Hasso Plattner Institut
Digital Engineering · Universität Potsdam

MOIA

KIT
Karlsruhe Institute of Technology

Algorithm Bootcamp

July 2024

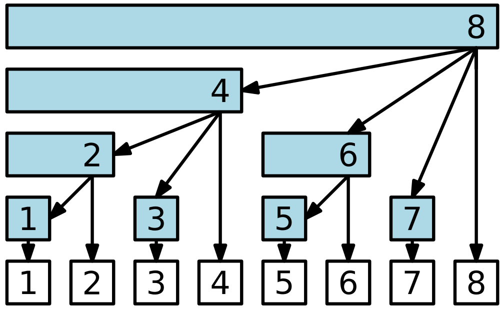
Day 4: Data Structures

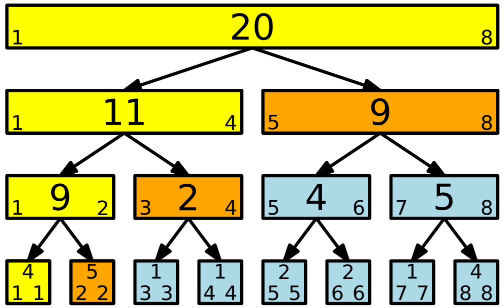
Dr. Christopher Weyand
Optimization Expert
Fleet Optimization

David Stangl
Software Engineer
Fleet Optimization

Lectures on data structures

- today
  - prefix sums/differences
  - square root decomposition
  - segment trees
- not today :)
  - sparse tables
  - minimum queues
  - fenwick trees
  - treaps





Problem we want to solve

- we have an array A of length n containing some values

Problem we want to solve

- we have an array A of length n containing some values
- we want to support

Problem we want to solve

- we have an array A of length n containing some values
- we want to support
  - queries on some aggregate over ranges of the values

Problem we want to solve

- we have an array A of length n containing some values
- we want to support
  - queries on some aggregate over ranges of the values
  - updates on the values

Problem we want to solve

- we have an array A of length n containing some values
- we want to support
  - queries on some aggregate over ranges of the values
  - updates on the values
- we want it to be fast

Problem we want to solve

- we have an array A of length n containing some values
- we want to support
  - queries on some aggregate over ranges of the values
  - updates on the values
- we want it to be fast

- aggregates:

Problem we want to solve

- we have an array A of length n containing some values
- we want to support
  - queries on some aggregate over ranges of the values
  - updates on the values
- we want it to be fast

- aggregates:
  - sum (our example for today)

Problem we want to solve

- we have an array A of length n containing some values
- we want to support
  - queries on some aggregate over ranges of the values
  - updates on the values
- we want it to be fast

- aggregates:
  - sum (our example for today)
  - product, min, max, gcd, ...

Problem we want to solve

- we have an array A of length n containing some values
- we want to support
  - queries on some aggregate over ranges of the values
  - updates on the values
- we want it to be fast

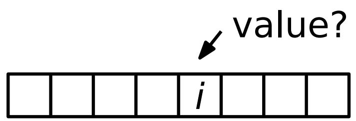
- aggregates:
  - sum (our example for today)
  - product, min, max, gcd, ...
  - even really crazy things: set of unique values, convex hull of points, ...

Queries/updates

- queries

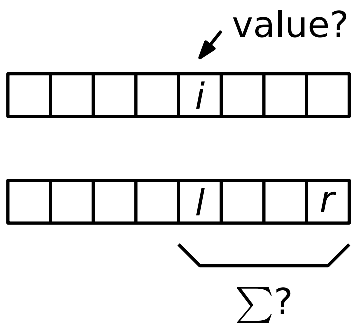
Queries/updates

- queries
  - *point query*: get the value of $A[i]$



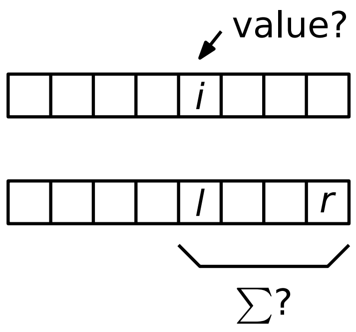
Queries/updates

- queries
  - *point query*: get the value of $A[i]$
  - *range query*: get sum of values from $A[l]$ to $A[r]$



Queries/updates

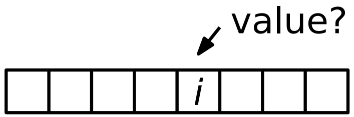
- queries
  - *point query*: get the value of $A[i]$
  - *range query*: get sum of values from $A[l]$ to $A[r]$



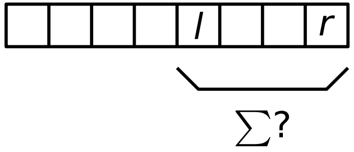
- updates

Queries/updates

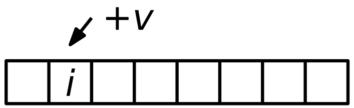
- queries
  - *point query*: get the value of $A[i]$



  - *range query*: get sum of values from $A[l]$ to $A[r]$

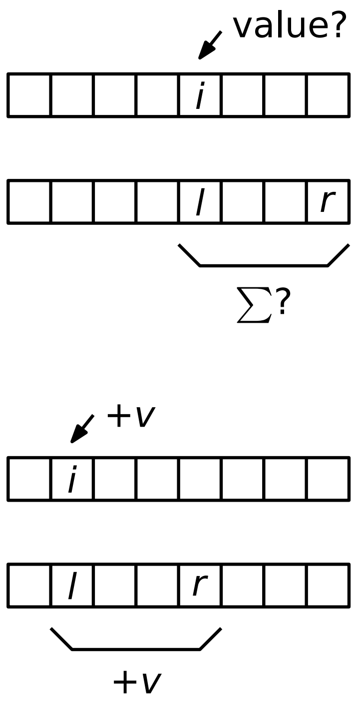


- updates
  - *point update*: add v to $A[i]$



Queries/updates

- queries
  - *point query*: get the value of $A[i]$
  - *range query*: get sum of values from $A[l]$ to $A[r]$
- updates
  - *point update*: add v to $A[i]$
  - *range update*: add v to every value between $A[l]$ and $A[r]$



Input array

input array	1	5	2	-2	4	8	-5	6

Input array

	query		update	
	point	range	point	range
input array				

1	5	2	-2	4	8	-5	6

Input array

	query		update	
	point	range	point	range
input array	$\mathcal{O}(1)$			

1	5	2	-2	4	8	-5	6

Input array

	query		update	
	point	range	point	range
input array	$\mathcal{O}(1)$	$\mathcal{O}(n)$		

1	5	2	-2	4	8	-5	6

Input array

	query		update	
	point	range	point	range
input array	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	

1	5	2	-2	4	8	-5	6

Input array

		query		update	
		point	range	point	range
input array	1, 5, 2, -2, 4, 8, -5, 6	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$

Input array

	1	5	2	-2	4	8	-5	6	query: point	query: range	update: point	update: range
input array	1	5	2	-2	4	8	-5	6	$\mathcal{O}(1)$	$-$	$\mathcal{O}(1)$	$-$

Prefix sums

		query		update	
		point	range	point	range
input array	1 | 5 | 2 | -2 | 4 | 8 | -5 | 6	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–

Prefix sums

									query		update	
									point	range	point	range
input array	1	5	2	-2	4	8	-5	6	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array												

Prefix sums

	query		update	
	point	range	point	range
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–

input array

1	5	2	-2	4	8	-5	6

prefix array

$P[i] = \sum_{j=1}^{i} A[j]$

Prefix sums

	query		update	
	point	range	point	range
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–

input array

1	5	2	-2	4	8	-5	6

prefix array

$P[i] = \sum_{j=1}^{i} A[j]$

1							

Prefix sums

	query		update	
	point	range	point	range
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–

input array

1	5	2	-2	4	8	-5	6

prefix array

$P[i] = \sum_{j=1}^{i} A[j]$

1	6						

Prefix sums

	query		update	
	point	range	point	range
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–

input array

1	5	2	-2	4	8	-5	6

prefix array

$P[i] = \sum_{j=1}^{i} A[j]$

1	6	8					

Prefix sums

	query		update	
	point	range	point	range
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–

input array	1	5	2	-2	4	8	-5	6
prefix array	1	6	8	6	10	18	13	19

$P[i] = \sum_{j=1}^{i} A[j]$

Prefix sums

		query: point	query: range	update: point	update: range
input array	1, 5, 2, -2, 4, 8, -5, 6	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	1, 6, 8, 6, 10, 18, 13, 19		$\mathcal{O}(1)$		

Prefix sums

	array	query: point	query: range	update: point	update: range
input array	$1 \mid 5 \mid \overbrace{2 \mid -2 \mid 4 \mid 8 \mid -5}^{A} \mid 6$	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	$1 \mid 6 \mid 8 \mid 6 \mid 10 \mid 18 \mid 13 \mid 19$		$\mathcal{O}(1)$		

Prefix sums

	query		update	
	point	range	point	range
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$		$\mathcal{O}(1)$		

input array:

1	5	2	-2	4	8	-5	6

prefix array:

