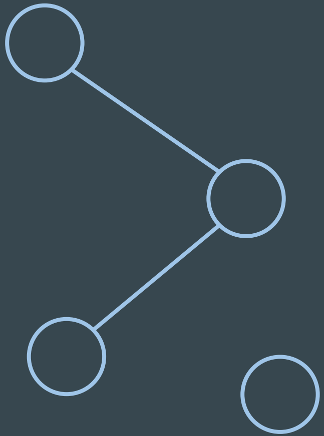
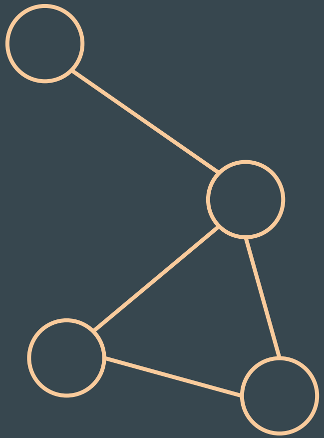
Segment Trees

Search trees on steroids

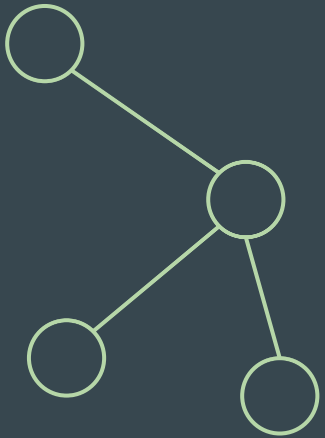
Reminder : what is a tree ?



acyclic graph



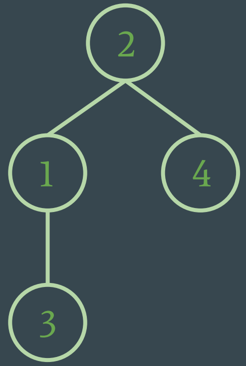
connected graph



tree = connected + acyclic

Properties of a segment tree?

- Similar to a binary tree:
  - It's a connected & acyclic graph
  - Finding a leaf in O(log(n))
  - Linear memory usage: 4n nodes needed for n elements
- But also different!
  - Find a subarray of element that verify of property in log(n)
  - Modification of one or multiple element in a subarray in log(n)



Sum of a subarray

We are given a list of elements and asked to answer k requests for the sum of a given subarray

→ “Moving sum” array:

1	5	7	6	7	9	3

We create a new array where each element is the sum of all the element before it.

0	1	6	13	19	26	35	38

Sum of a subarray

We are given a list of elements and asked to answer k requests for the sum of a given subarray

→ "Moving sum" array:

We create a new array where each element is the sum of all the element before it.

1	5	7	6	7	9	3

0	1	6	13	19	26	35	38

Sum of the subarray [1:3]
→ 19-1=18

Sum of the subarray [0:5]
→ 26-0=26

Sum of a subarray

We are given a list of elements and asked to answer k requests for the sum of a given subarray

But what if we also need to change a value in the array?

→ We have to recompute everything!

1	10	7	6	7	9	3

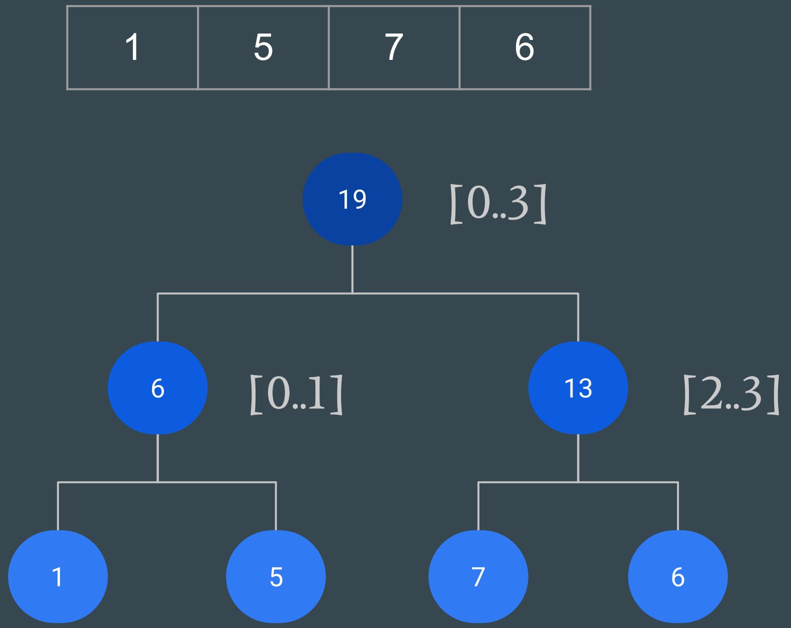
0	1	11	18	24	31	40	43

Sum of a subarray, with segment trees

We are given a list of elements and asked to answer k requests to either update the array or to compute the sum of a subarray

Leaves are the array, in the same order. Each parent is the sum of his two children.

