Lecture 8

RANGE-QUERY DATA STRUCTURES

(SEGTREES)

Range Queries

Given $a_0, \ldots a_{n-1}$

Ask queries about $a_i, \ldots a_{j-1}$

Eg. sums

Approach 1 Loop over $a_i \ldots a_{j-1}$ $O(n)$

Approach 2 Compute prefix sums $P_{0 \ldots n}$

answer = $P_j - P_i$ $O(1)$

$O(n)$ preprocessing

Updates

Goal: API

Assign (i, x): $a_i \leftarrow x$

RangeSum (i, j): return $\sum_{i \le k < j} a_k$

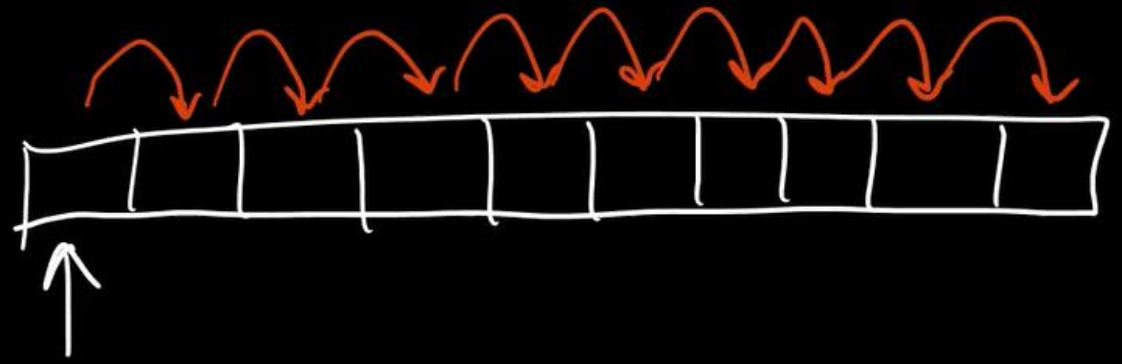
Approach 1

Assign $O(1)$, Query $O(n)$

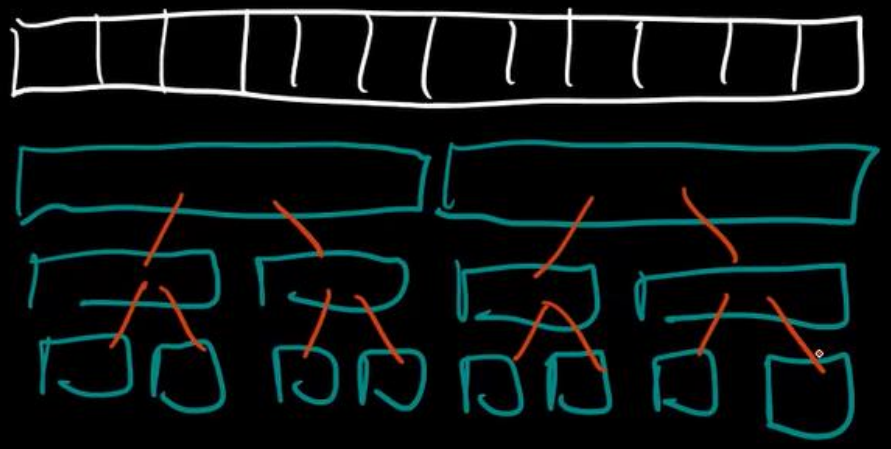
Approach 2

Assign: $O(n)$ Query $O(1)$

Dependencies



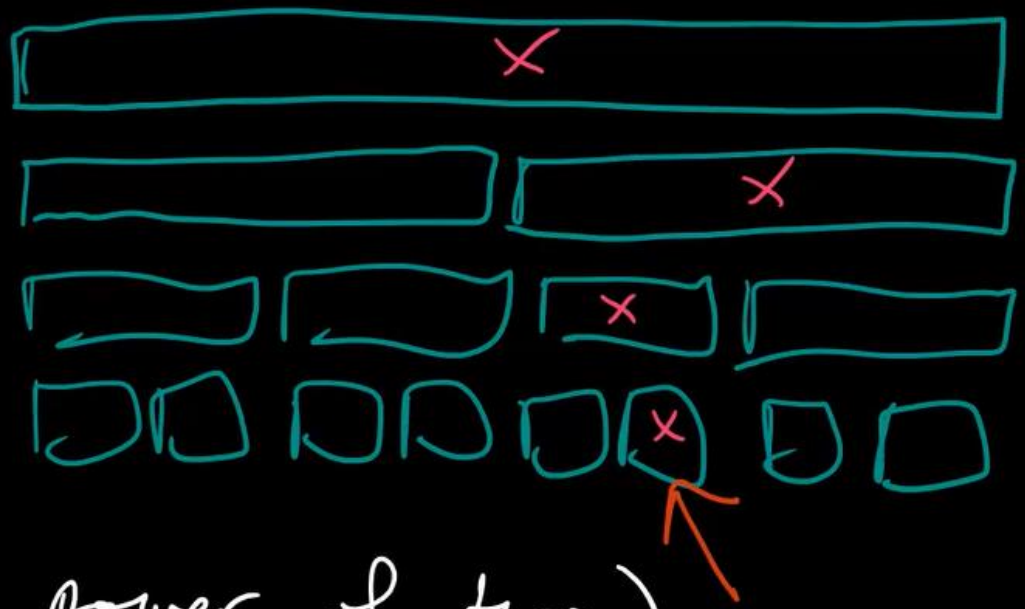
Can we get fewer dependencies.



Divide & Conquer

Idea

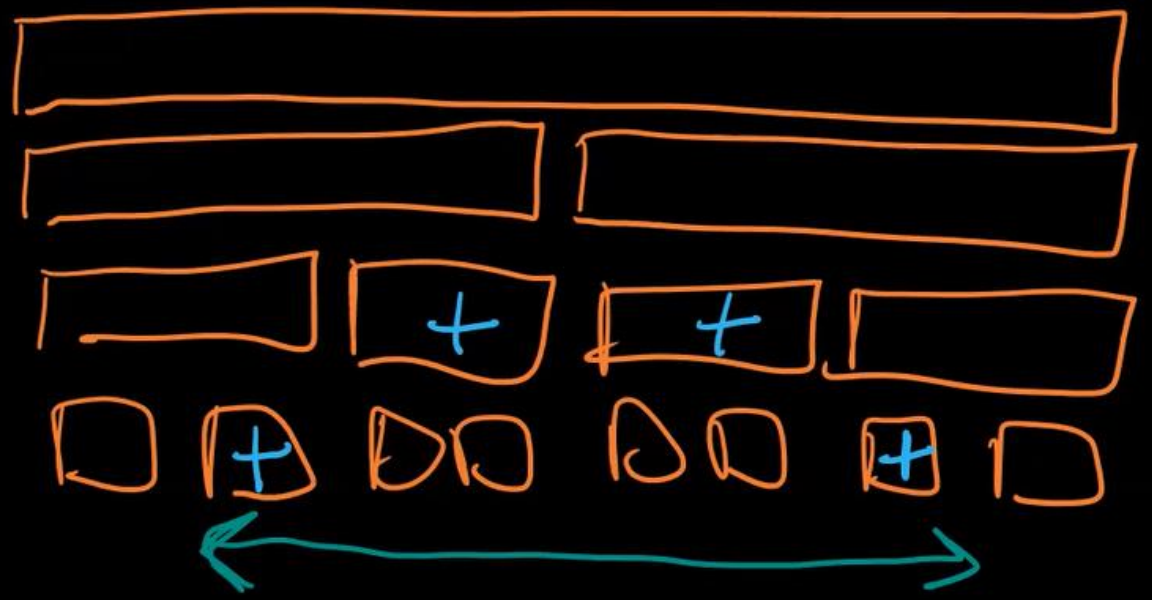
Use a tree (D&C tree)



