15451 Spring 2023

Range query data structures

Elaine Shi

Range queries have many applications

- Databases:
  Select avg(**price**) from **Trades** where **time** ∈ [2023/1/1, 2023/2/1]

Range queries have many applications

- Databases:
  Select avg(**price**) from **Trades** where **time** ∈ [2023/1/1, 2023/2/1]
- Computational geometry
- Geographic information systems
- Computer-aided design
- **Connection to parallel algorithms**
- Cryptography (Elaine's favorite topic)

Today: 1-D range query

Today: 1-D range query

Suppose we have an array a[0], a[1] … , a[n-1]

Design a data structure that supports:

RangeSum(i, j): Return $\sum_{i \leq k < j} a[k]$

✓ Space : O(n)

Query time : O(n)

Idea 1:

Store the array a

Compute the sum $\sum_{i \leq k < j} a[k]$ on the fly

Idea 1:

Store the array a

Compute the sum $\sum_{i \leq k < j} a[k]$ on the fly

O(n) space, O(n) time per query

Idea 2:

Precompute all prefix sums $b_k = \sum_{0 \leq i < k} a[i]$

On query (i, j), return $b_j - b_i$

Idea 2:

Precompute all prefix sums $b_k = \sum_{0 \leq i < k} a[i]$

On query (i, j), return $b_j - b_i$

O(n) pre-processing,
O(n) space, O(1) query time

Suppose we also want to support update

Suppose we have an array a[0], a[1] … , a[n-1]

Design a data structure that supports:

RangeSum(i, j): Return $\sum_{i \leq k < j} a[k]$

Update(i, x): update a[i] to x

What's the cost of **Update** in Idea 2?

Precompute all prefix sums $b_k = \sum_{0 \leq i < k} a[i]$

On query (i, j), return $b_j - b_i$

Idea 1:

Store the array a

Compute the sum $\sum_{i \leq k < j} a[k]$ on the fly

Idea 1: O(1) Update, O(n) RangeSum

Store the array a

Compute the sum $\sum_{i \leq k < j} a[k]$ on the fly

Good for infrequent queries and frequent updates

Idea 2:

Precompute all prefix sums $b_k = \sum_{0 \leq i < k} a[i]$

On query (i, j), return $b_j - b_i$

Idea 2: O(n) update, O(1) RangeSum

Precompute all prefix sums $b_k = \sum_{0 \leq i < k} a[i]$

On query (i, j), return $b_j - b_i$

Good for frequent queries and infrequent updates

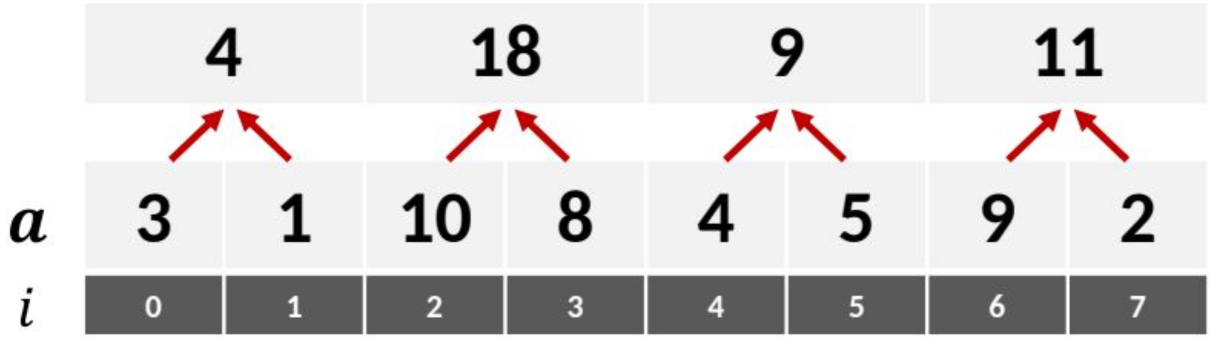
Can we balance the **Update** and **RangeSum** costs?