1	6	8	6	10	18	13	19

Prefix sums

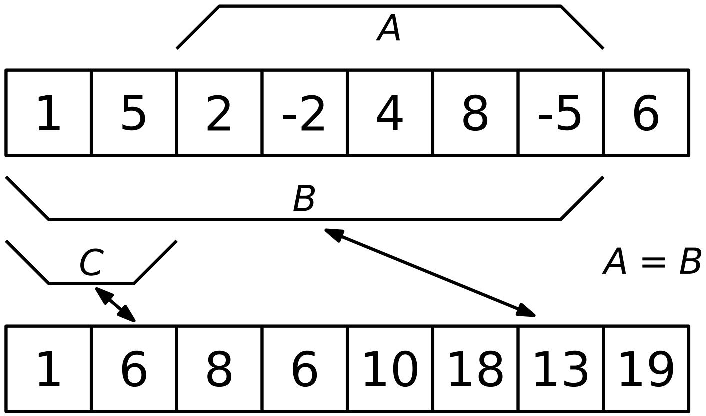
			query		update	
			point	range	point	range
input array	1 | 5 | 2 | -2 | 4 | 8 | -5 | 6		$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	1 | 6 | 8 | 6 | 10 | 18 | 13 | 19			$\mathcal{O}(1)$		

A (elements 3 to 7), B (elements 1 to 7), C (elements 1 to 2)

$A = B - C$

Prefix sums

	query		update	
	point	range	point	range
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$		$\mathcal{O}(1)$		



$A = B - C$

Prefix sums

		query		update	
		point	range	point	range
input array	1, 5, 2, -2, 4, 8, -5, 6	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	1, 6, 8, 6, 10, 18, 13, 19		$\mathcal{O}(1)$		

Prefix sums

	query		update	
	point	range	point	range
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	$\mathcal{O}(1)$	$\mathcal{O}(1)$		

input array

1	5	2	-2	4	8	-5	6

prefix array

$P[i] = \sum_{j=1}^{i} A[j]$

1	6	8	6	10	18	13	19

Prefix sums

		query		update	
		point	range	point	range
input array		$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$		$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

input array:

1	5	2	-2	4	8	-5	6

prefix array:

1	6	8	6	10	18	13	19

Prefix sums

		query		update	
		point	range	point	range
input array		1 | 5 | 2 | -2 | 4 | 8 | -5 | 6	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–

prefix sum

prefix array $P[i] = \sum_{j=1}^{i} A[j]$	1 | 6 | 8 | 6 | 10 | 18 | 13 | 19	$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

Prefix sums

									query: point	query: range	update: point	update: range
input array	1	5	2	-2	4	8	-5	6	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	1	6	8	6	10	18	13	19	$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

diff

prefix sum

Difference array

									query: point	query: range	update: point	update: range
input array	1	5	2	-2	4	8	-5	6	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	1	6	8	6	10	18	13	19	$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

diff

prefix sum

Difference array

difference array								
input array	1	5	2	-2	4	8	-5	6
prefix array	1	6	8	6	10	18	13	19

prefix array $\xrightarrow{\text{diff}}$ input array

input array $\xrightarrow{\text{prefix sum}}$ prefix array

$P[i] = \sum_{j=1}^{i} A[j]$

	query		update	
	point	range	point	range
difference array				
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array	$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

Difference array

		query		update	
		point	range	point	range
difference array $D[i] = A[i] - A[i-1]$	[, , , , , , ,]				
input array	[1, 5, 2, -2, 4, 8, -5, 6]	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	[1, 6, 8, 6, 10, 18, 13, 19]	$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

diff prefix sum

Difference array

									query point	query range	update point	update range
difference array $D[i] = A[i] - A[i-1]$	1											
input array	1	5	2	-2	4	8	-5	6	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	1	6	8	6	10	18	13	19	$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

diff

prefix sum

Difference array

									query point	query range	update point	update range
difference array $D[i] = A[i] - A[i-1]$	1	4										
input array	1	5	2	-2	4	8	-5	6	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	1	6	8	6	10	18	13	19	$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

diff

prefix sum

Difference array

									query		update	
									point	range	point	range
difference array $D[i] = A[i] - A[i-1]$	1	4	-3									
input array	1	5	2	-2	4	8	-5	6	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	1	6	8	6	10	18	13	19	$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

diff

prefix sum

Difference array

difference array

$D[i] = A[i] - A[i-1]$

1	4	-3	-4	6	4	-13	11

input array

1	5	2	-2	4	8	-5	6

diff

prefix sum

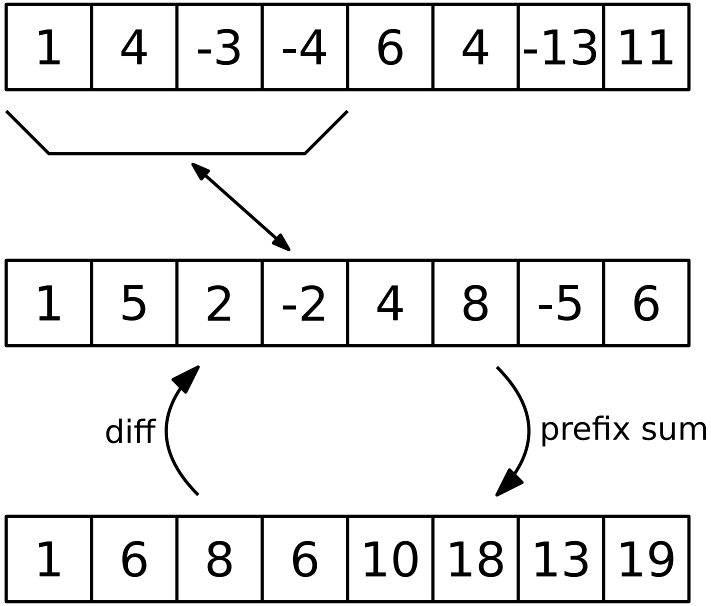
prefix array

$P[i] = \sum_{j=1}^{i} A[j]$

1	6	8	6	10	18	13	19

	query		update	
	point	range	point	range
difference array				
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array	$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

Difference array



	query		update	
	point	range	point	range
difference array $D[i] = A[i] - A[i-1]$				
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

Difference array

										query point	query range	update point	update range
difference array $D[i] = A[i] - A[i-1]$	1	4	-3	-4	6	4	-13	11					$\mathcal{O}(1)$
input array	1	5	2	-2	4	8	-5	6		$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	1	6	8	6	10	18	13	19		$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

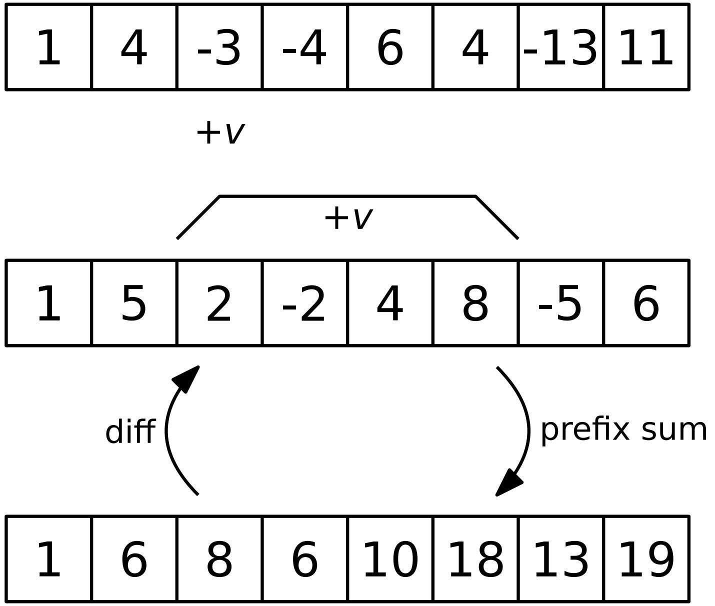
diff

prefix sum

Difference array

		query		update	
		point	range	point	range
difference array $D[i] = A[i] - A[i-1]$	1 | 4 | -3 | -4 | 6 | 4 | -13 | 11				$\mathcal{O}(1)$
	$+v$				
input array	1 | 5 | 2 | -2 | 4 | 8 | -5 | 6	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
	diff ↑ ↓ prefix sum				
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	1 | 6 | 8 | 6 | 10 | 18 | 13 | 19	$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

Difference array



	query point	query range	update point	update range
difference array $D[i] = A[i] - A[i-1]$				$\mathcal{O}(1)$
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

Difference array

	query point	query range	update point	update range
difference array $D[i] = A[i] - A[i-1]$				$\mathcal{O}(1)$
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

difference array

1	4	-3	-4	6	4	-13	11
		$+v$				$-v$	

input array

$+v$

1	5	2	-2	4	8	-5	6

diff

prefix sum

prefix array

1	6	8	6	10	18	13	19

Difference array

	array	query point	query range	update point	update range
difference array $D[i] = A[i] - A[i-1]$	1, 4, -3, -4, 6, 4, -13, 11				$\mathcal{O}(1)$
input array	1, 5, 2, -2, 4, 8, -5, 6	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	1, 6, 8, 6, 10, 18, 13, 19	$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

diff

prefix sum

Difference array

	array	query: point	query: range	update: point	update: range
difference array $D[i] = A[i] - A[i-1]$	1, 4, -3, -4, 6, 4, -13, 11			$\mathcal{O}(1)$	$\mathcal{O}(1)$
input array	1, 5, 2, -2, 4, 8, -5, 6	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	1, 6, 8, 6, 10, 18, 13, 19	$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

diff

prefix sum

Difference array

		query		update	
		point	range	point	range
difference array $D[i] = A[i] - A[i-1]$	1	4	-3	-4	6	4	-13	11	–	–	$\mathcal{O}(1)$	$\mathcal{O}(1)$
input array	1	5	2	-2	4	8	-5	6	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	1	6	8	6	10	18	13	19	$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

diff

prefix sum

Difference array

		query point	query range	update point	update range
difference array $D[i] = A[i] - A[i-1]$	1 | 4 | -3 | -4 | 6 | 4 | -13 | 11	–	–	$\mathcal{O}(1)$	$\mathcal{O}(1)$
input array	1 | 5 | 2 | -2 | 4 | 8 | -5 | 6	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	1 | 6 | 8 | 6 | 10 | 18 | 13 | 19	$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