(Assume n is power of two)

Update: Update $\log n$ ancestors

Query



Lemma Few blocks

At most 2 blocks per level to make $[i, j)$

Assume 3 adjacent blocks on one level

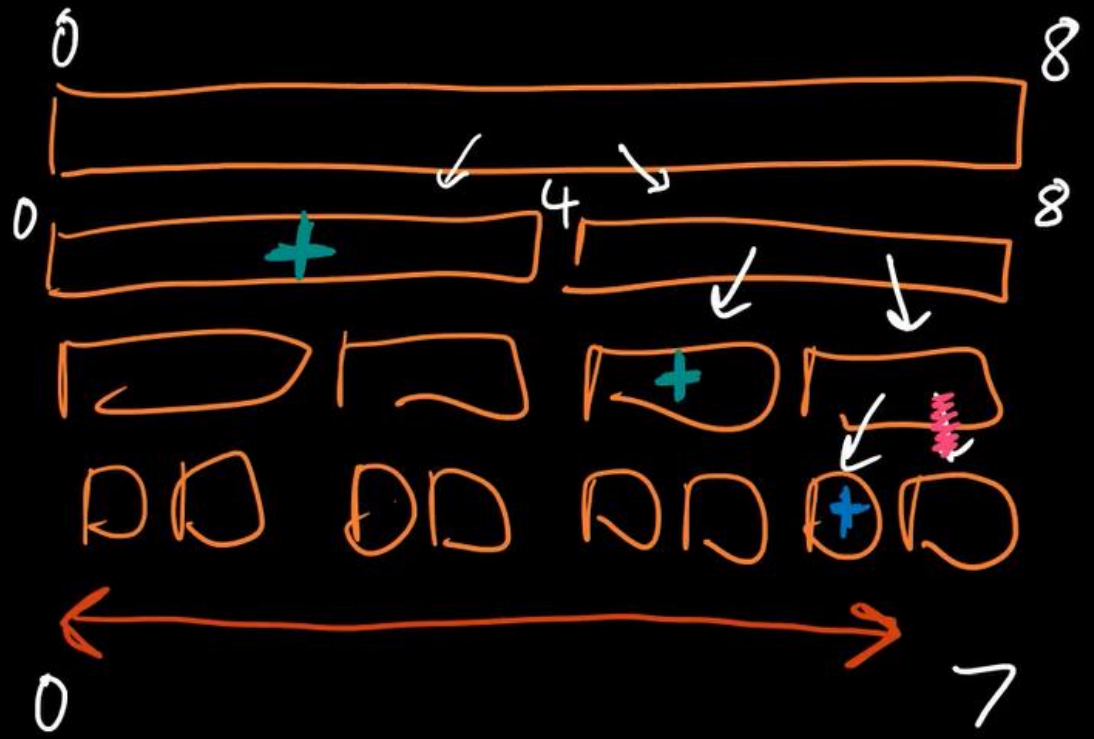
Assume 3 non-adjacent blocks

Corrollary

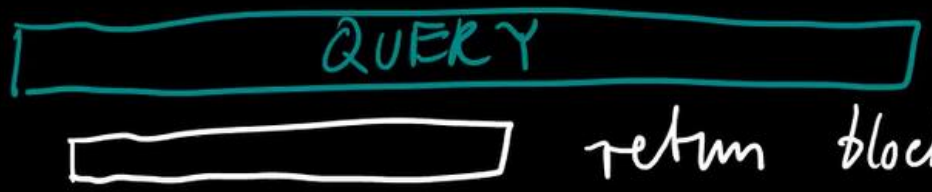
At most $2 \log_2 n$ block to make $[i, j)$

Queries (RangeSum (i, j)) in $O(\log_2 n)$ time.

