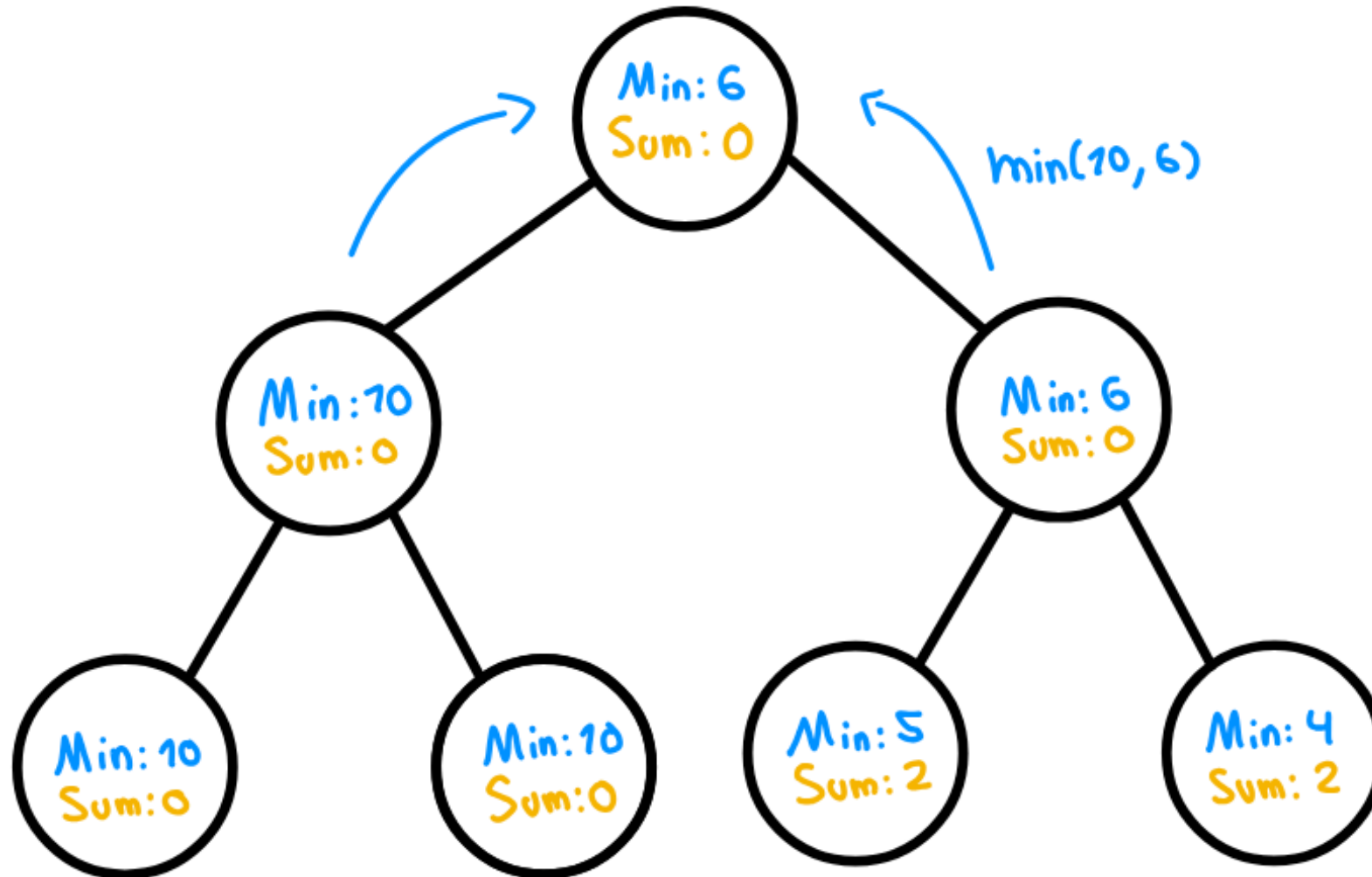
Para  $a = [10, 10, 5, 4]$  y consulta de mínimo en el rango  $[2, 3]$ :





# Consulta en rango

Para  $a = [10, 10, 5, 4]$  y consulta de mínimo en el rango  $[2, 3]$ :







# Cómo usarlo

`T1` es el tipo de dato del Segment Tree, `T2` es el tipo de dato de la actualización.

- `T1 merge(T1, T1)` : Es el mismo merge del Segment Tree normal.
- `void pushUpdate(T2 parent, T2 &child, int leftParent, int rightParent, int leftChild, int rightChild)` : Aplica la actualización del padre al hijo. Los parámetros `leftParent`, `rightParent`, `leftChild`, `rightChild` son los rangos de los nodos padre e hijo respectivamente.
- `void applyUpdate(T2 update, T1 &node, int left, int right)` : Aplica la actualización al nodo. Los parámetros `left` y `right` son el rango del nodo.