e.g., suppose queries and updates are both frequent

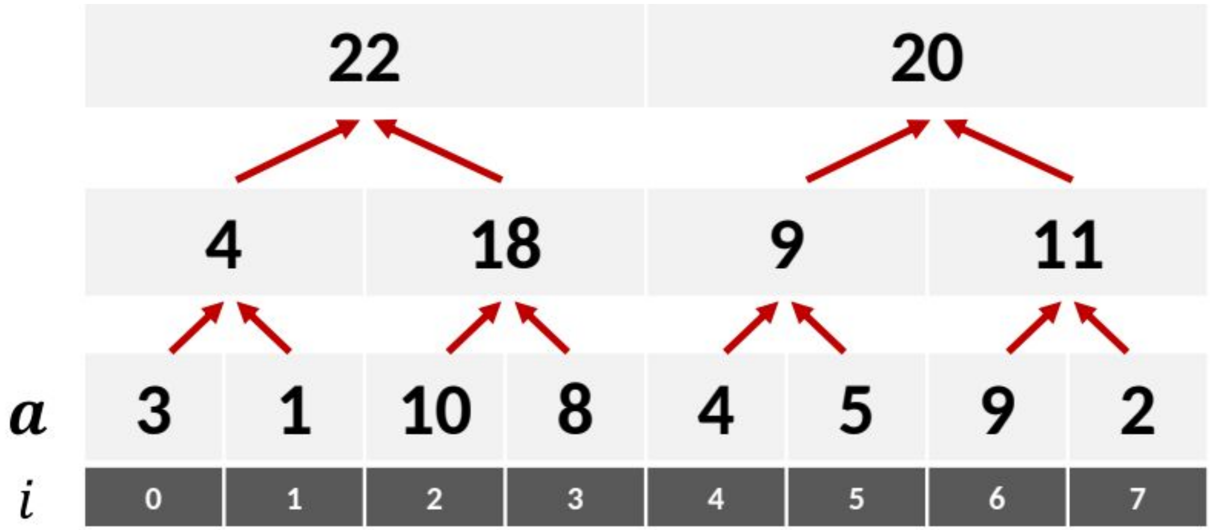
Inspired by parallel algorithms

a	3	1	10	8	4	5	9	2
i	0	1	2	3	4	5	6	7

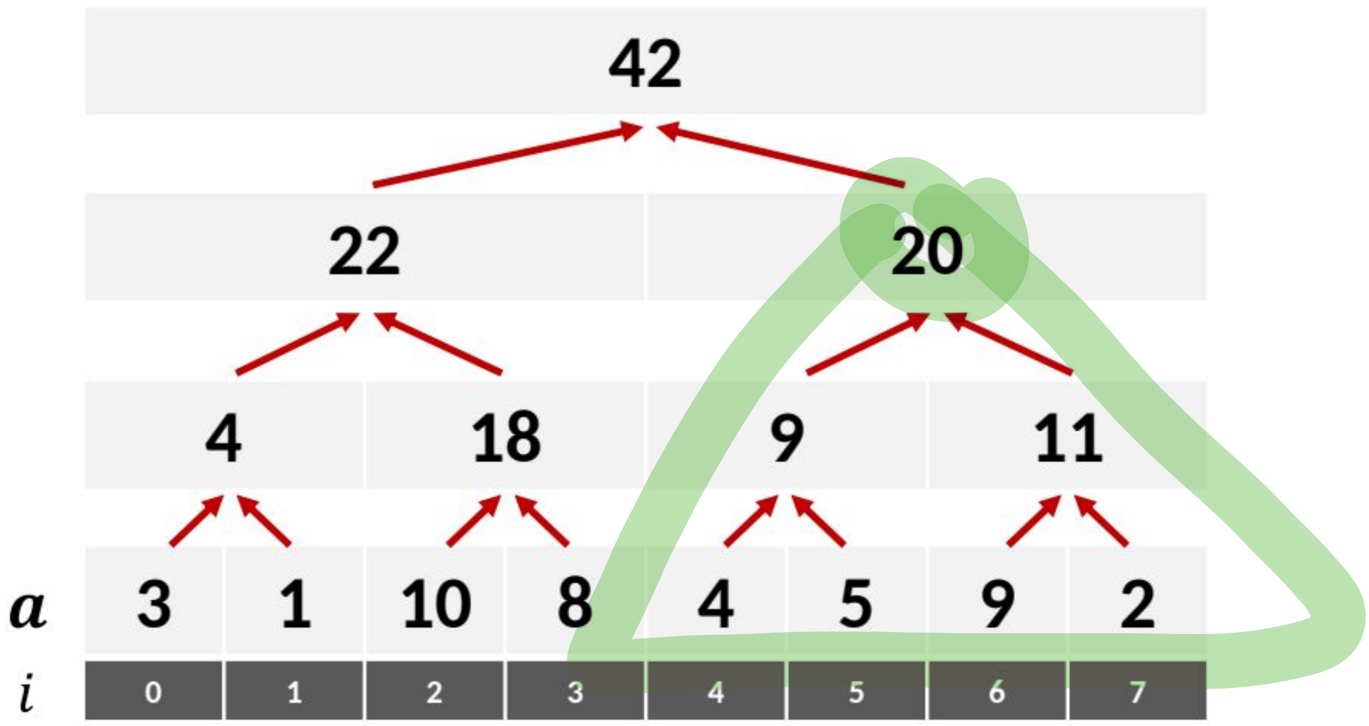
Inspired by parallel algorithms



Inspired by parallel algorithms

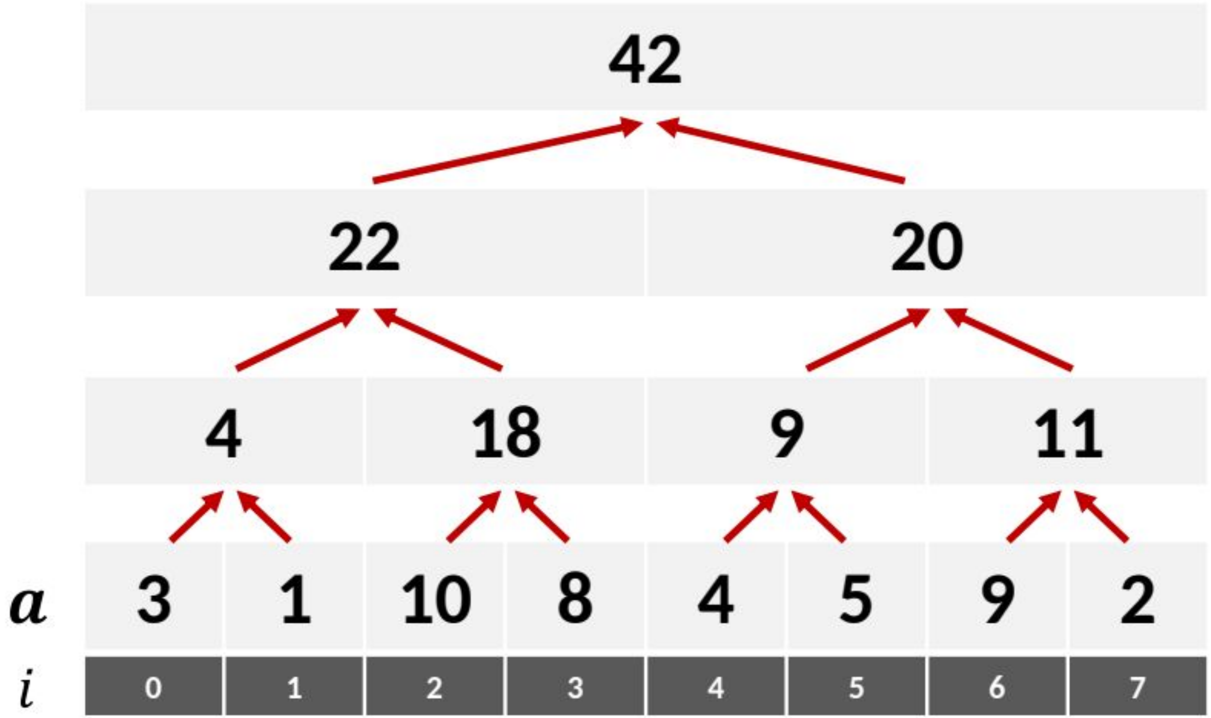


Inspired by parallel algorithms

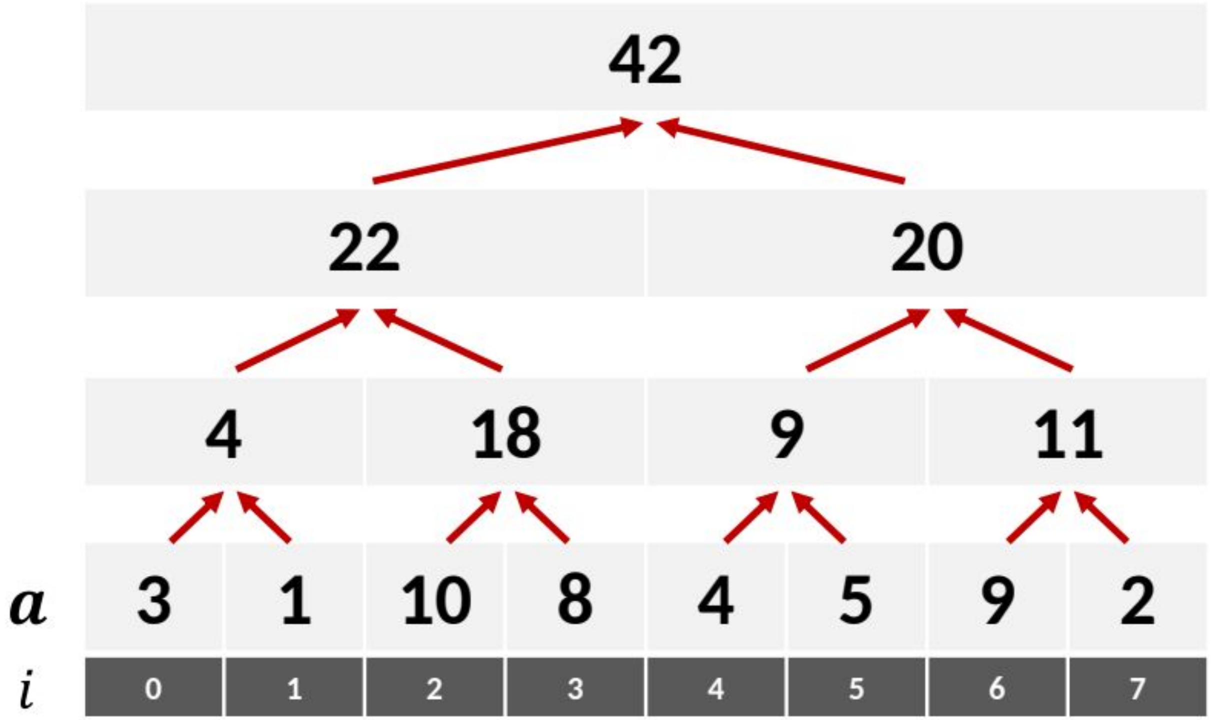


Preprocess:

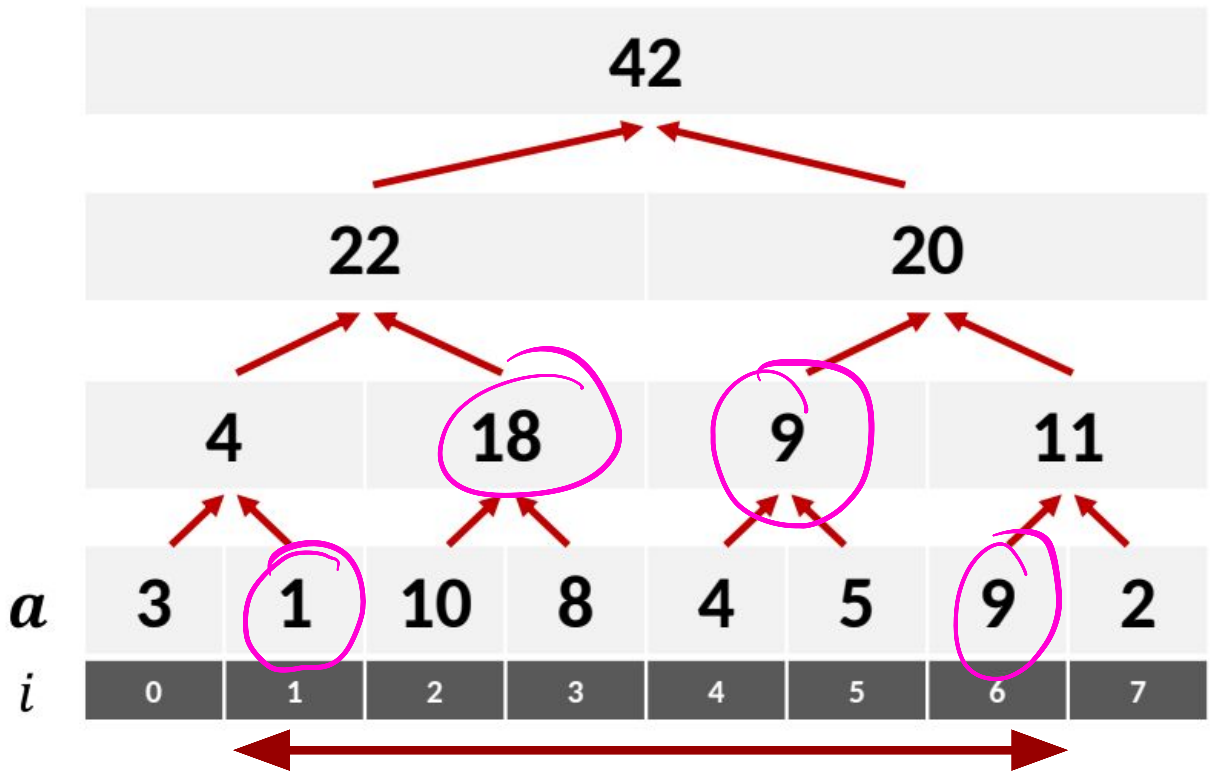
- compute this tree
- each node stores the sum of the subtree



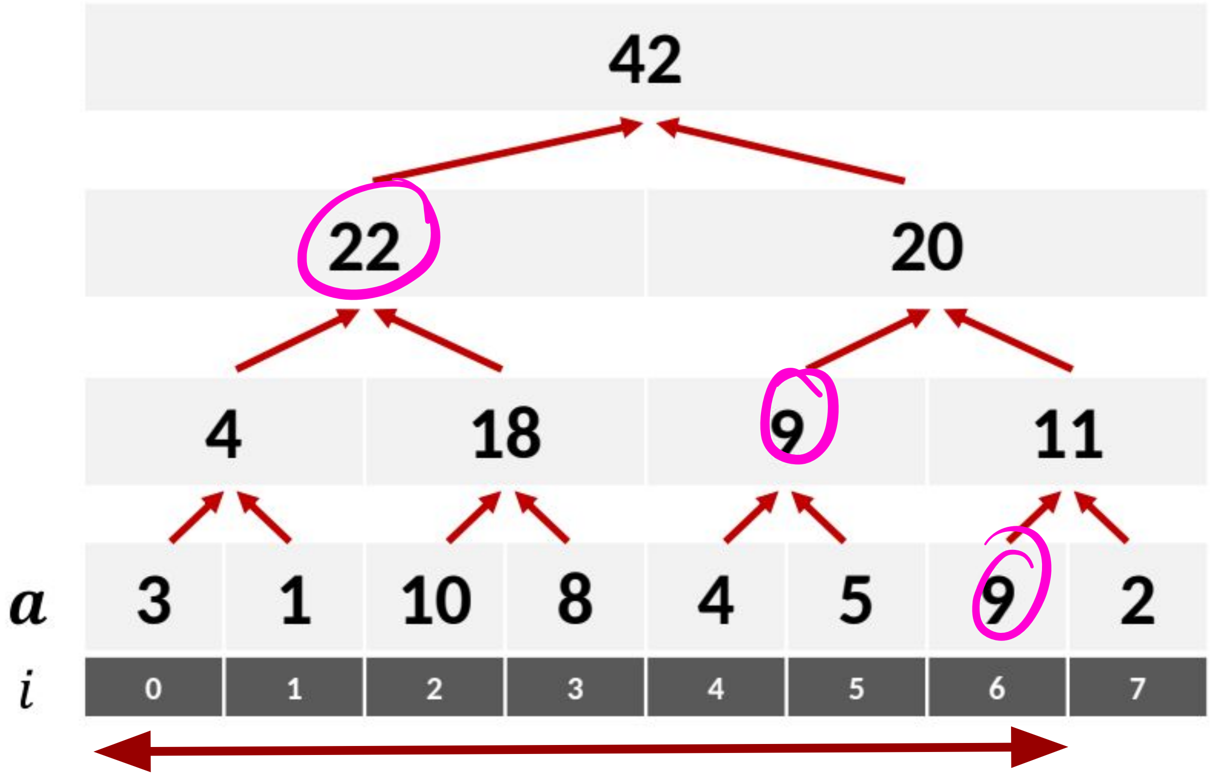
How to support **RangeSum** and **Update**?