$\text{input array} \xrightarrow{\text{diff}} \text{difference array}$

$\text{prefix array} \xrightarrow{\text{diff}} \text{input array}$

$\text{input array} \xrightarrow{\text{prefix sum}} \text{prefix array}$

Difference array

			query		update	
			point	range	point	range
difference array $D[i] = A[i] - A[i-1]$	1 4 -3 -4 6 4 -13 11		–	–	$\mathcal{O}(1)$	$\mathcal{O}(1)$
input array	1 5 2 -2 4 8 -5 6		$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	1 6 8 6 10 18 13 19		$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

$$\text{prefix array} \xrightarrow{\text{diff}} \text{input array} \xrightarrow{\text{diff}} \text{difference array}$$

$$\text{difference array} \xrightarrow{\text{prefix sum}} \text{input array} \xrightarrow{\text{prefix sum}} \text{prefix array}$$

Arrays — Overview

difference array

$D[i] = A[i] - A[i-1]$

1	4	-3	-4	6	4	-13	11

diff ↑ ↓ prefix sum

input array

1	5	2	-2	4	8	-5	6

diff ↑ ↓ prefix sum

prefix array

$P[i] = \sum_{j=1}^{i} A[j]$

1	6	8	6	10	18	13	19

	query point	query range	update point	update range
difference array	–	–	$\mathcal{O}(1)$	$\mathcal{O}(1)$
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array	$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

Arrays — Overview

		query		update	
		point	range	point	range
difference array $D[i] = A[i] - A[i-1]$	1, 4, -3, -4, 6, 4, -13, 11	–	–	$\mathcal{O}(1)$	$\mathcal{O}(1)$
	diff $\mathcal{O}(n)$ ↑ / $\mathcal{O}(n)$ prefix sum ↓				
input array	1, 5, 2, -2, 4, 8, -5, 6	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
	diff $\mathcal{O}(n)$ ↑ / $\mathcal{O}(n)$ prefix sum ↓				
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	1, 6, 8, 6, 10, 18, 13, 19	$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	–

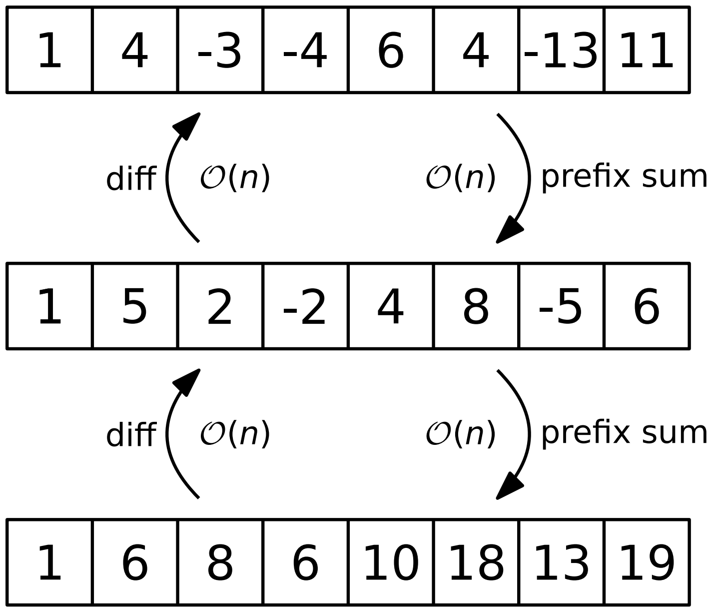
Arrays — Overview

		query point	query range	update point	update range
difference array $D[i] = A[i] - A[i-1]$	1 | 4 | -3 | -4 | 6 | 4 | -13 | 11	–	–	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$
	diff $\mathcal{O}(n)$ ↑ ↓ $\mathcal{O}(n)$ prefix sum				
input array	1 | 5 | 2 | -2 | 4 | 8 | -5 | 6	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
	diff $\mathcal{O}(n)$ ↑ ↓ $\mathcal{O}(n)$ prefix sum				
prefix array $P[i] = \sum_{j=1}^{i} A[j]$	1 | 6 | 8 | 6 | 10 | 18 | 13 | 19	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$	–	–

$*$: requires invertible operation

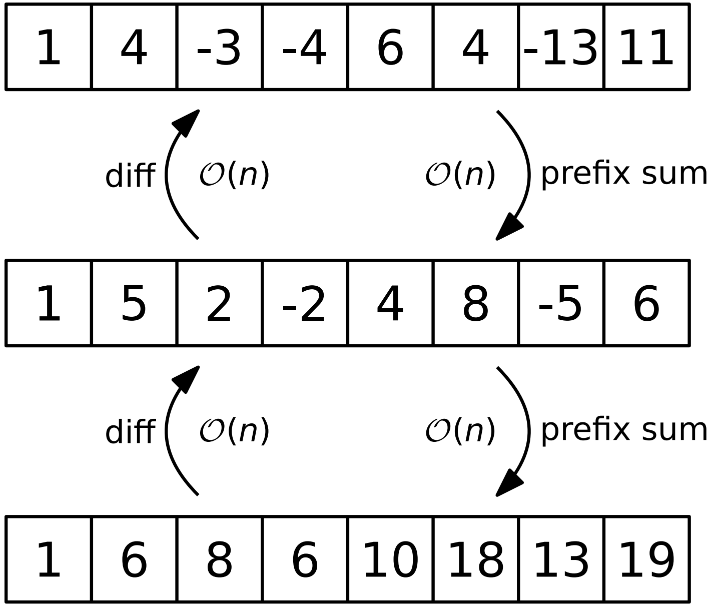
Arrays — Conclusion

- pros/cons



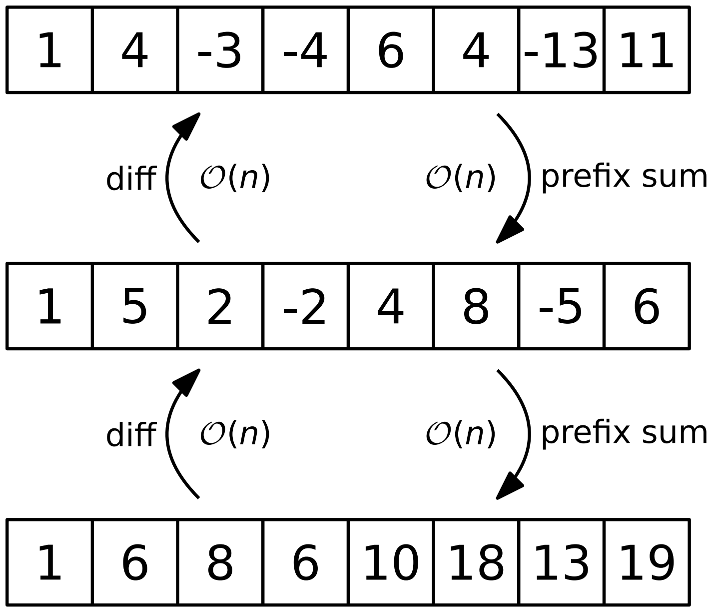
Arrays — Conclusion

- pros/cons
  - + any operation possible in $\mathcal{O}(1)$



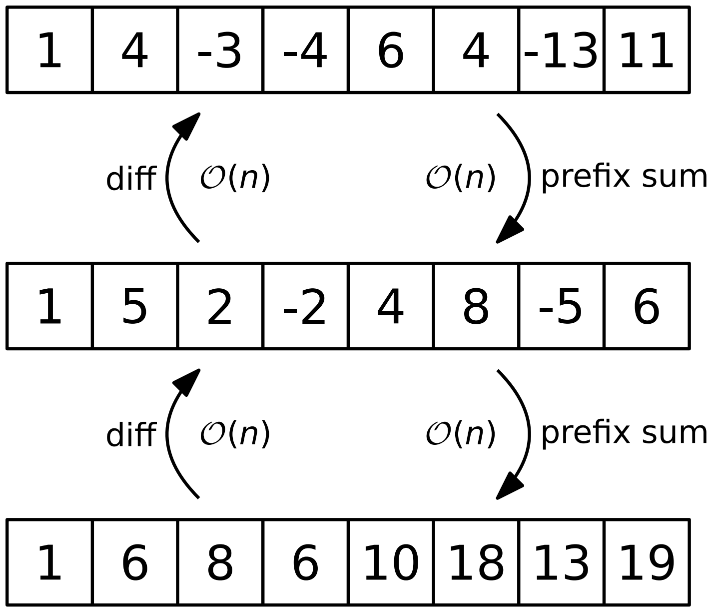
Arrays — Conclusion

- pros/cons
  - \+ any operation possible in $\mathcal{O}(1)$
  - \+ trivial to implement



