Advanced Programming Practice
Segment Tree

2022 Fall, CSE4152

Sogang University

Range sum in 1D array

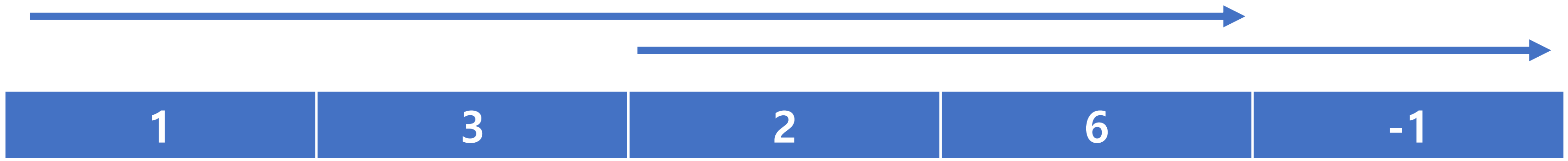
- There is an N integer array. Compute sum of each range. The number of ranges is M.
- Example

1	3	2	6	-1

  - Sum of 1-4 interval: 12
  - Sum of 3-5 interval: 7

Finding a solution

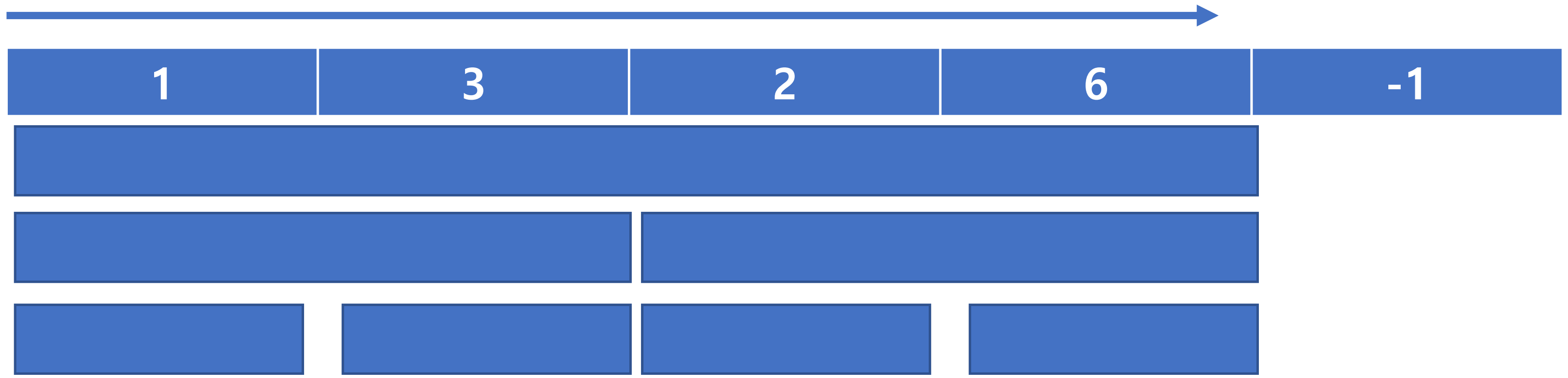
- Method 1: compute interval sum for each query
  - Add all elements of the interval one-by-one.



  - The worst time complexity for each interval sum: O(N)
  - The total time complexity T(N, M)= O(NM)
  - How could we compute sums of intervals quickly?

Finding a solution

- Method 2: Computing interval sum with divide-and-conquer.
  - We divided the interval into two parts recursively (lengths of them should be similar).



  - The worst time complexity for summing up the interval: T(N) = T(N/2) + O(1) = O(logN)
    - The number of binary nodes <= 2N
  - The total time complexity is T(N, M)= O(MlogN).

How could we add up the interval quickly?

- Idea: Let us cumulate the interval.
  - sum(a, b): sum from a to b
  - C[i]: sum(1, i)
  - sum(i, j) = C[j] – C[i-1]
  - It takes O(1) to compute the interval sum if we know cumulated sum at every index.