How to compute the sum in the subarray?



Sum of a subarray, with segment trees

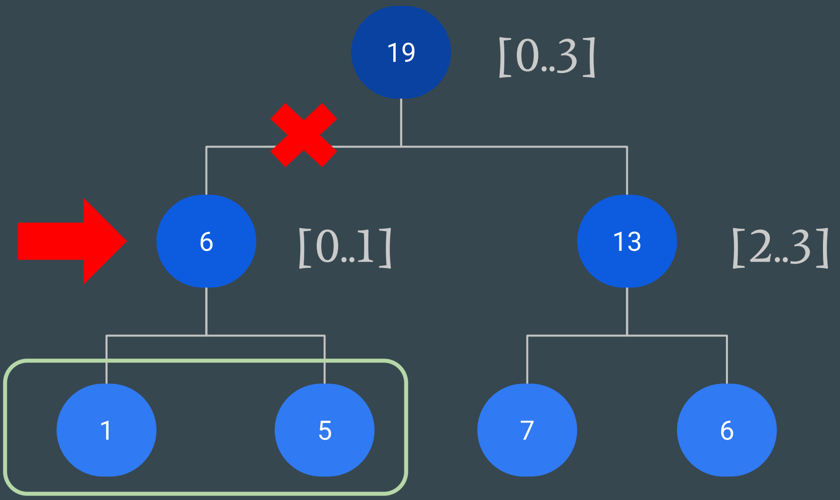
We are given a list of elements and asked to answer k requests to either update the array or to compute the sum of a subarray

Leaves are the array, in the same order.
Each parent is the sum of his two children.

How to compute the sum in the subarray?

Sum of the subarray [0:1]
→ 6

1	5	7	6



Sum of a subarray, with segment trees

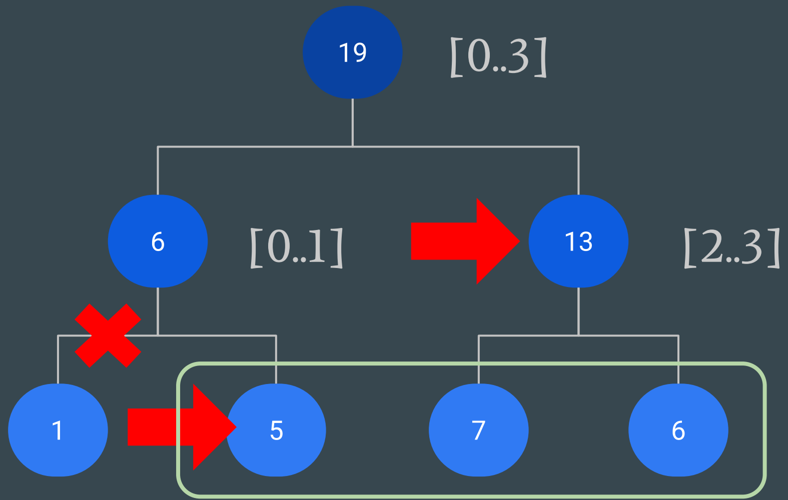
We are given a list of elements and asked to answer k requests to either update the array or to compute the sum of a subarray

Leaves are the array, in the same order. Each parent is the sum of his two children.

How to compute the sum in the subarray?