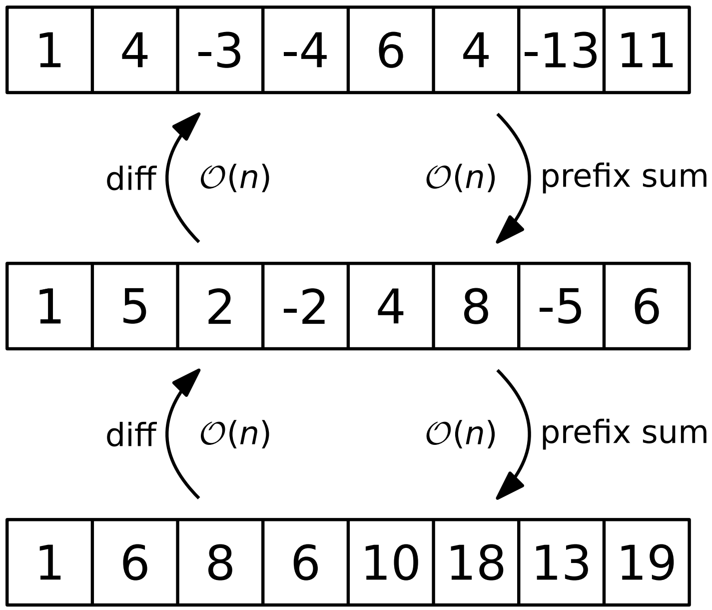
Arrays — Conclusion

- pros/cons
  - \+ any operation possible in $\mathcal{O}(1)$
  - \+ trivial to implement
  - – requires $\mathcal{O}(n)$ conversions



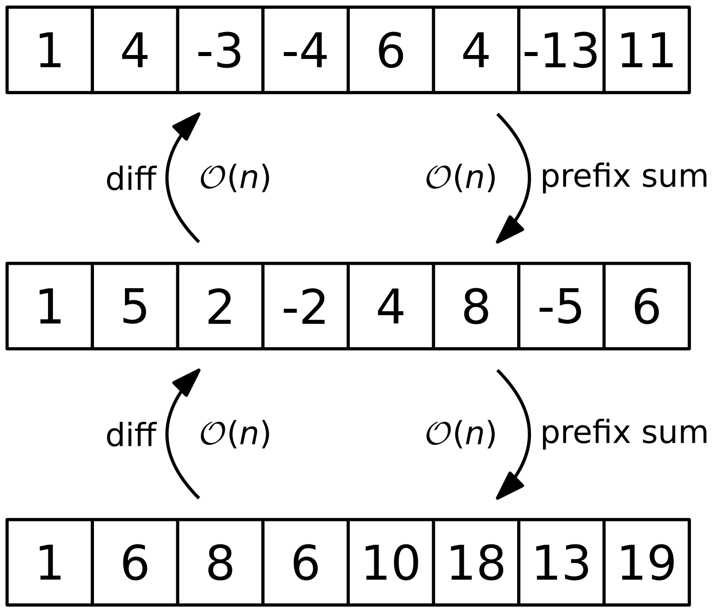
Arrays — Conclusion

- pros/cons
  - \+ any operation possible in $\mathcal{O}(1)$
  - \+ trivial to implement
  - – requires $\mathcal{O}(n)$ conversions
  - – requires invertible operation



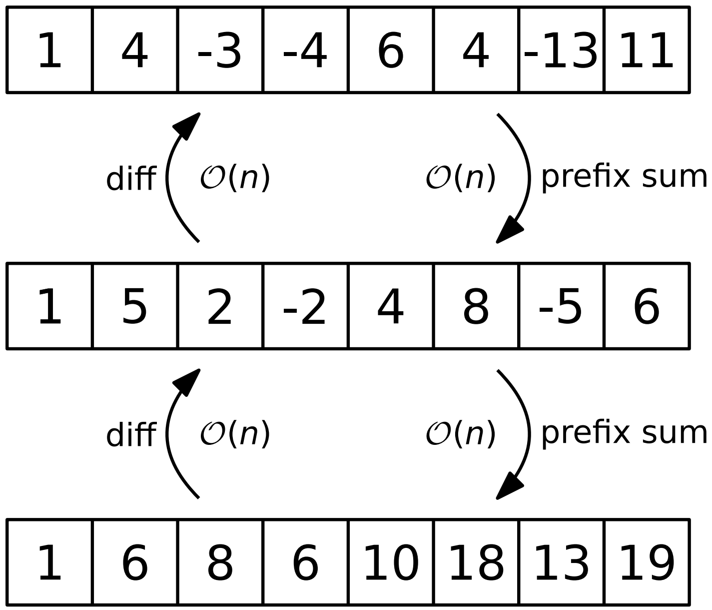
Arrays — Conclusion

- pros/cons
  - \+ any operation possible in $\mathcal{O}(1)$
  - \+ trivial to implement
  - – requires $\mathcal{O}(n)$ conversions
  - – requires invertible operation
- good when



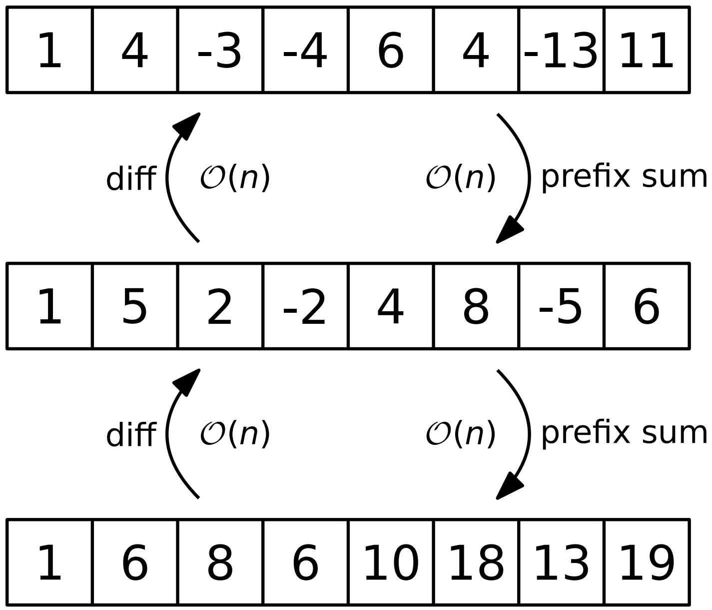
Arrays — Conclusion

- pros/cons
  - \+ any operation possible in $\mathcal{O}(1)$
  - \+ trivial to implement
  - – requires $\mathcal{O}(n)$ conversions
  - – requires invertible operation
- good when
  - queries & updates are separated



Arrays — Conclusion

- pros/cons
  - \+ any operation possible in $\mathcal{O}(1)$
  - \+ trivial to implement
  - – requires $\mathcal{O}(n)$ conversions
  - – requires invertible operation
- good when
  - queries & updates are separated
  - only queries/only updates



Square root decomposition

- let's try to find a middle ground between $\mathcal{O}(1)$ and $\mathcal{O}(n)$: $\mathcal{O}\left(\sqrt{n}\right)$

Square root decomposition

- let's try to find a middle ground between $\mathcal{O}(1)$ and $\mathcal{O}(n)$: $\mathcal{O}\left(\sqrt{n}\right)$

7	0	1	-5	8	-2	-9	6	-2	9	-4	-6	8	2	-8	-1

Square root decomposition

- let's try to find a middle ground between $\mathcal{O}(1)$ and $\mathcal{O}(n)$: $\mathcal{O}\left(\sqrt{n}\right)$
- group array into blocks of size $\sqrt{n}$

7	0	1	-5	8	-2	-9	6	-2	9	-4	-6	8	2	-8	-1

Square root decomposition

- let's try to find a middle ground between $\mathcal{O}(1)$ and $\mathcal{O}(n)$: $\mathcal{O}\left(\sqrt{n}\right)$
- group array into blocks of size $\sqrt{n}$

7	0	1	-5	8	-2	-9	6	-2	9	-4	-6	8	2	-8	-1

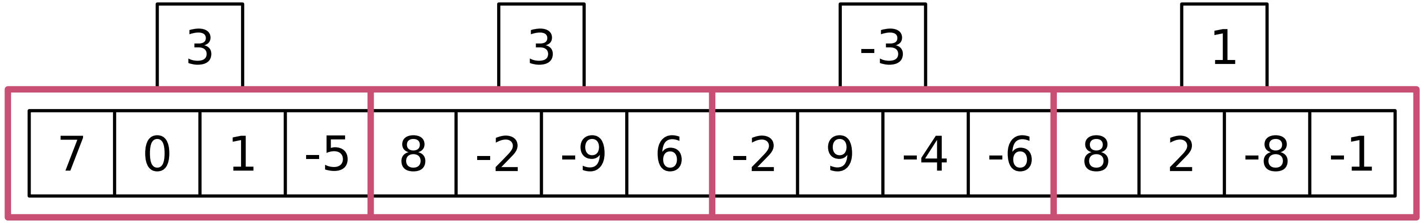
Square root decomposition

- let's try to find a middle ground between $\mathcal{O}(1)$ and $\mathcal{O}(n)$: $\mathcal{O}\left(\sqrt{n}\right)$
- group array into blocks of size $\sqrt{n}$
  - calculate the sum for each block

7	0	1	-5	8	-2	-9	6	-2	9	-4	-6	8	2	-8	-1

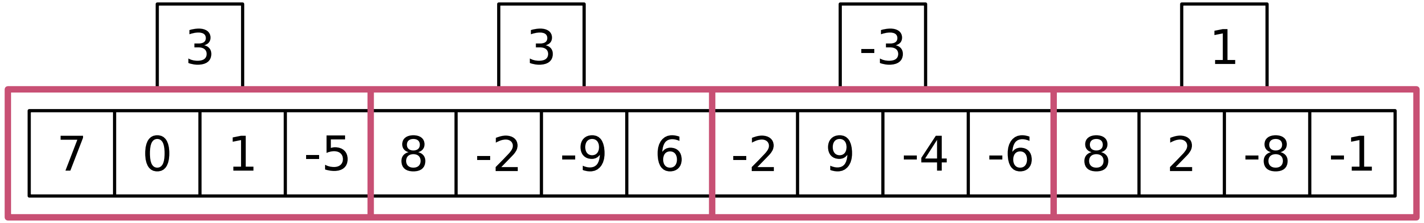
Square root decomposition

- let’s try to find a middle ground between $\mathcal{O}(1)$ and $\mathcal{O}(n)$: $\mathcal{O}\left(\sqrt{n}\right)$
- group array into blocks of size $\sqrt{n}$
  - calculate the sum for each block