Solution

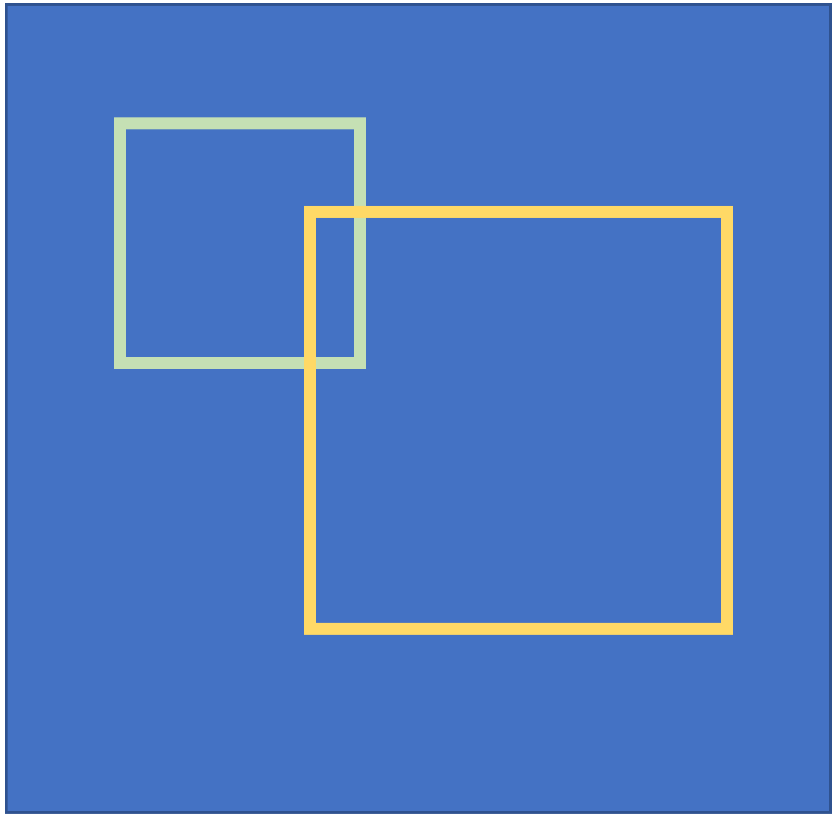
- In preprocessing, we cumulate the array.
  - $O(N)$ time complexity