Sum of the subarray [1:3]
→ 5+13=18

1	5	7	6



Sum of a subarray, with segment trees

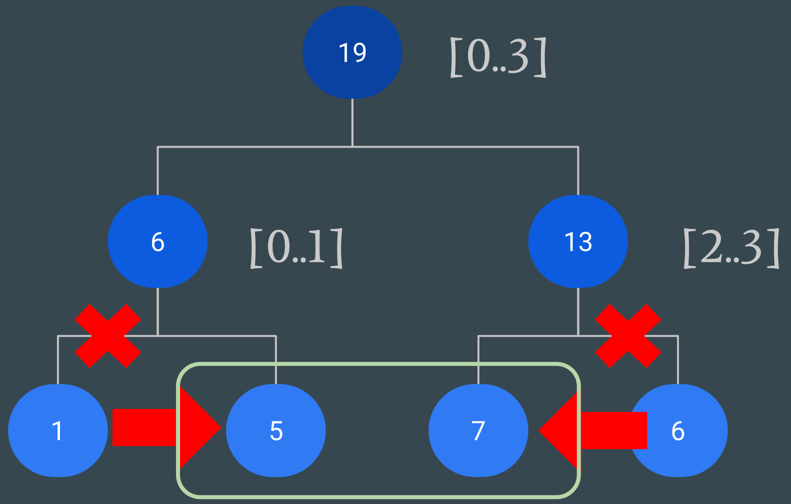
We are given a list of elements and asked to answer k requests to either update the array or to compute the sum of a subarray

Leaves are the array, in the same order. Each parent is the sum of his two children.

How to compute the sum in the subarray?

Sum of the subarray [1:2]
→ 5+7=12

1	5	7	6



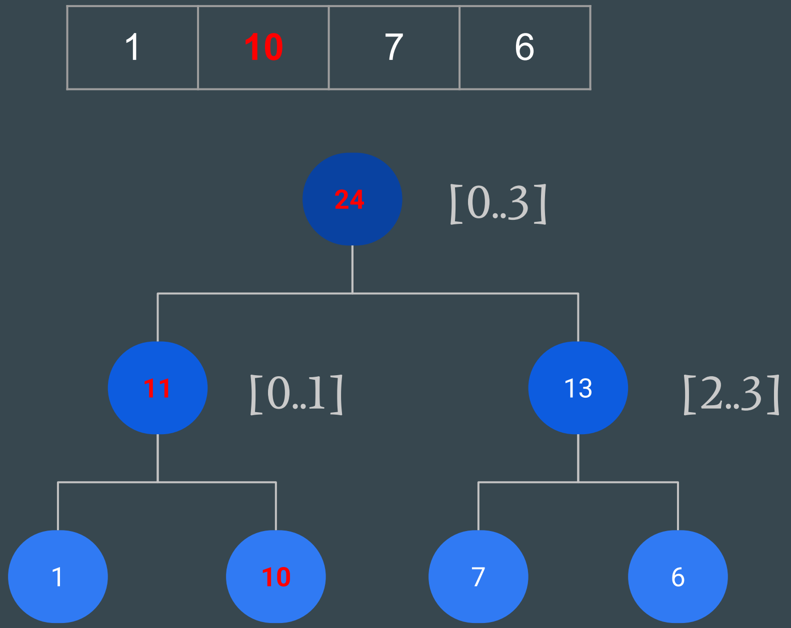
Sum of a subarray, with segment trees

We are given a list of elements and asked to answer k requests to either update the array or to compute the sum of a subarray

And what if we need to update a value?