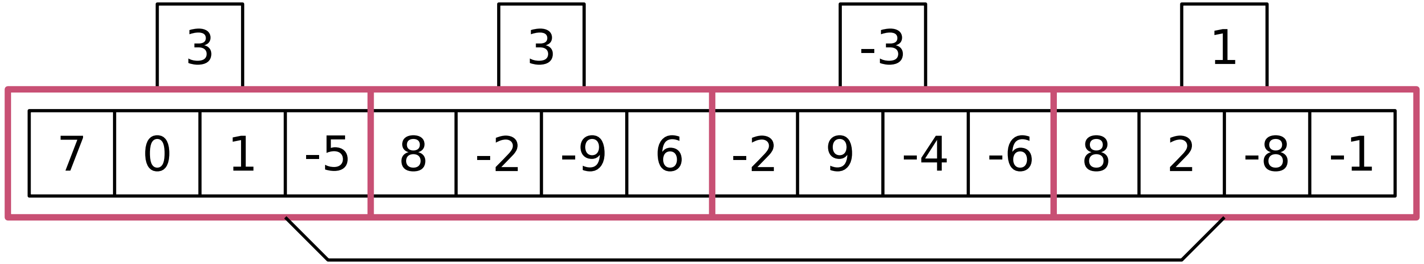
Square root decomposition

- let's try to find a middle ground between $\mathcal{O}(1)$ and $\mathcal{O}(n)$: $\mathcal{O}\left(\sqrt{n}\right)$
- group array into blocks of size $\sqrt{n}$
  - calculate the sum for each block
- range query: use block sum for fully covered blocks $\Rightarrow \mathcal{O}(\sqrt{n})$



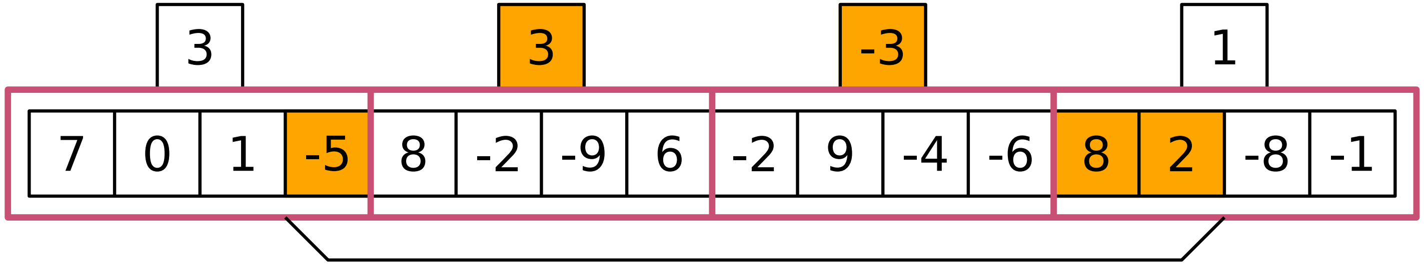
Square root decomposition

- let's try to find a middle ground between $\mathcal{O}(1)$ and $\mathcal{O}(n)$: $\mathcal{O}\left(\sqrt{n}\right)$
- group array into blocks of size $\sqrt{n}$
  - calculate the sum for each block
- range query: use block sum for fully covered blocks $\Rightarrow \mathcal{O}(\sqrt{n})$



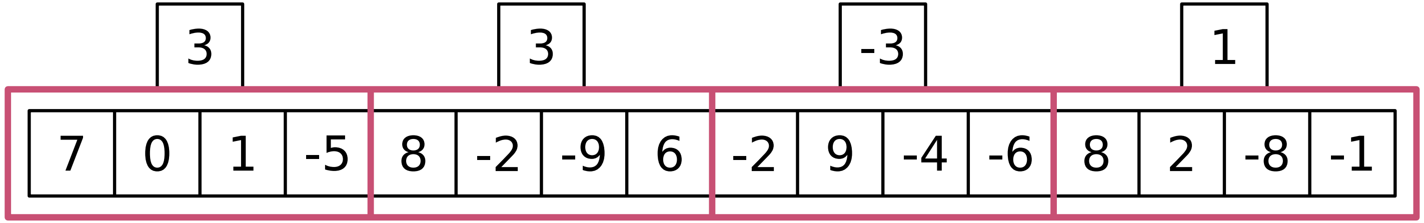
Square root decomposition

- let’s try to find a middle ground between $\mathcal{O}(1)$ and $\mathcal{O}(n)$: $\mathcal{O}\left(\sqrt{n}\right)$
- group array into blocks of size $\sqrt{n}$
  - calculate the sum for each block
- range query: use block sum for fully covered blocks $\Rightarrow \mathcal{O}(\sqrt{n})$



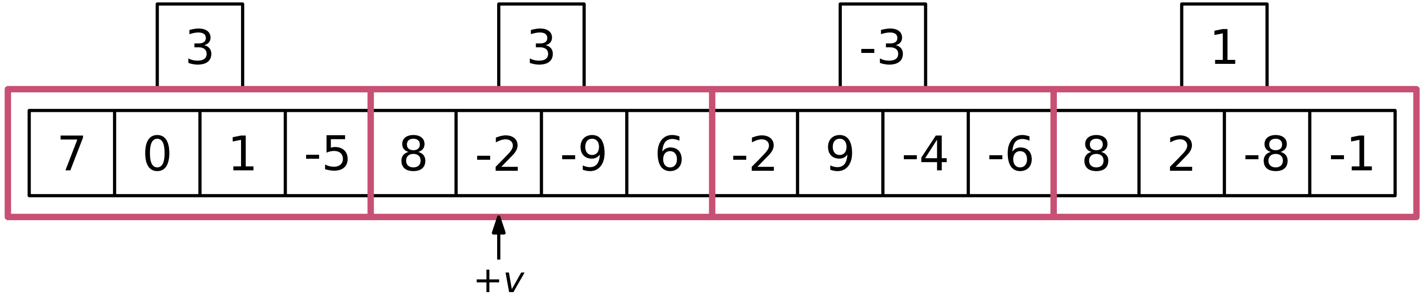
Square root decomposition

- let’s try to find a middle ground between $\mathcal{O}(1)$ and $\mathcal{O}(n)$: $\mathcal{O}\left(\sqrt{n}\right)$
- group array into blocks of size $\sqrt{n}$
  - calculate the sum for each block
- range query: use block sum for fully covered blocks $\Rightarrow \mathcal{O}(\sqrt{n})$
- point update: recalculate block sum $\Rightarrow \mathcal{O}(\sqrt{n})$



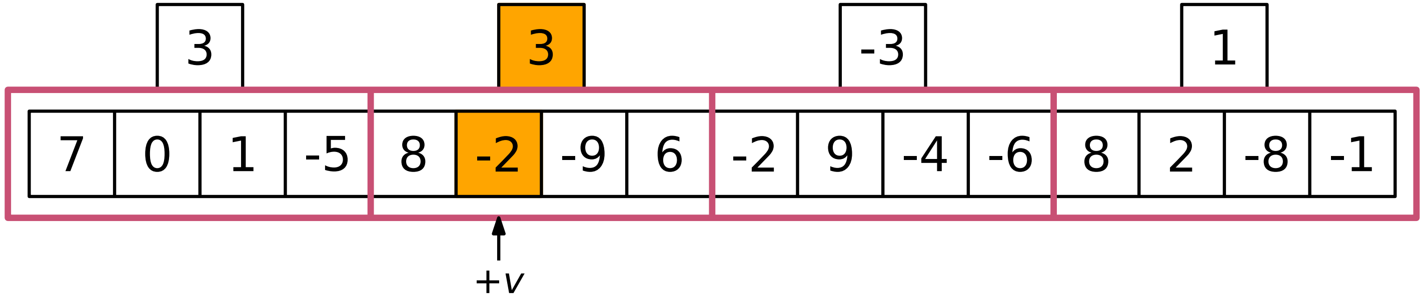
Square root decomposition

- let's try to find a middle ground between $\mathcal{O}(1)$ and $\mathcal{O}(n)$: $\mathcal{O}\left(\sqrt{n}\right)$
- group array into blocks of size $\sqrt{n}$
  - calculate the sum for each block
- range query: use block sum for fully covered blocks $\Rightarrow \mathcal{O}(\sqrt{n})$
- point update: recalculate block sum $\Rightarrow \mathcal{O}(\sqrt{n})$