- Computing sum of the query: $O(1)$

- Total time complexity: $T(N) = O(N) + M * O(1) = O(\max(N, M))$

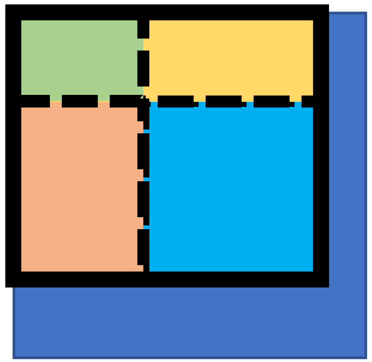
Interval Sum in a 2D integer array

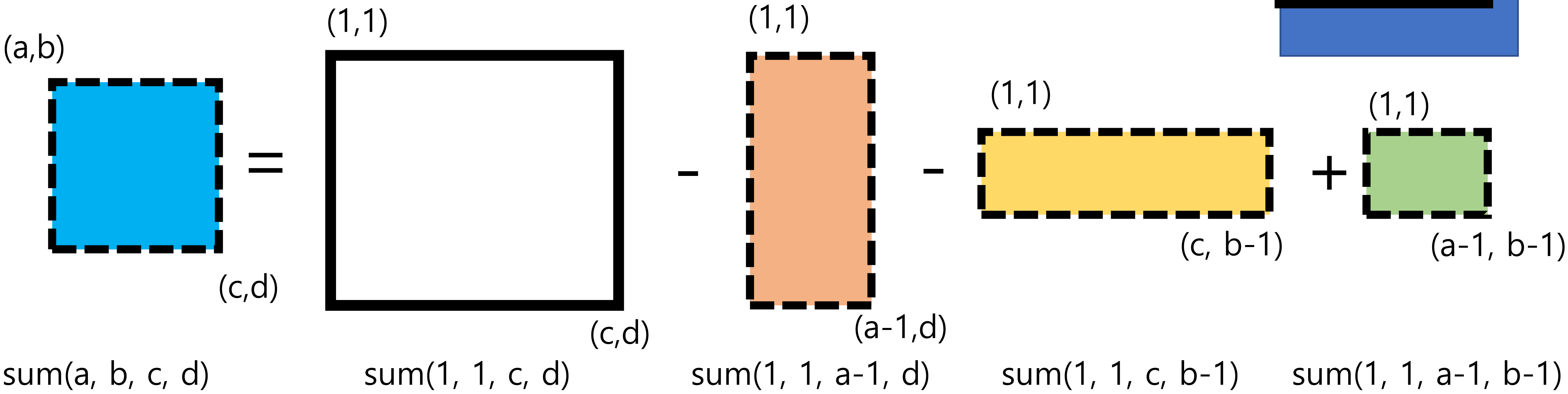
- Method 1. Computing 2D sum for each query.
  - Time complexity: $O(MN^2)$
  - It always suffers from repeated computation.
- Method 2. 2D Cumulation
  - How should we cumulate the 2D array to compute sums of 2D intervals efficiently?



Method 2. 2D Cumulation

- This way we can compute sum of the 2D interval by O(1).





Allowing change of elements

- There are two types of queries: element modification and computing the interval sum.
  - The interval sum would be changed after modifying one element.
- Example

1	3	2 -> -1	6	-1

  - Sum of 1-4 interval: 12
  - Modify the 3rd element to -1
  - Sum of 3-5 interval: 4

Cumulation is not helpful

- Cumulation: O(N)
- Computing sum of an interval: O(1)
- Modifying an element value: O(N)
  - Because we need to cumulate an 1D array again.
- The total time complexity: T(N) = O(N) + M * O(1+N) = O(MN)

How could we update the cumulation array efficiently?

How could we update the cumulation array efficiently?