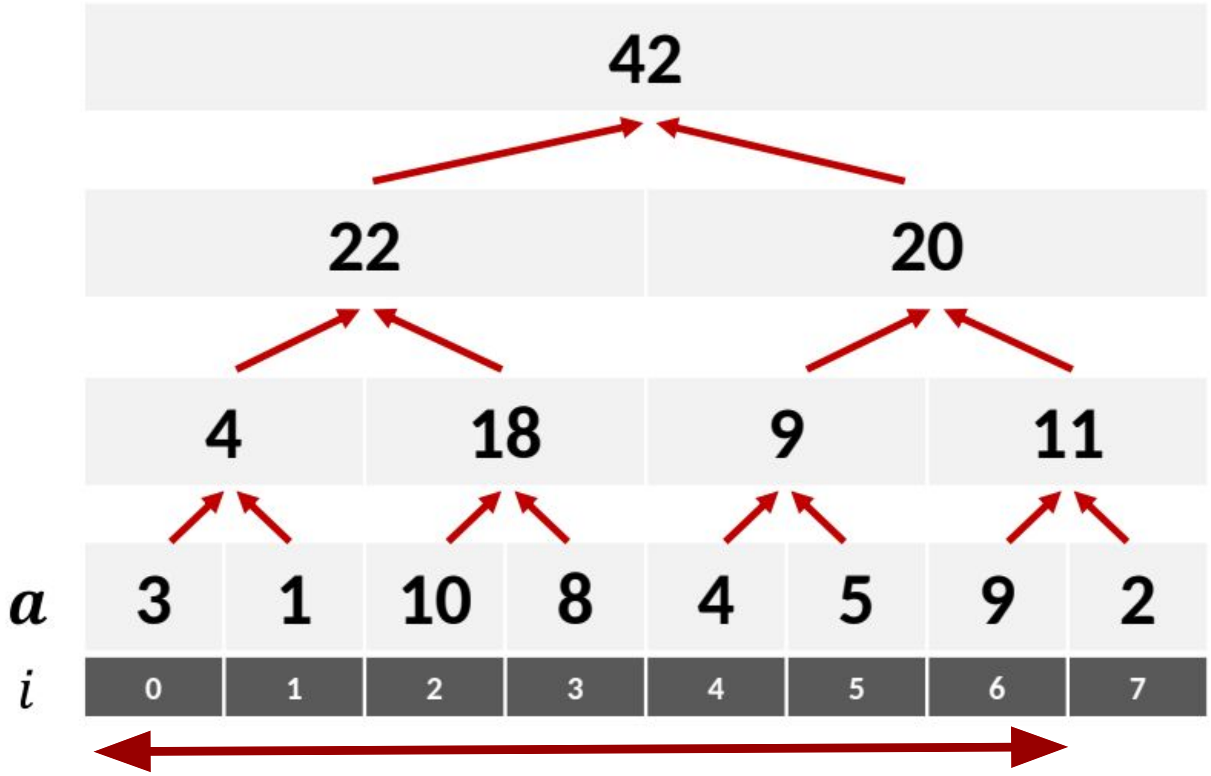
RangeSum(1, 6) query



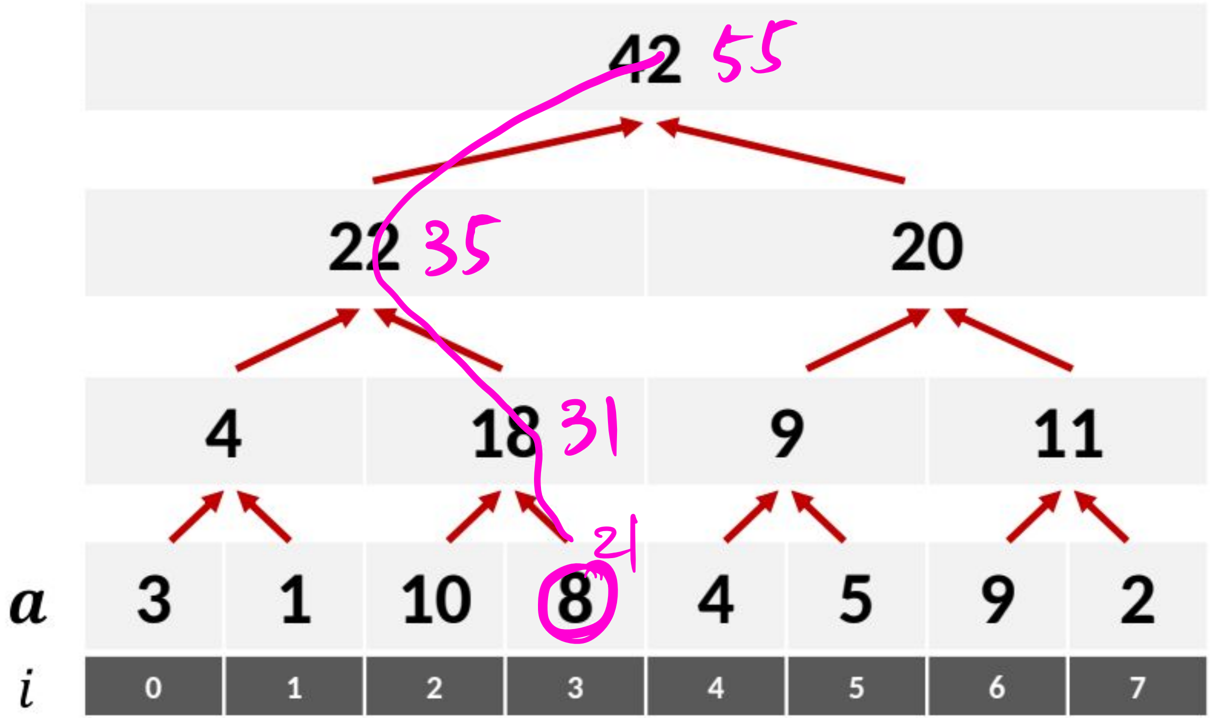
RangeSum(0, 6) query



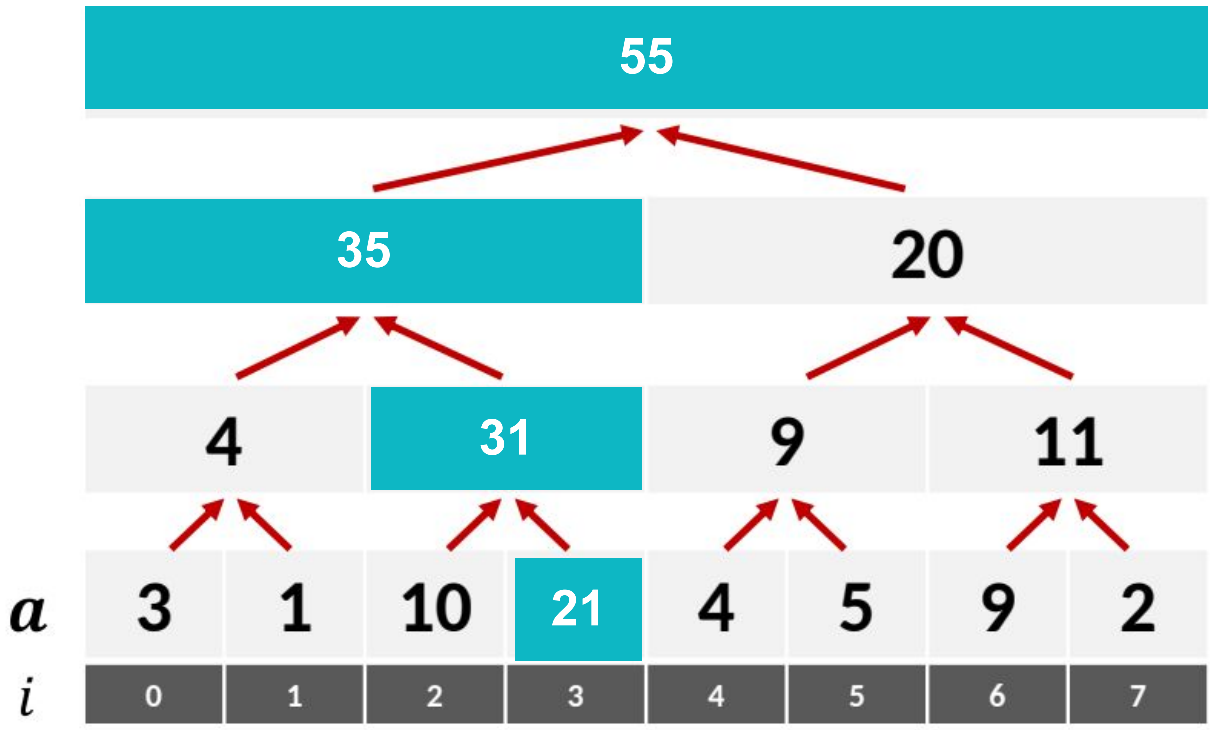
Claim: every range sum is the sum of at most $2\log_2 n$ nodes in the tree



Update(3, 21)



Update(3, 21) takes log n time!



Claim: any interval [i, j) can be made up by at most 2 intervals from each level.

interval node

Claim: any interval [i, j) can be made up using at most 2 intervals from each level.

Claim: any interval [i, j) can be made up using at most 2 intervals from each level.

Proof:

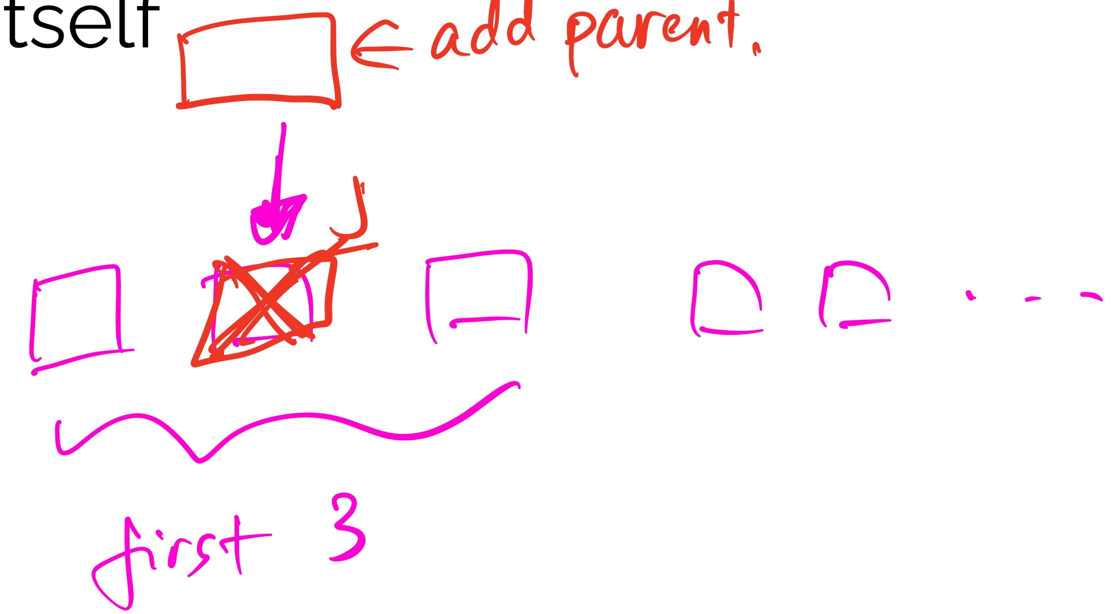
- Suppose that [i, j) is expressed by some configuration C that contains more than 2 intervals in some level(s)

Claim: any interval [i, j) can be made up using at most 2 intervals from each level.