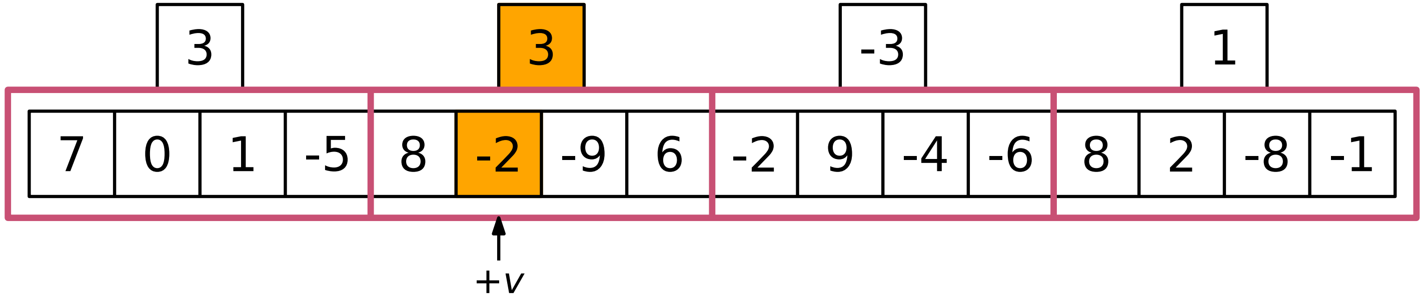
Square root decomposition

- let's try to find a middle ground between $\mathcal{O}(1)$ and $\mathcal{O}(n)$: $\mathcal{O}\left(\sqrt{n}\right)$
- group array into blocks of size $\sqrt{n}$
  - calculate the sum for each block
- range query: use block sum for fully covered blocks $\Rightarrow \mathcal{O}(\sqrt{n})$
- point update: recalculate block sum $\Rightarrow \mathcal{O}(\sqrt{n})$



Square root decomposition

- let's try to find a middle ground between $\mathcal{O}(1)$ and $\mathcal{O}(n)$: $\mathcal{O}\left(\sqrt{n}\right)$
- group array into blocks of size $\sqrt{n}$
  - calculate the sum for each block
- range query: use block sum for fully covered blocks $\Rightarrow \mathcal{O}(\sqrt{n})$
- point update: recalculate block sum $\Rightarrow \mathcal{O}(\sqrt{n})$
  - actually in $\mathcal{O}(1)$ in this case, but not always (range minimum, etc.)



Square root decomposition

Can we get range updates to work with sqrt decomposition (without any extra effort)? What do we give up for it?

Square root decomposition

Can we get range updates to work with sqrt decomposition (without any extra effort)? What do we give up for it?

- we can use the difference array as our base!

Square root decomposition

Can we get range updates to work with sqrt decomposition (without any extra effort)? What do we give up for it?

- we can use the difference array as our base!
- remember, on the difference array

Square root decomposition

Can we get range updates to work with sqrt decomposition (without any extra effort)? What do we give up for it?

- we can use the difference array as our base!
- remember, on the difference array
  - range update $\Rightarrow$ 2 point updates

Square root decomposition

Can we get range updates to work with sqrt decomposition (without any extra effort)? What do we give up for it?

- we can use the difference array as our base!
- remember, on the difference array
  - range update $\Rightarrow$ 2 point updates
  - point query $\Rightarrow$ prefix sum/range query

Square root decomposition

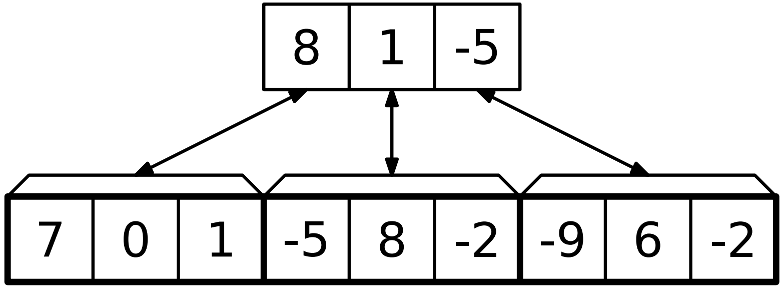
Can we get range updates to work with sqrt decomposition (without any extra effort)? What do we give up for it?

- we can use the difference array as our base!
- remember, on the difference array
  - range update $\Rightarrow$ 2 point updates
  - point query $\Rightarrow$ prefix sum/range query
- disadvantage: no more range queries

it is definitely possible to have both, but requires more effort and depends on the aggregate

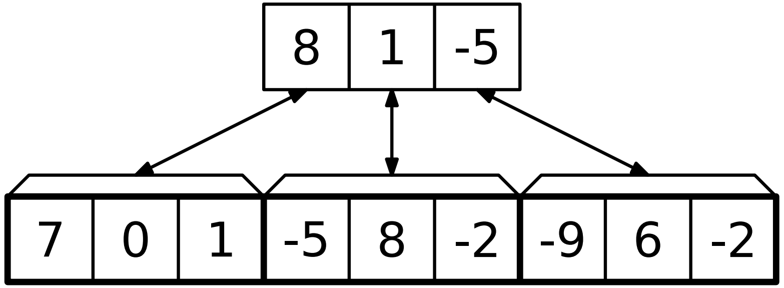
Square root decomposition — Conclusion

- pros/cons



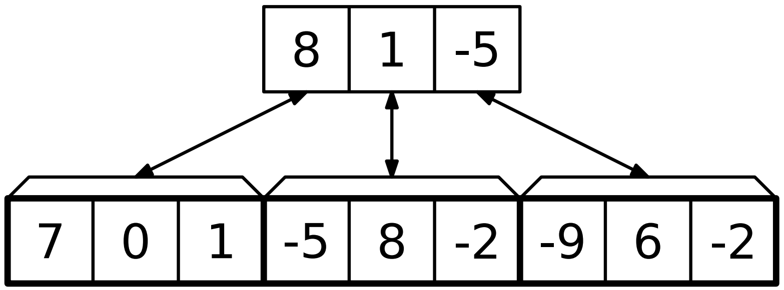
Square root decomposition — Conclusion

- pros/cons
  - \+ almost universally applicable



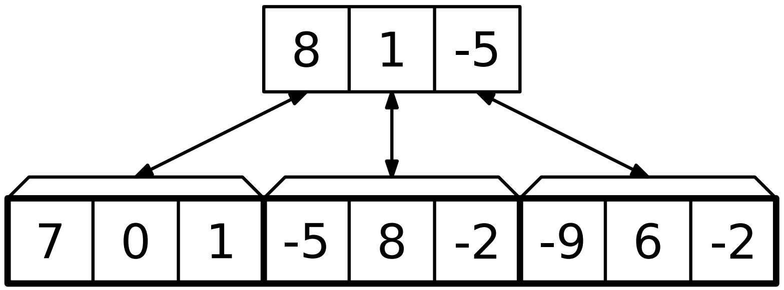
Square root decomposition — Conclusion

- pros/cons
  - + almost universally applicable
  - – no simultaneous range update/query



Square root decomposition — Conclusion

- pros/cons
  - + almost universally applicable
  - – no simultaneous range update/query
  - – $\mathcal{O}(\sqrt{n})$ rarely fast enough (if $\mathcal{O}(n)$ too slow)



Overview

	query		update	
	point	range	point	range
difference array†	–	–	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$	–	–
sqrt decomposition	$\mathcal{O}(1)$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(\sqrt{n})$	–

*: requires invertible operation

Overview

	query		update	
	point	range	point	range
difference array†	–	–	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$	–	–
sqrt decomposition	$\mathcal{O}(1)$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(\sqrt{n})$	–

*: requires invertible operation
†: as base for other DS, enables range updates at the price of range queries

Overview

	query		update	
	point	range	point	range
difference array[†]	–	–	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$	–	–
sqrt decomposition	$\mathcal{O}(1)$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(\sqrt{n})$	–
sqrt decomposition (on differences)	$\mathcal{O}(\sqrt{n})$	–	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(\sqrt{n})$

$*$: requires invertible operation

$\dagger$: as base for other DS, enables range updates at the price of range queries

Overview

	query		update	
	point	range	point	range
difference array†	–	–	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$	–	–
sqrt decomposition	$\mathcal{O}(1)$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(\sqrt{n})$	–

*: requires invertible operation
†: as base for other DS, enables range updates at the price of range queries

Overview

	query		update	
	point	range	point	range
difference array[†]	–	–	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$	–	–
sqrt decomposition	$\mathcal{O}(1)$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(\sqrt{n})$	–
segment tree	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)^{\S}$

*: requires invertible operation
†: as base for other DS, enables range updates at the price of range queries
§: requires lazy propagation

5 minute break

You have $n \leq 2 \cdot 10^5$ products, each identified by its price p_i and quality q_i ($1 \leq p_i, q_i \leq 10^9, p_i \neq p_j, q_i \neq q_j$). For each product, find the number of strictly worse products (higher price *and* lower quality)

assume you have a segment tree with everything in $O(\log n)$

5 minute break

You have $n \leq 2 \cdot 10^5$ products, each identified by its price p_i and quality q_i ($1 \leq p_i, q_i \leq 10^9, p_i \neq p_j, q_i \neq q_j$). For each product, find the number of strictly worse products (higher price *and* lower quality)

assume you have a segment tree with everything in $O(\log n)$

- use coordinate compression (price/quality index instead of value)

5 minute break

You have $n \leq 2 \cdot 10^5$ products, each identified by its price p_i and quality q_i ($1 \leq p_i, q_i \leq 10^9, p_i \neq p_j, q_i \neq q_j$). For each product, find the number of strictly worse products (higher price *and* lower quality)

assume you have a segment tree with everything in $O(\log n)$

- use coordinate compression (price/quality index instead of value)
- use segment tree, insert in quality order (low to high), count products with higher price

Segment trees

- binary tree where each node corresponds to a segment $[l, r]$

Segment trees

- binary tree where each node corresponds to a segment $[l, r]$
- root corresponds to $[1, n]$