- Segment Tree

1-8
112

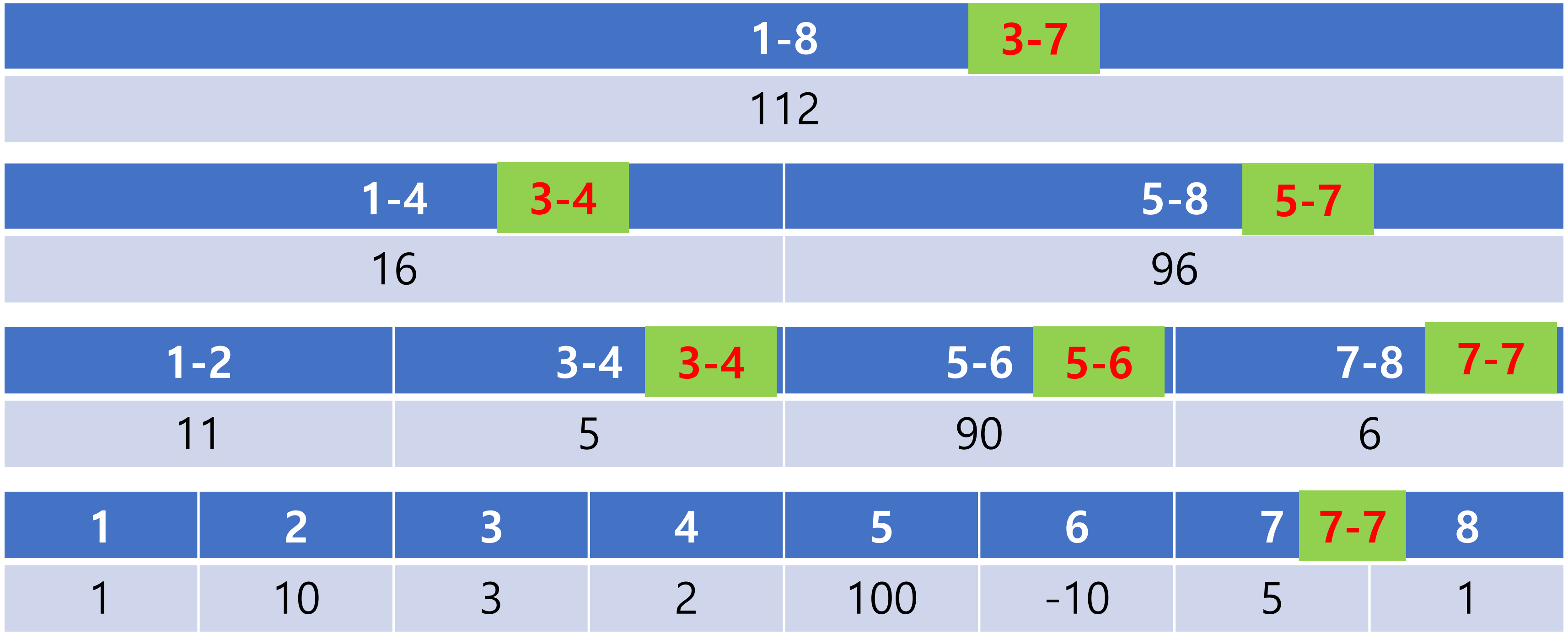
1-4	5-8
16	96

1-2	3-4	5-6	7-8
11	5	90	6

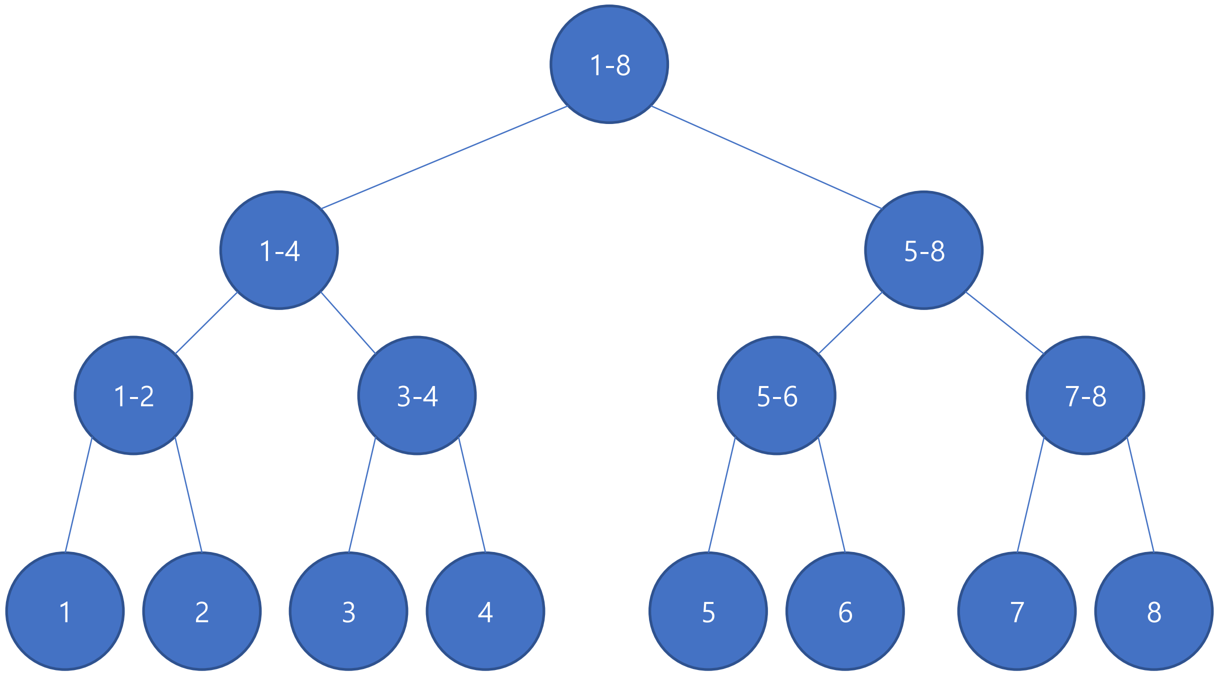
1	2	3	4	5	6	7	8
1	10	3	2	100	-10	5	1

How could we update the cumulation array efficiently?