# Cómo usarlo

```
int merge(int a, int b) { return min(a, b); };

void pushUpdate(int parent, int &child, int leftParent, int rightParent,
               int leftChild, int rightChild) {
    child += parent;
};

void applyUpdate(int update, int &node, int, int) { node += update; }

int main() {
    vector<int> values = {10, 5, 2, 1, 5, 6};

    SegmentTree<int, merge, pushUpdate, applyUpdate> SegmentTreeLazy(values);

    int result = segmentTree.query(1, 4); // Mínimo en el rango [1, 4]

    segmentTree.update(1, 2, 3); // Suma 3 en el rango [1, 2]
}
```



# Circular RMQ

Tienes un arreglo circular de tamaño  $n$  ( $1 \leq n \leq 2 * 10^5$ ) de enteros entre 1 y  $10^6$ . Se te piden realizar hasta  $2 * 10^5$  operaciones de dos tipos:

Operaciones:

- $inc(left, right, v)$ :  
Aumenta en  $v$  ( $1 \leq v \leq 10^6$ ) todos los elementos del arreglo desde el índice  $left$  hasta  $right$  (ambos incluidos).
- $rmq(left, right)$ :  
Devuelve el valor mínimo en el segmento circular  $[left, right]$ .

El intervalo puede ser circular: si  $left > right$ , se considera que pasa por el final y sigue desde el principio.

 Link del problema: [codeforces.com/contest/52/problem/C](https://codeforces.com/contest/52/problem/C)

 Link del código solución: [miniurl.cl/oaj8s5](https://miniurl.cl/oaj8s5)



# XOR on Segment

Tienes un arreglo de  $n$  ( $1 \leq n \leq 10^5$ ) enteros en el rango  $[1, 10^6]$  y  $m$  ( $1 \leq m \leq 5 * 10^4$ ) operaciones. Cada operación puede ser de dos tipos:

- Tipo 1: Pedir la suma de los elementos entre las posiciones  $[left, right]$ .
- Tipo 2: Aplicar **XOR** con un valor  $x$  a todos los elementos entre  $[left, right]$ .

Debes procesar todas las operaciones en orden. Para cada consulta de tipo 1, debes imprimir el resultado de la suma correspondiente.

 Link del problema: [codeforces.com/contest/242/problem/E](https://codeforces.com/contest/242/problem/E)

 Link del código solución: [miniurl.cl/kl8zd3](https://miniurl.cl/kl8zd3)



# Lazy Segment Tree

Se te entrega un arreglo binario  $A$  de largo  $N$  ( $1 \leq N \leq 2 \times 10^5$ ), donde cada elemento  $A[i]$  es 0 o 1. Además, recibirás  $Q$  consultas ( $1 \leq Q \leq 2 \times 10^5$ ), cada una de las siguientes dos formas:

- $T_i = 1 \ L_i \ R_i$ : Invierte los bits entre las posiciones  $L_i$  y  $R_i$  (es decir, cambia 0 por 1 y 1 por 0).
- $T_i = 2 \ L_i \ R_i$ : Calcula cuántas inversiones hay en el subarreglo  $A[L_i \dots R_i]$ , donde una inversión es un par de índices  $i < j$  tal que  $A[i] > A[j]$ .

 Link del problema: [atcoder.jp/contests/practice2/tasks/practice2\\_1](https://atcoder.jp/contests/practice2/tasks/practice2_1)

 Link del código solución: [miniurl.cl/6awpyr](https://miniurl.cl/6awpyr)



# Sum of Squares with Segment Tree

Te dan un arreglo  $a$  de hasta  $10^5$  elementos y debes responder hasta  $10^5$  consultas sobre él usando un Segment Tree con Lazy Propagation. Las consultas pueden ser de tres tipos:

- $0 \ l \ r \ x$ : Asignar todos los elementos entre los índices  $l$  y  $r$  al valor  $x$  ( $-10^3 \leq x \leq 10^3$ ).
- $1 \ l \ r \ x$ : Incrementar todos los elementos entre los índices  $l$  y  $r$  en  $x$  ( $-10^3 \leq x \leq 10^3$ ).
- $2 \ l \ r$ : Calcular la suma de cuadrados de los elementos entre los índices  $l$  y  $r$ .

Todos los índices están en el rango  $1 \leq l \leq r \leq 10^5$ , y los valores iniciales del arreglo están acotados por  $|a_i| \leq 10^3$ .

# Sum of Squares with Segment Tree

- 🔗 Link del problema: [spoj.com/problems/SEGSQRSS/](https://spoj.com/problems/SEGSQRSS/)
- 🔗 Link del código solución: [miniurl.cl/luiuqe](https://miniurl.cl/luiuqe)