→ Update the leaf and all of its parents

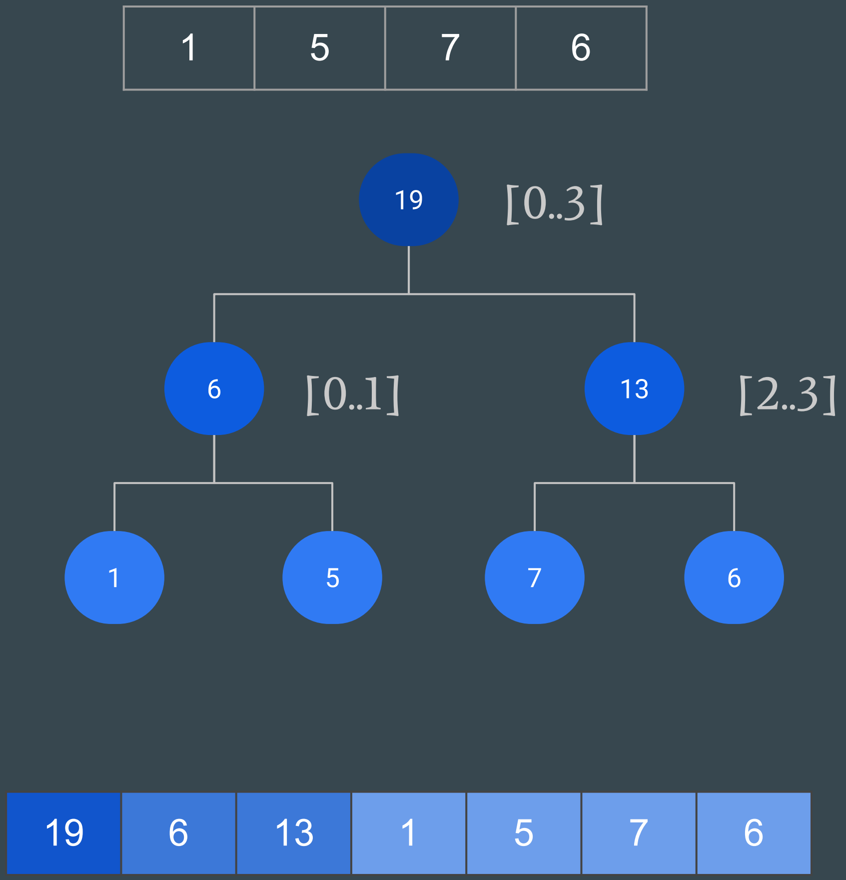
O(log(n)) operations, this is much better than the previous O(n)



Implementation of a segment tree

Similar to most binary trees.

For a node at the index i, its children are stored at the indexes 2*i and 2*i+1.



Implementation of a segment tree

Can be generalized to higher dimensions!

With a 2D matrix, we first construct the segment tree on each row (instead of each cell/value)

Then, we construct a segment tree for each node of the previous tree, on the values of each row

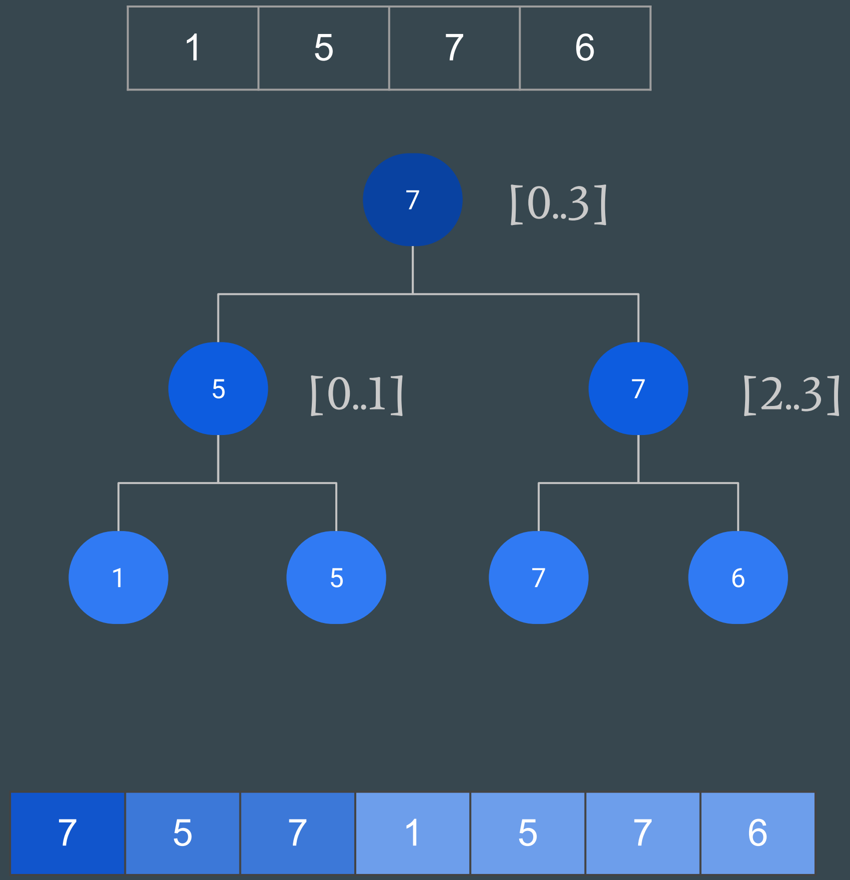
1	5	3	7
13	9	6	7
2	12	4	3
3	6	3	2

86	51	35	19	32	16	19
51	28	23	14	14	9	14
35	23	12	5	18	7	5
16	6	10	1	5	3	7
35	22	13	13	9	6	7
21	14	7	2	12	4	3
14	9	5	3	6	3	2

Generalization

The heuristic can be adapted to solve a wide range of problems.