- Computing sum of an interval: T(N) = T(N/2) + O(1) = O(log N)
  - Travel the tree from the root to the bottom.
  - If the node interval is included in the target interval, (예: 5-6).
  - Note that we track both ends so we only need to travel two paths.



Segment tree



How could we update tree info fast?

- Modifying the element and sums of related intervals: T(N) = O(log N)
  - We travel nodes that include the changed element and add the difference to every traveled node.

1-8
112 + (8-3) = 117

1-4	5-8
16 + (8-3) =21	96

1-2	3-4	5-6	7-8
11	5+ (8-3) = 10	90	6

1	2	3	4	5	6	7	8
1	10	3 ->8	2	100	-10	5	1

How could we update tree info fast?

- We don’t need to initialize the segment tree.
- We update only changes on the segment tree.
- We compute sum of an interval by adding two sums: interval sum on the initial array and delta sum from the segment tree.

1-8
0 + (8-3) = 5

1-4	5-8
0 + (8-3) =5	0

1-2	3-4	5-6	7-8
0	5	0	0

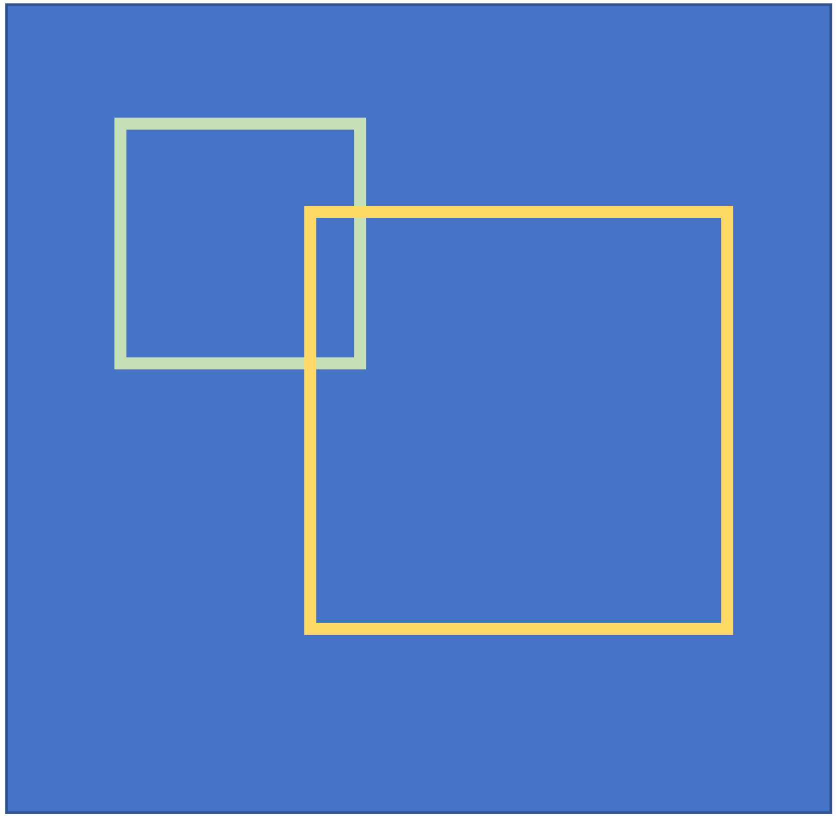
1	2	3	4	5	6	7	8
0	0	5	0	0	0	0	0

Solution: use segment tree

- Cumulate the array: O(N)
- Compute sum of changes in the interval: O(log N)
- Update the segment tree: O(log N)
- Total time complexity T(N) = O(N) + M * O(log N) = O(MlogN)

Element modification on 2D array.