Proof:

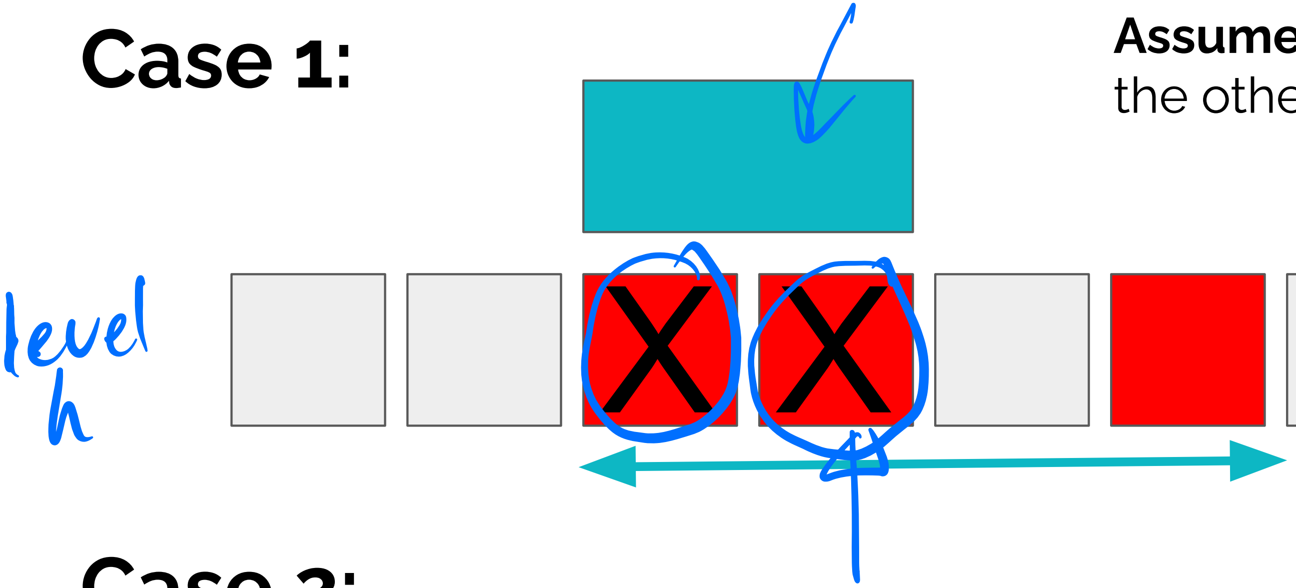
- Suppose that [i, j) is expressed by some configuration C that contains more than 2 intervals in some level(s)
- Let h be the deepest level with > 2 intervals
- Construct another config D that represents [i, j) s.t.
  - **Every level deeper than h uses at most 2 intervals**
  - **Level h uses at most s-1 intervals**

- Find the middle interval J at level h
- Add its parent P
- Delete all of P's descendants including J itself



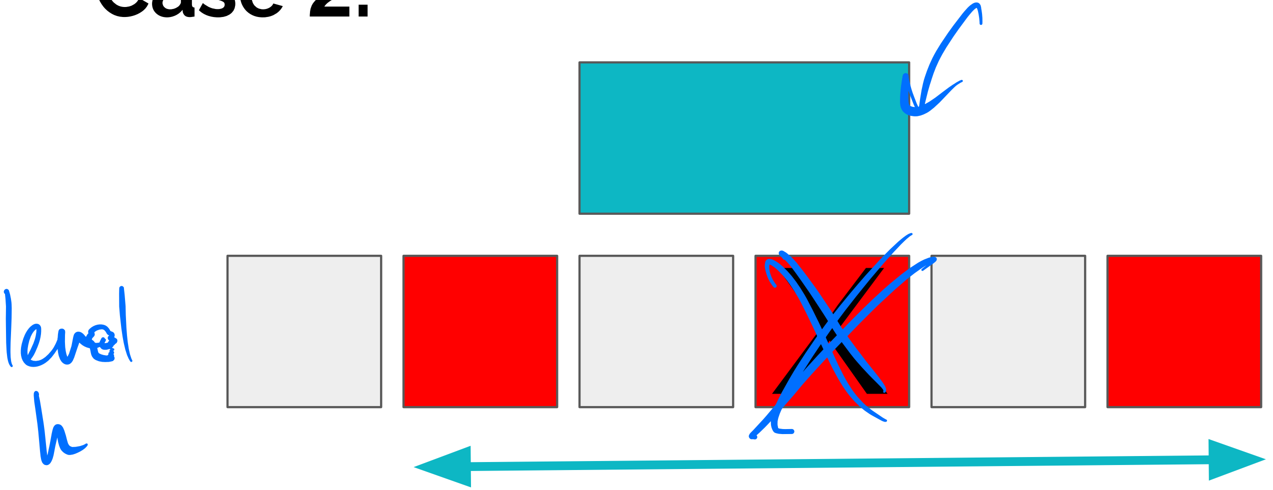
Case 1:

Assume: middle interval is right child, the other case is mirroring



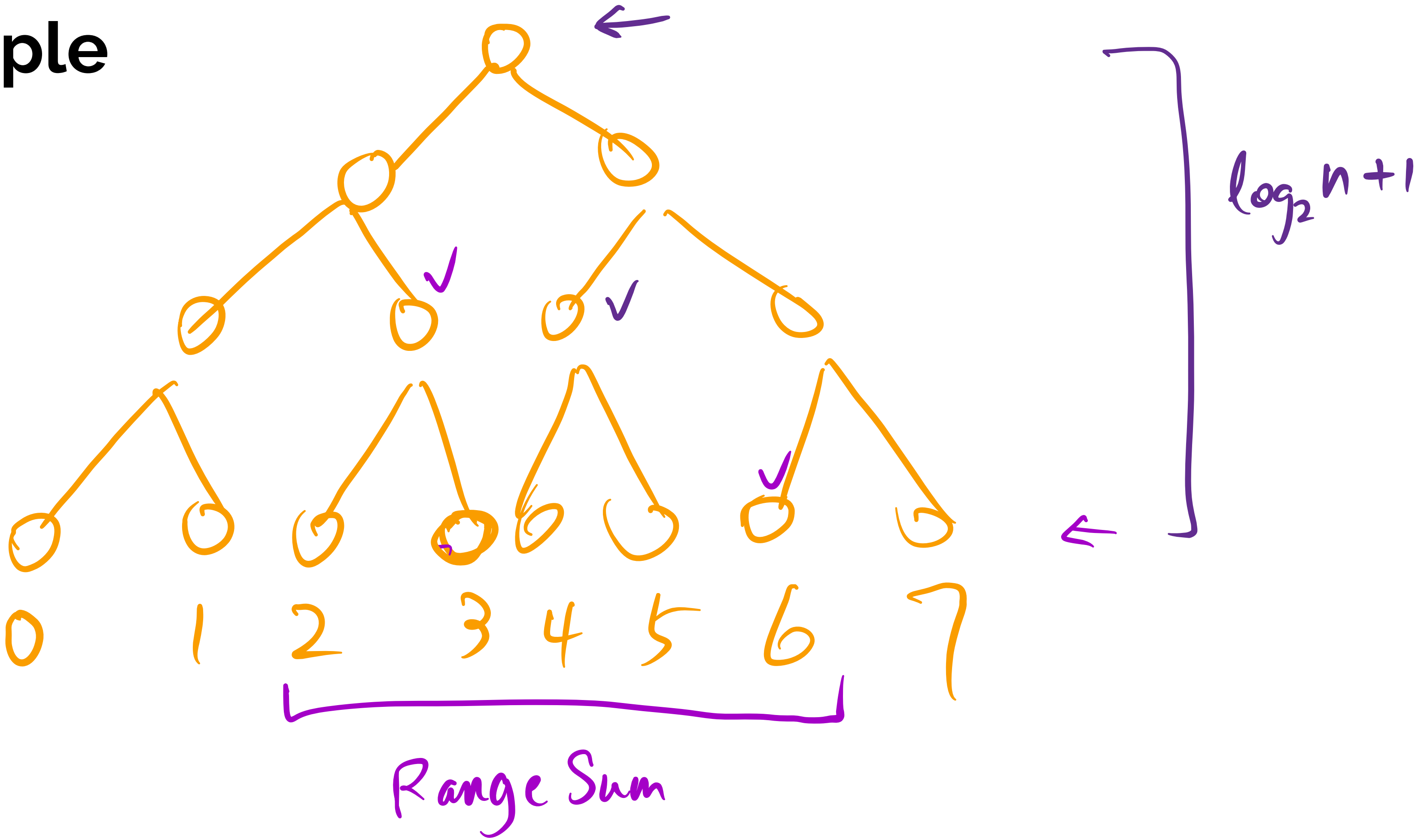
middle interval is paired with another node in C