Segment trees

- binary tree where each node corresponds to a segment $[l, r]$
- root corresponds to $[1, n]$
- left child corresponds to left half of parents range,
  right child to right half

Segment trees

- binary tree where each node corresponds to a segment $[l, r]$
- root corresponds to $[1, n]$
- left child corresponds to left half of parents range,
  right child to right half
- each node stores the aggregate over its range of values

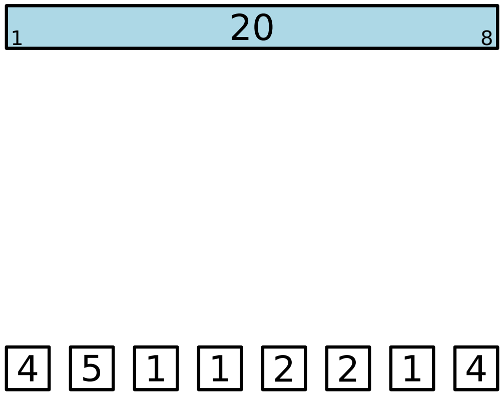
Segment trees

- binary tree where each node corresponds to a segment $[l, r]$
- root corresponds to $[1, n]$
- left child corresponds to left half of parents range,
  right child to right half
- each node stores the aggregate over its range of values

$\boxed{4}\ \boxed{5}\ \boxed{1}\ \boxed{1}\ \boxed{2}\ \boxed{2}\ \boxed{1}\ \boxed{4}$

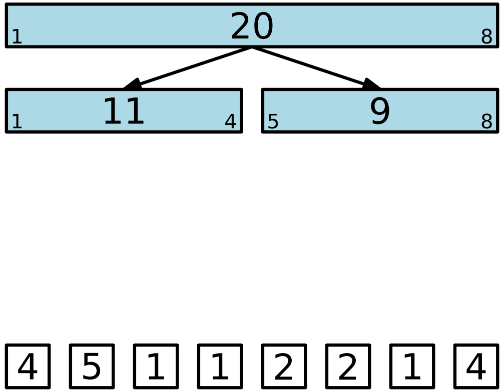
Segment trees

- binary tree where each node corresponds to a segment $[l, r]$
- root corresponds to $[1, n]$
- left child corresponds to left half of parents range,
  right child to right half
- each node stores the aggregate over its range of values



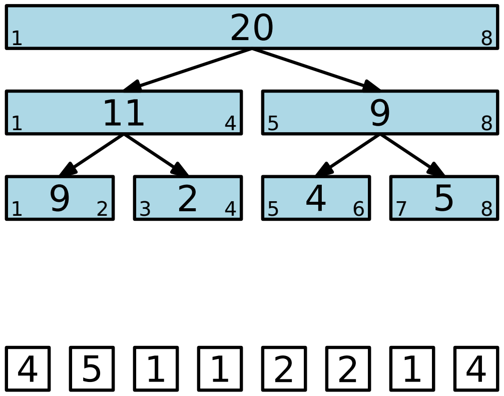
Segment trees

- binary tree where each node corresponds to a segment $[l, r]$
- root corresponds to $[1, n]$
- left child corresponds to left half of parents range, right child to right half
- each node stores the aggregate over its range of values



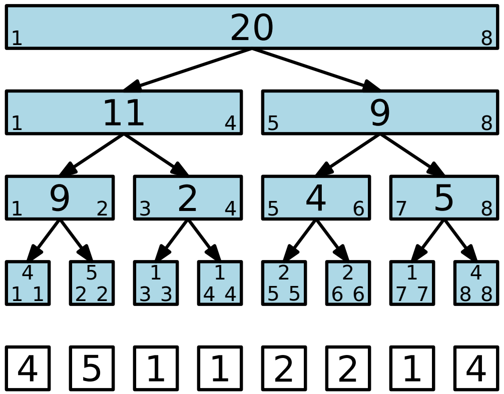
Segment trees

- binary tree where each node corresponds to a segment $[l, r]$
- root corresponds to $[1, n]$
- left child corresponds to left half of parents range, right child to right half
- each node stores the aggregate over its range of values



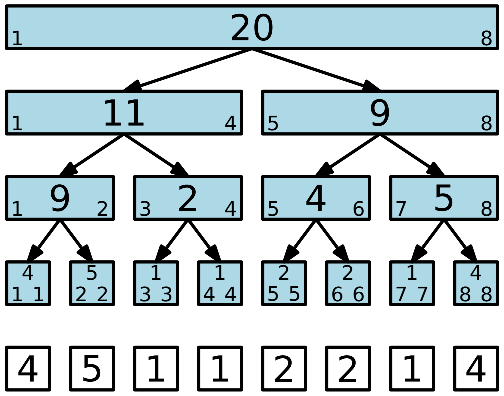
Segment trees

- binary tree where each node corresponds to a segment $[l, r]$
- root corresponds to $[1, n]$
- left child corresponds to left half of parents range, right child to right half
- each node stores the aggregate over its range of values



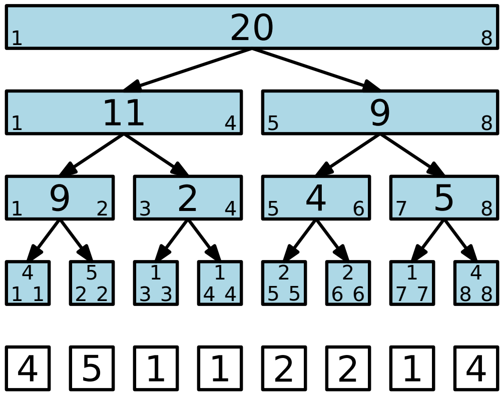
Segment trees

- binary tree where each node corresponds to a segment $[l, r]$
- root corresponds to $[1, n]$
- left child corresponds to left half of parents range,
  right child to right half
- each node stores the aggregate over its range of values
- how does this structure help us?



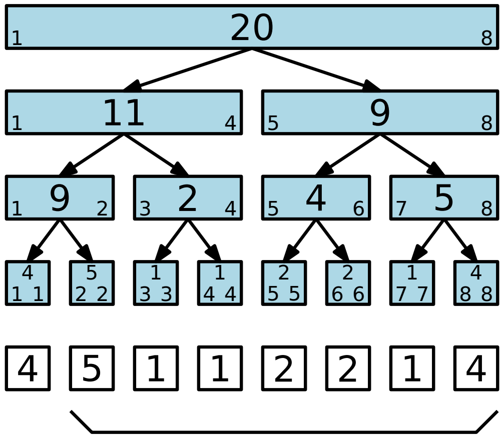
Segment trees

- binary tree where each node corresponds to a segment $[l, r]$
- root corresponds to $[1, n]$
- left child corresponds to left half of parents range, right child to right half
- each node stores the aggregate over its range of values
- how does this structure help us?
  - range queries



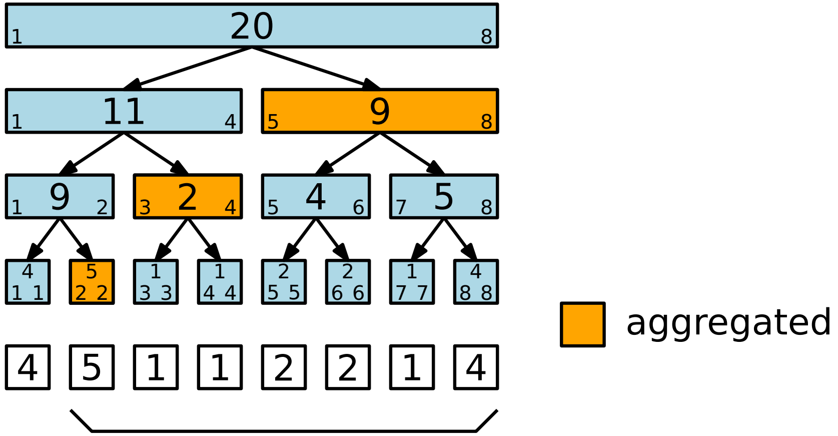
Segment trees

- binary tree where each node corresponds to a segment $[l, r]$
- root corresponds to $[1, n]$
- left child corresponds to left half of parents range,
  right child to right half
- each node stores the aggregate over its range of values
- how does this structure help us?
  - range queries



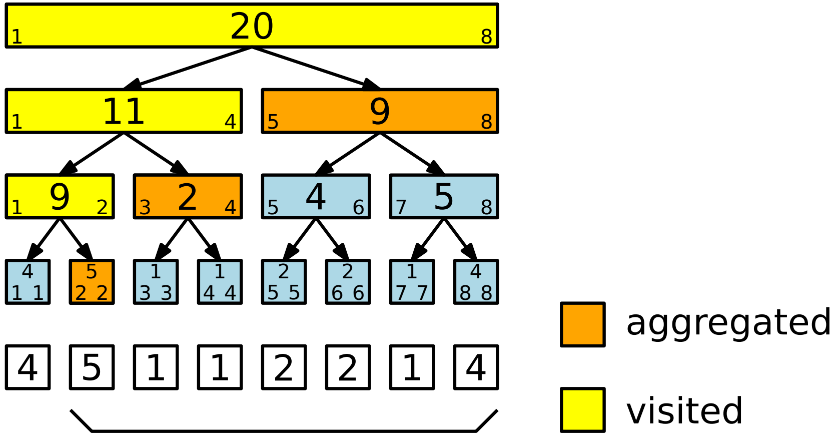
Segment trees

- binary tree where each node corresponds to a segment $[l, r]$
- root corresponds to $[1, n]$
- left child corresponds to left half of parents range, right child to right half
- each node stores the aggregate over its range of values
- how does this structure help us?
  - range queries