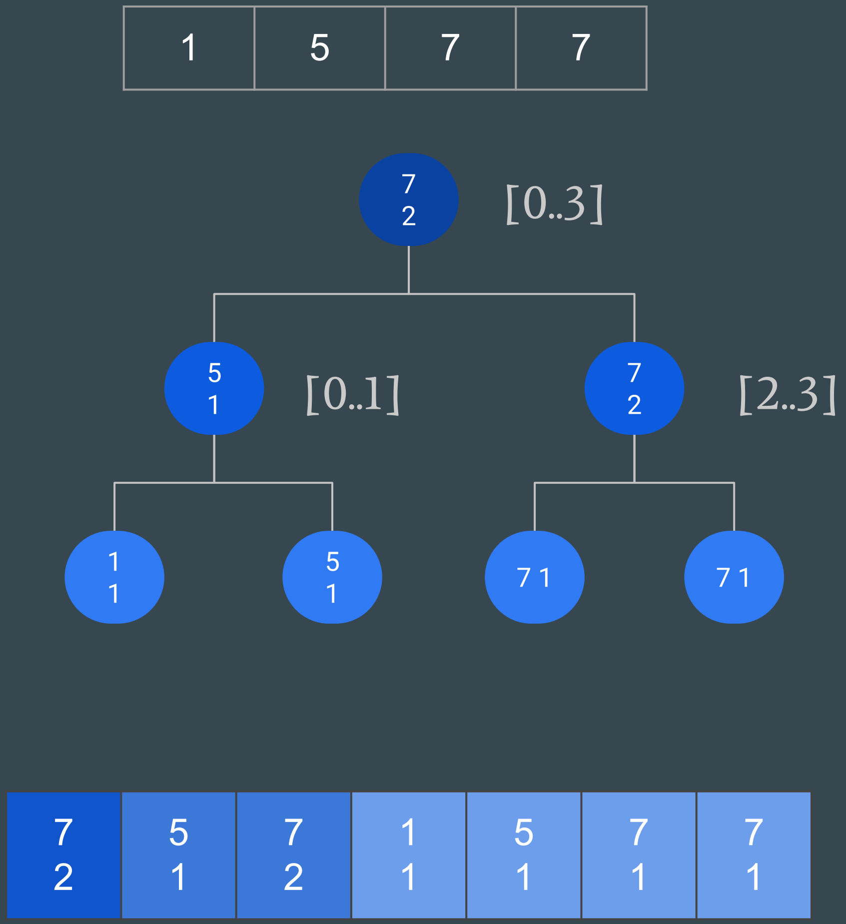
→ Search of the local maximum



Generalization

The heuristic can be adapted to solve a wide range of problems.

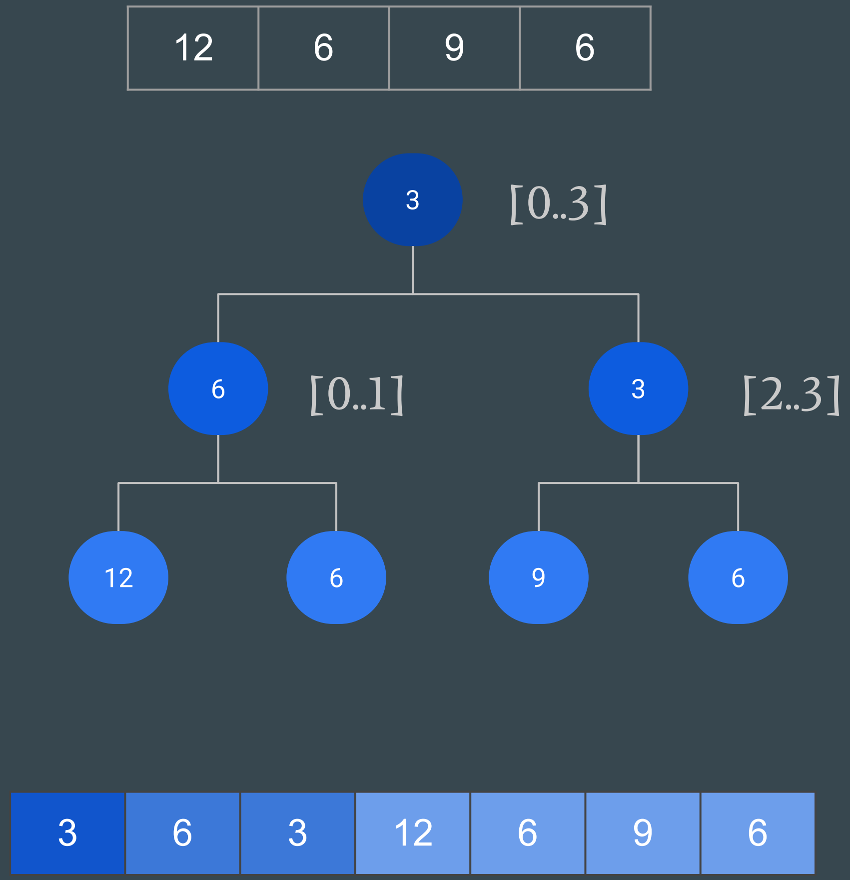
→ Search of the local maximum, and how many times it appears