Case 2:

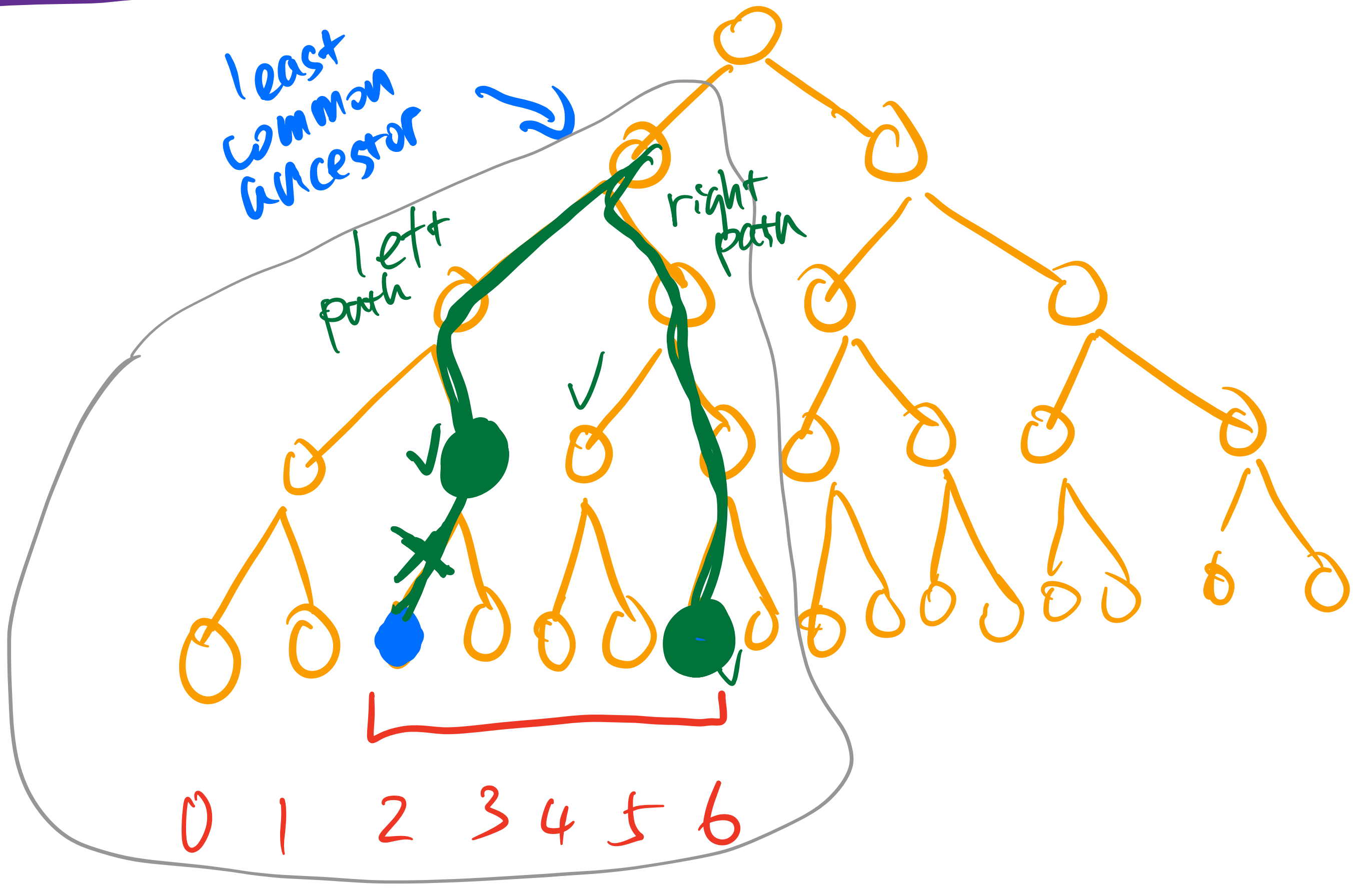


middle interval not paired with another node in C

Example



Finding the intervals in O(log n) time



Implementing the Range Tree

Take advantage of the binary heap ordering trick

Store tree in an array

- Root index: 0
- LeftChild(i) = 2i + 1
- RightChild(i) = 2i + 2
- Parent(i) = $\lfloor (i-1)/2 \rfloor$

See notes for the detailed pseudocode

Speeding up algorithms with Range Trees

Inversion counting

Given a permutation p of 0.. n-1

 # inversions of p:

# of pairs (i, j) such that i < j but p[i] > p[j]

Inversion counting

Example: 5, 3, 1, 2, 6, 0, 4, 8, 7

5: /

3: 5

1: 5, 3

2: 5, 3

6: /

⋮

Example: 5, 3, 1, 2, 6, 0, 4, 7

$O(n^2)$

0 5:

1 3: 5

2 1: 5, 3

2 2: 5, 3

0 6:

5 0: 5, 3, 1, 2, 6

2 4: 5, 6

+ 1 7:

13

Design an algorithm for inversion counting

Naive: $O(n^2)$ time

Example: **5, 3, 1, 2, 6, 0, 4, 7**

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

Example: 5, 3, 1, 2, 6, 0, 4, 7

Total = 0

RangeSum(5, 7): 0

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

Example: 5, 3, 1, 2, 6, 0, 4, 7

Total = 0

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

Example: 5, 3, 1, 2, 6, 0, 4, 7

Total = 1

RangeSum(3, 7): 1

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

Example: 5, 3, 1, 2, 6, 0, 4, 7 **Total = 1**

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

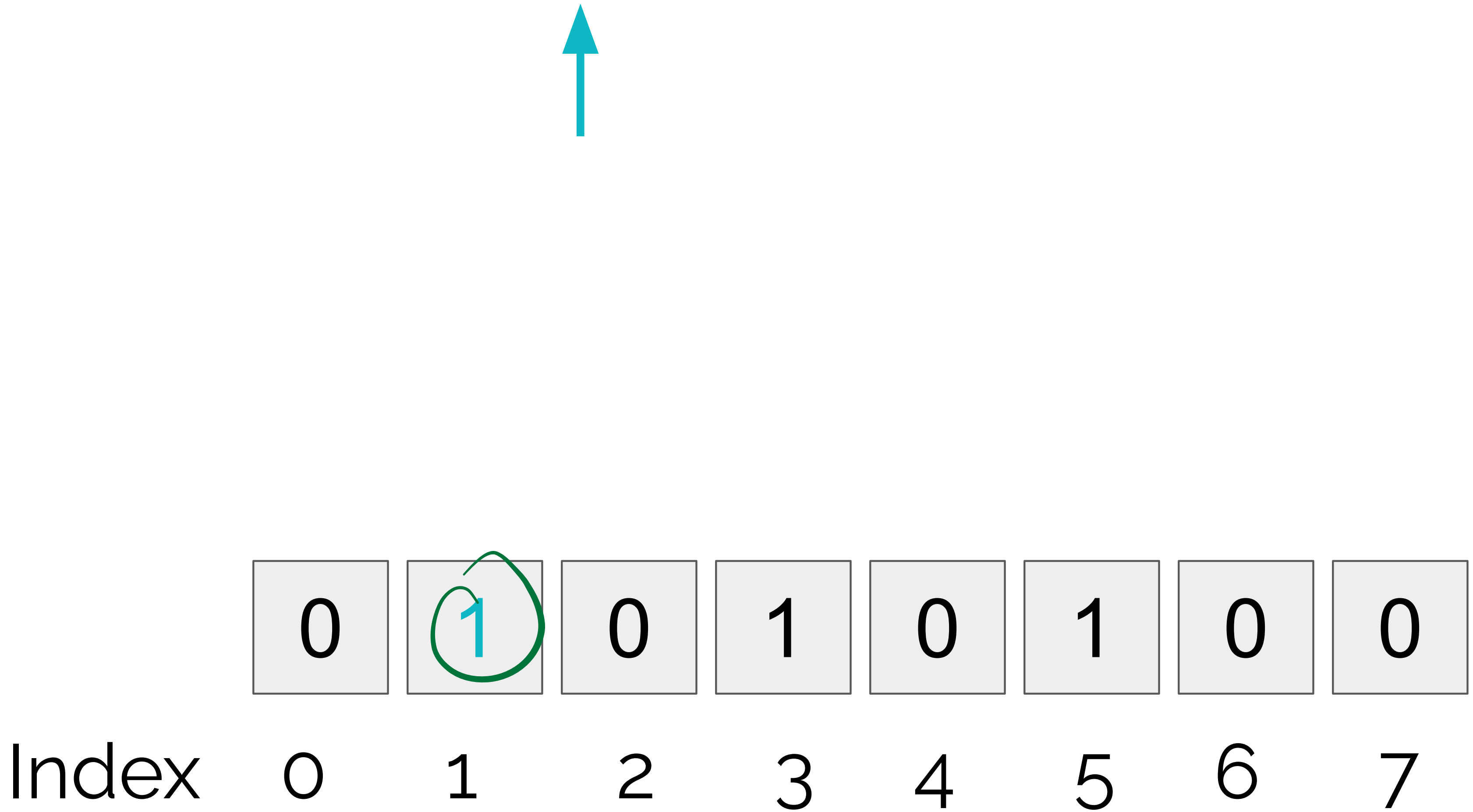
Example: 5, 3, 1, 2, 6, 0, 4, 7

Total = 3

RangeSum(1, 7): 2

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

Example: 5, 3, 1, 2, 6, 0, 4, 7 **Total = 3**



Just repeat this for the entire input

For every elem that arrives:

→ one RangeSum ← O(log n)

→ one Update ← O(log n)

Running time? O(n log n)

Extensions of Range Trees

Sum need not be sum

$(A \circ B) \circ C = A \circ (B \circ C)$

Can be any associate operator
e.g., min, max, average

How do we support

Add(i, x): update a[i] to a[i] + x

Update(i, x): assign x to a[i]

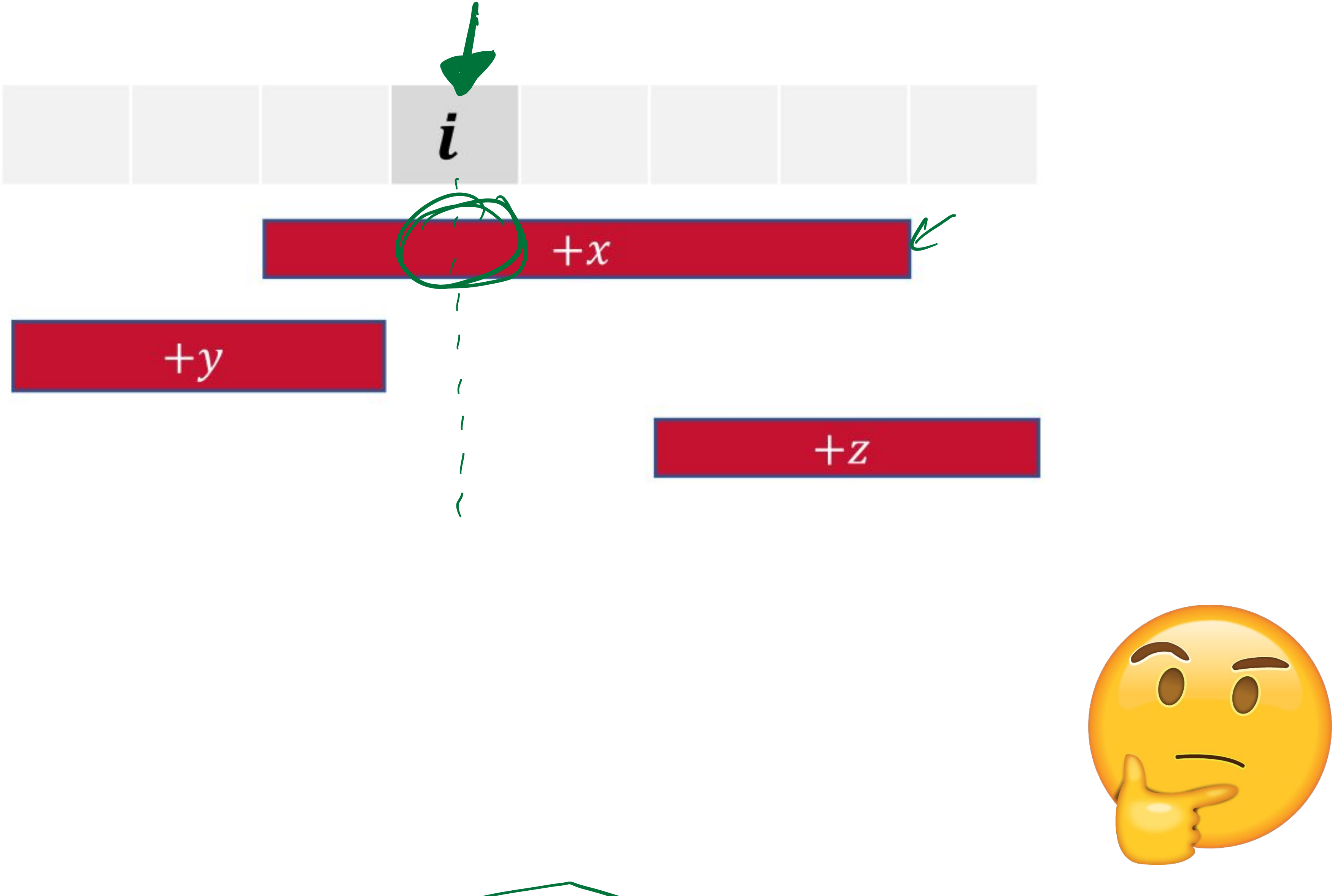
RangeSum(i, i+1)

update(i, a[i]+x)

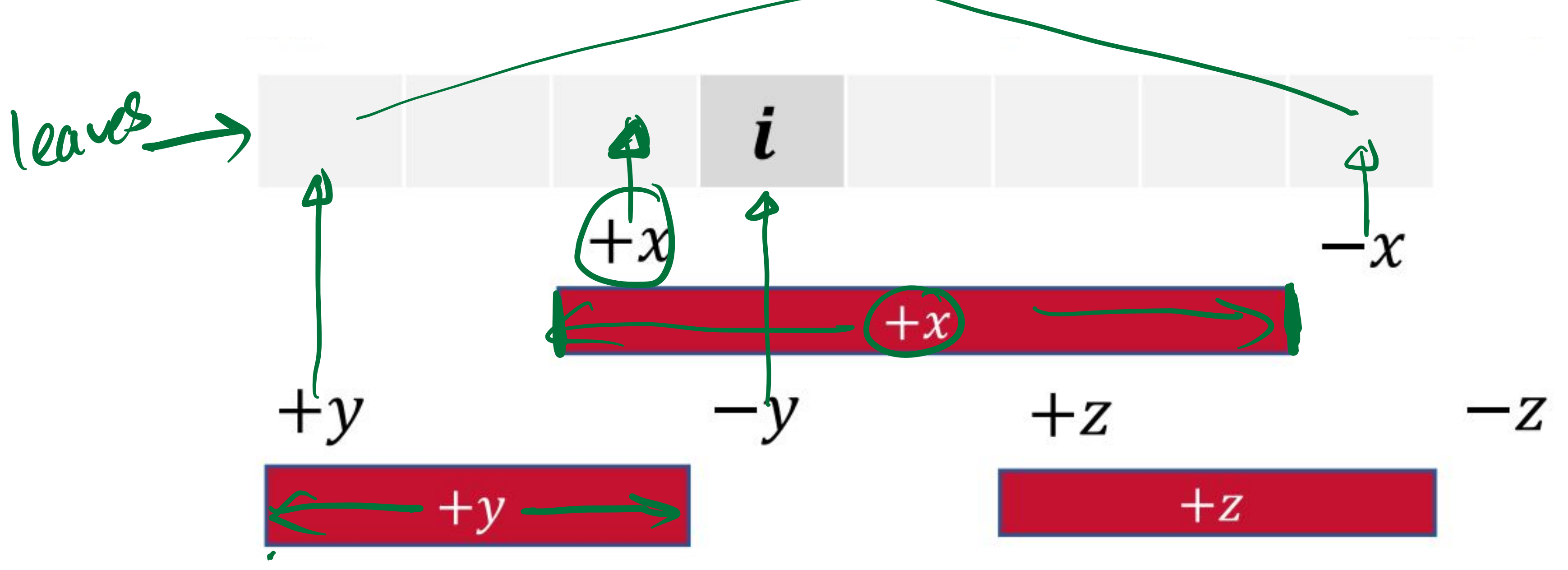
Design a data structure that supports

RangeAdd(i, j, x): add x to a[i] … a[j-1]