- The naïve solution would take $O(MN^2)$ to update cumulation info.
- How could we update cumulation info efficiently?

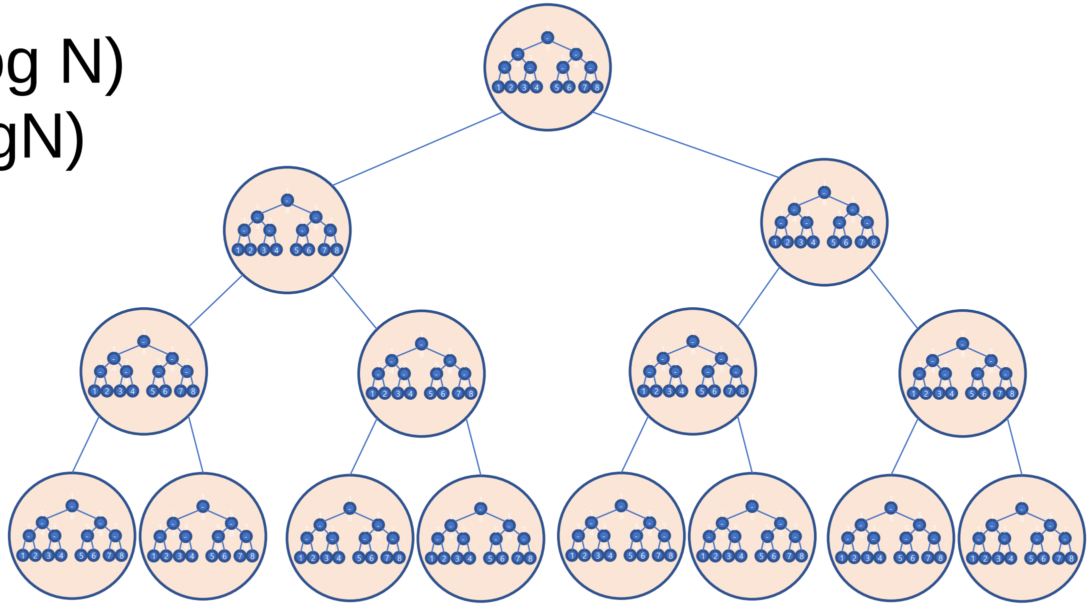


Double Segment Tree

- We have two hierarchical levels, a top level for columns and a low level for rows.
- Each node in the column tree contains a segment tree for rows.

- Computing sum of 2D interval
  - The number of releated column nodes O(log N)
  - Computing sum of rows at each node O(logN)

- Modifying one element
  - Update every low-level segment tree
    - O(logN * logN)
    - Need to update logN column-interval nodes.



Bigger N

- We need $O(N^2)$ low-level nodes.

- If N is 10 thousands, it may suffer from memory overflow.

Dynamic segment tree

- We only update changes on the segment tree.
  - That means we don't need to update the tree because sums of changes of all intervals are 0.

- When modifying the element, we dynamically allocate necessary nodes and update sum info.

- For the 2D interval problem,
  - at most O(M) O(logNlogN) nodes can be generated.
    - M is the number of "modify" queries.
  - The upper-bound of nodes is $O(N^2)$ which is much larger than O(MlogNlogN)

Assignment

1D sum

For given N integer elements and Q queries, please compute interval sum or modify the element. There are two types of queries: summing up elements in the interval and modifying the element.

N <=1000000 Q <= 10000

Input

5	// input N
1 2 3 4 5	// N elements
3	// Q queries
0 0 3	// 0 indicates computing the sum of the interval. The result is 10.
1 1 1	// 1 indicates modification. The second element is changed to 1
0 0 3	// The result is 9.

Hint: use an interval tree and the sum table together.

2D sum

For given NxM integer elements and Q queries, please compute 2D sums for given Q intervals.

N<=1000, M<=1000, Q <=1000

Input

```
5 3             // input N
1 2 3 4 5       // NxM elements
1 2 3 4 5
1 2 3 4 5
2               // Q queries
0 2 0 2 // 18
0 0 0 0         // 1
```