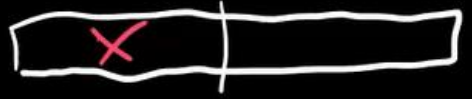
Algorithm



Query in O(log n)



return block sum



only recurse right

recurse both sides

Assign : $O(\log n)$

RangeSum $O(\log n)$

Implementation

Take advantage of perfect binary tree

"Heap" trick. Store array of nodes

root = node 0

left child $(i) = 2i + 1$

right child $(i) = 2i + 2$

parent $(i) = \lfloor (i-1)/2 \rfloor$

Don't need to store pointers!

2n nodes

Speeding up Algorithms

Example: Inversion count

Ordered sequence $a_0, \ldots a_{n-1}$

"Inversion": $(i < j)$ s.t. $a_i > a_j$

Algorithm

Check all pairs in $O(n^2)$ time

For each j: count $i < j$ with $a_i > a_j$

Counting things bigger than a_j is a range query!

```
n = size(a)
counts = SegTree(n, 0)
result = 0
for j = 0 to n-1 do {
    result = result + counts.RangeSum(a_j, n)
    counts.Assign(a_j, 1)
}
return result
```

$O(n \log n)$

Extensions

- Any associative reduction (min, max, sum)

Adding to a value

Add (i, x) : $a_i = a_i + x$

: Assign $(i$, RangeSum $(i, i+1) + x)$

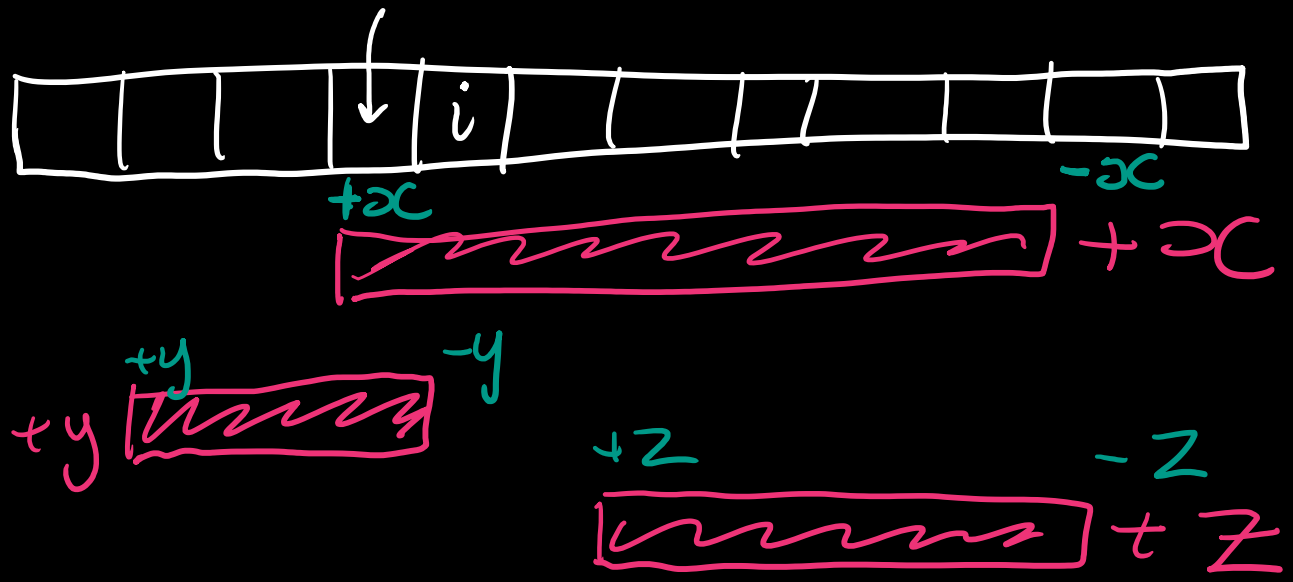
Flip the operations

Opposite API

RangeAdd (i, j, x): Add x to $a_i, \ldots a_{j-1}$

Get (i) : Returns a_i

Reduce to regular SegTrees



RangeAdd(i, j, x): Add(i, x); Add(j, -x)

Get(i): Return RangeSum(0, i+1)

O(log n) time