GetVal(i): return a[i]



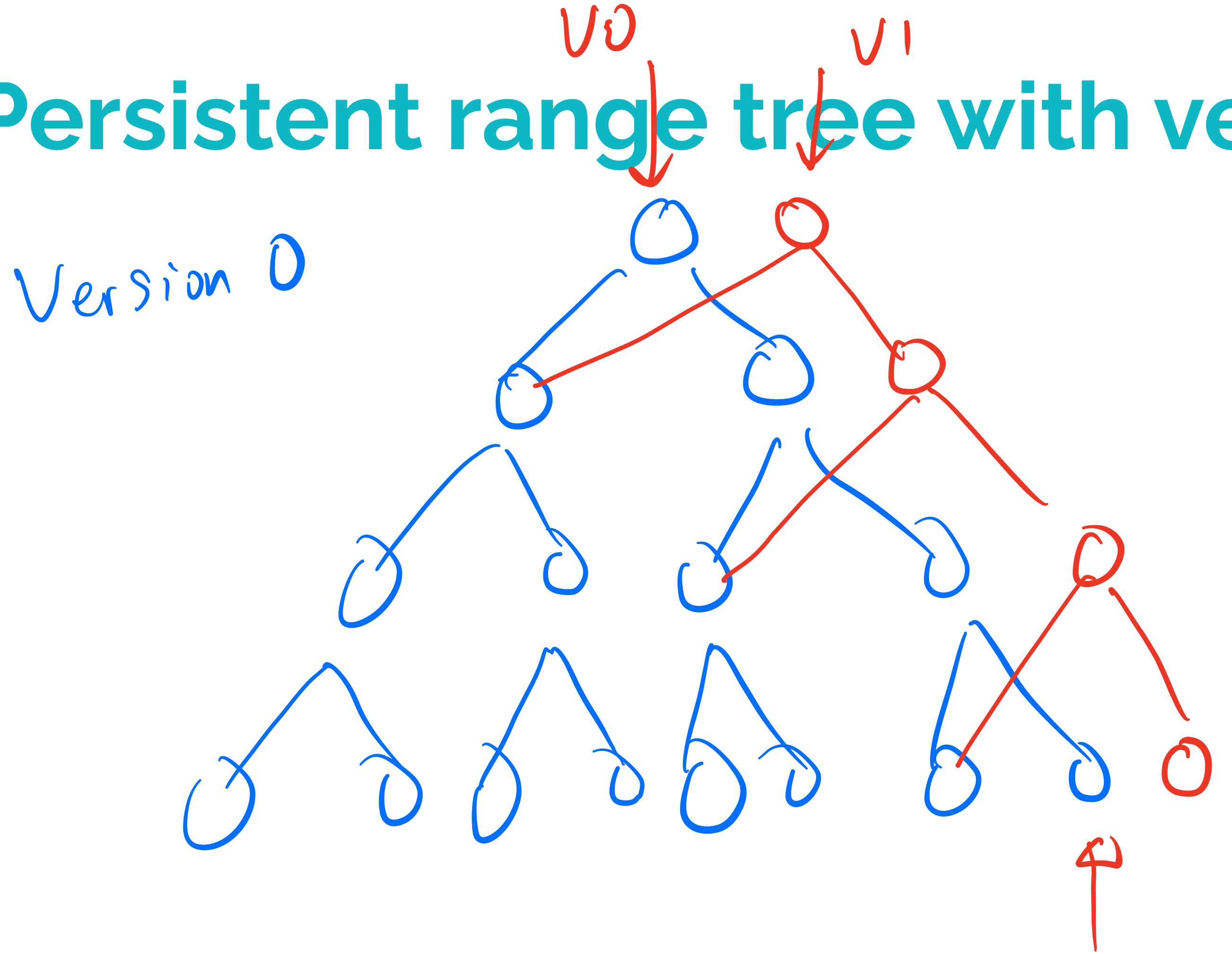
i
$+x$
$+y$
$+z$



GetVal(i): RangeSum(0, i+1)

RangeUpdate(i, j, x): Add(i, x)
Add(j, -x)

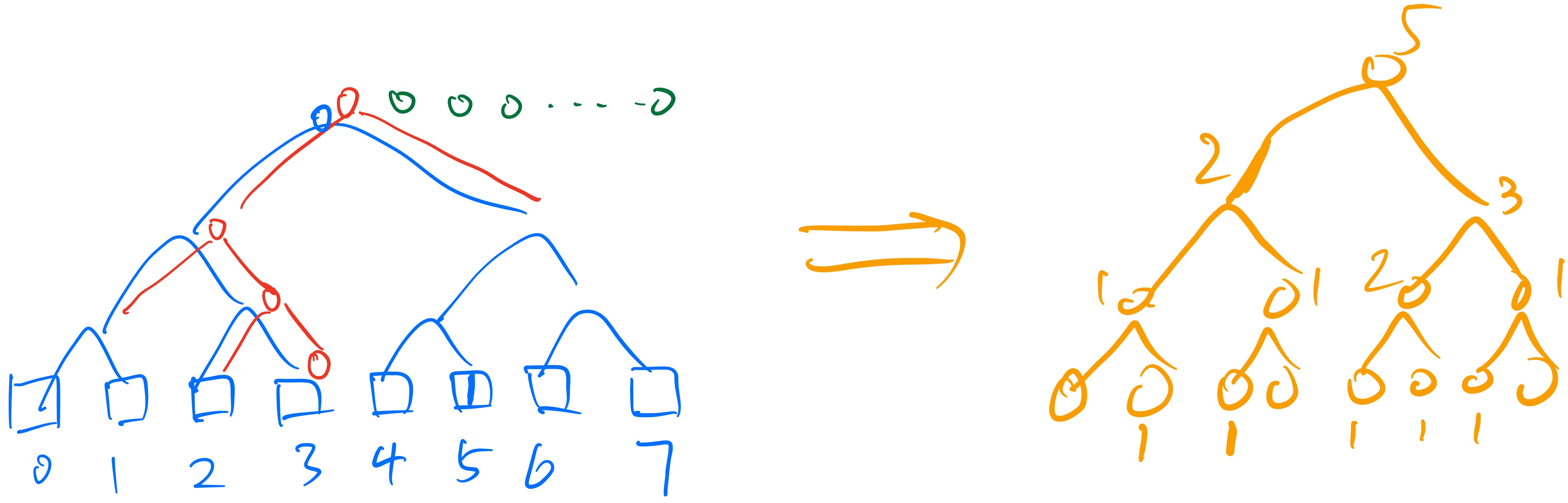
Persistent range tree with versioning



Optional: finding the k-th smallest number in some prefix

[0, j)

5 3 1 7 2 0 6 ...



Just for fun: applications of range tree in cryptography and privacy

- Privacy-preserving federated learning
  - Deployed by Google's Spanish GBoard
  - Uses Elaine's algorithm!
- Range queries over **encrypted** data (see Elaine's Ph.D. thesis!)
- Puncturable PRFs
- Efficient private information retrieval
- Efficient/parallel data-oblivious algorithms