Generalization

The heuristic can be adapted to solve a wide range of problems.

→ Computing the GCD (or LCM) of a subarray



Implementation

You can implement most
segment trees using 4 functions:

merge()	build()	update()	query()
Compute a node value from its children’s value	Generate the segment tree from a dataset	Updates a value from the dataset and updates the necessary nodes	Compute a property on a given subarray of the dataset

Implementation: merge

Defines how a node value depends on its children.

It is usually called by all other three functions.

```
def merge_func(self, node_left, node_right):
        return node_left + node_right
```

Implementation: build

Creates the tree from a dataset (here from self.data)

It is usually called only once and runs in O(n) if the merge function runs in O(1), because there are n internal nodes in the tree.

```
self.segtree = [0] * (4 * len(data))

def build(self, node, start, end):
    if start == end:
        self.segtree[node] = self.data[start]
    else:
        mid = (start + end) // 2
        self.build(2 * node + 1, start, mid)
        self.build(2 * node + 2, mid + 1, end)
        self.segtree[node] = self.merge(
            self.segtree[2 * node + 1],
            self.segtree[2 * node + 2]
        )
```

Implementation: build

Important: Here, for a node of index i, its children's indexes are 2i and 2i+1 (2i+1 and 2i+2 if our array is 0-indexed).

This is NOT the most optimal indexing method memory-wise, as a segment tree only requires 2n-1 vertices, but it is easier to implement.

```
self.segtree = [0] * (4 * len(data))

def build(self, node, start, end):
    if start == end:
        self.segtree[node] = self.data[start]
    else:
        mid = (start + end) // 2
        self.build(2 * node + 1, start, mid)
        self.build(2 * node + 2, mid + 1, end)
        self.segtree[node] = self.merge(
            self.segtree[2 * node + 1],
            self.segtree[2 * node + 2]
        )
```

Implementation: update

Updates a value from the dataset and updates the affected internal nodes.

This function could be optimized for updating a subarray of the dataset, by generating a smaller segment tree and inserting it.

```
def update(self, node, start, end, idx, value):
    if start == end:
        self.data[idx] = value
        self.segtree[node] = value
    else:
        mid = (start + end) // 2
        if idx <= mid:
            self.update(2*node+1, start, mid, idx, value)
        else:
            self.update(2*node+2, mid+1, end, idx, value)

        self.segtree[node] = self.merge(
            self.segtree[2*node+1],
            self.segtree[2*node+2]
        )
```

Implementation: query

Efficiently compute a property on a given subarray of the dataset, using the segment tree.

This function's implementation might change a lot depending on the property.

```
def query(self, node, start, end, L, R):
    if R < start or end < L:
        return 0

    if L == start and end == R:
        return self.segtree[node]

    mid = (start + end) // 2
    left_sum = self.query(
        2*node+1,
        start, mid,
        L, min(R, mid)
    )
    right_sum = self.query(
        2*node+2,
        mid+1, end,
        max(L, mid+1), R
    )

    return self.merge(left_sum, right_sum)
```

Credits

Slides: William Michaud for INSAlgo

Sources:

- https://cp-algorithms.com/data_structures/segment_tree.html#advanced-versions-of-segment-trees
- https://en.wikipedia.org/wiki/Segment_tree