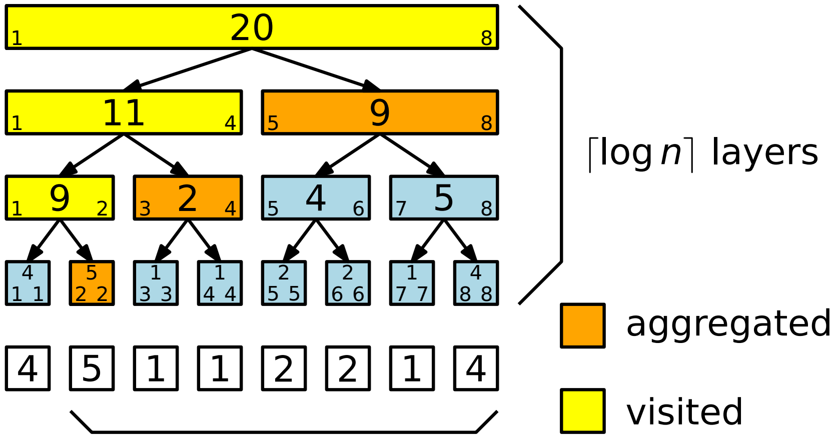
Segment trees

- binary tree where each node corresponds to a segment $[l, r]$
- root corresponds to $[1, n]$
- left child corresponds to left half of parents range,
  right child to right half
- each node stores the aggregate over its range of values
- how does this structure help us?
  - range queries



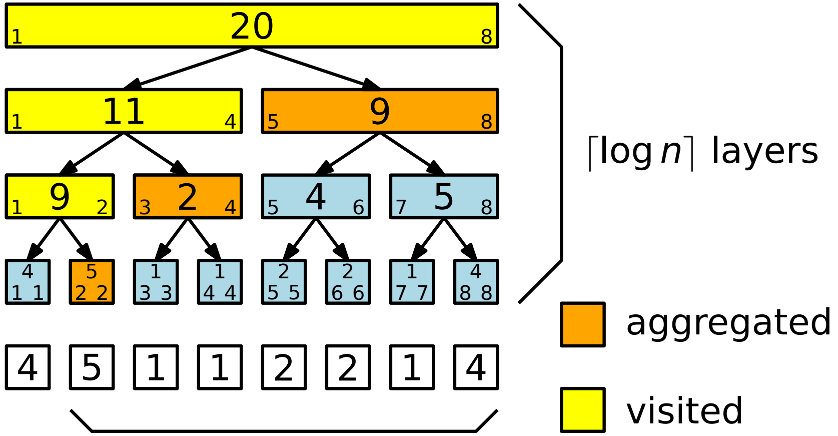
Segment trees

- binary tree where each node corresponds to a segment $[l, r]$
- root corresponds to $[1, n]$
- left child corresponds to left half of parents range, right child to right half
- each node stores the aggregate over its range of values
- how does this structure help us?
  - range queries



Segment trees

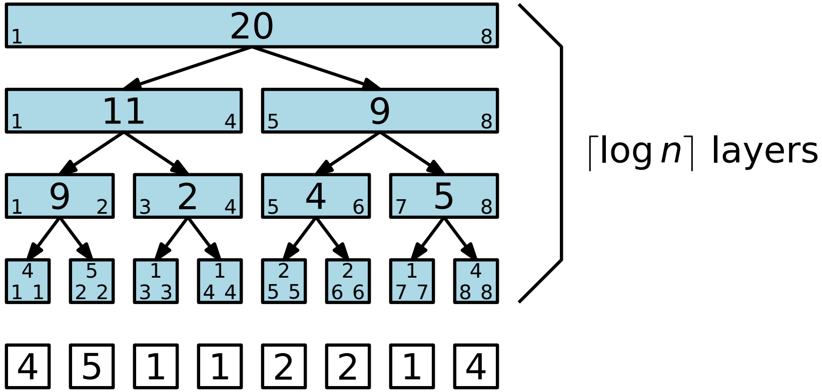
- binary tree where each node corresponds to a segment $[l, r]$
- root corresponds to $[1, n]$
- left child corresponds to left half of parents range,
  right child to right half
- each node stores the aggregate over its range of values
- how does this structure help us?
  - range queries
    - at most 4 nodes per layer*
      $\Rightarrow \mathcal{O}(\log n)$



*: can be proven via induction

Segment trees

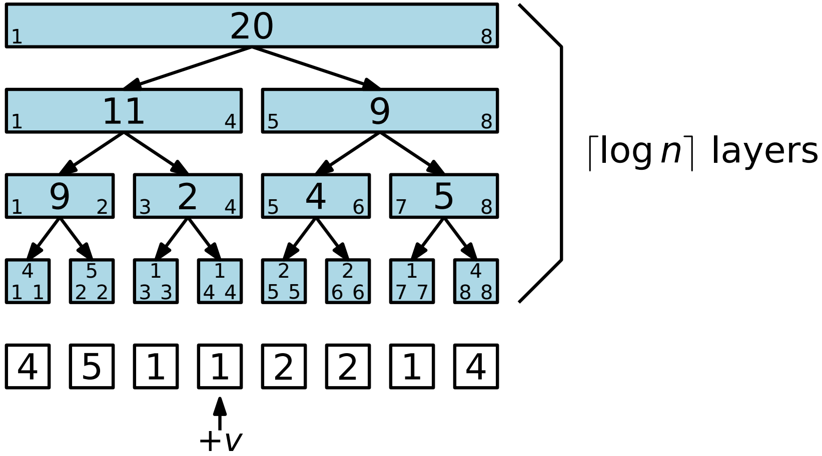
- binary tree where each node corresponds to a segment $[l, r]$
- root corresponds to $[1, n]$
- left child corresponds to left half of parents range, right child to right half
- each node stores the aggregate over its range of values
- how does this structure help us?
  - range queries
    - at most 4 nodes per layer* $\Rightarrow \mathcal{O}(\log n)$
  - point updates



*: can be proven via induction

Segment trees

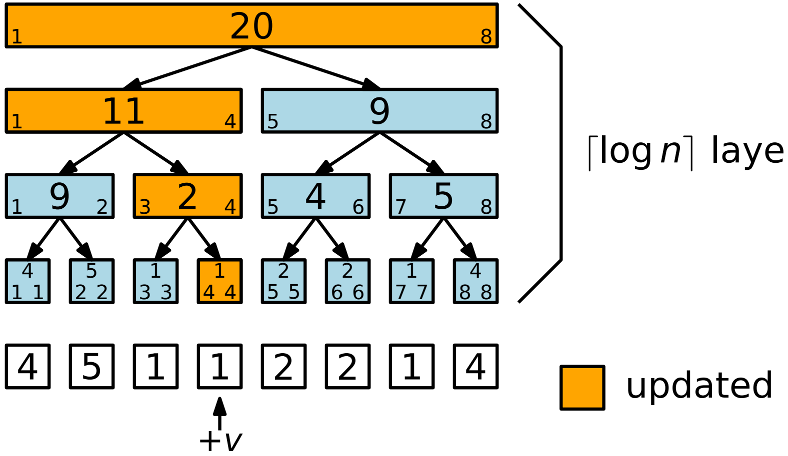
- binary tree where each node corresponds to a segment $[l, r]$
- root corresponds to $[1, n]$
- left child corresponds to left half of parents range, right child to right half
- each node stores the aggregate over its range of values
- how does this structure help us?
  - range queries
    - at most 4 nodes per layer* $\Rightarrow \mathcal{O}(\log n)$
  - point updates



*: can be proven via induction

Segment trees

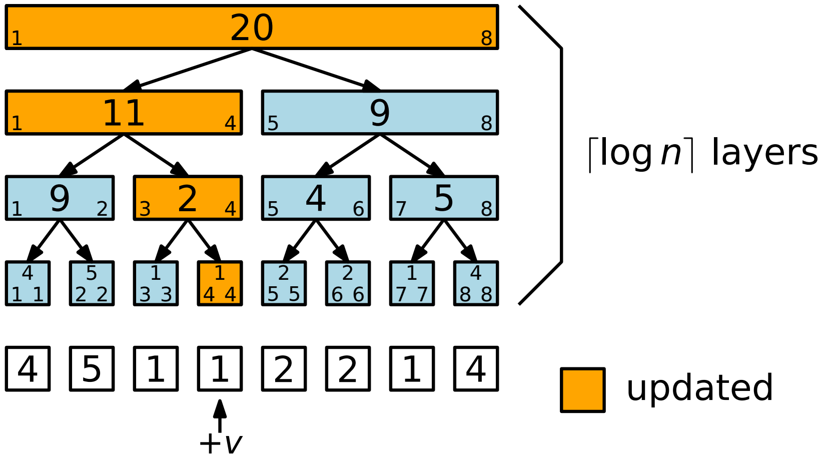
- binary tree where each node corresponds to a segment $[l, r]$
- root corresponds to $[1, n]$
- left child corresponds to left half of parents range, right child to right half
- each node stores the aggregate over its range of values
- how does this structure help us?
  - range queries
    - at most 4 nodes per layer*
      $\Rightarrow \mathcal{O}(\log n)$
  - point updates



*: can be proven via induction

Segment trees

- binary tree where each node corresponds to a segment $[l, r]$
- root corresponds to $[1, n]$
- left child corresponds to left half of parents range, right child to right half
- each node stores the aggregate over its range of values
- how does this structure help us?
  - range queries
    - at most 4 nodes per layer* $\Rightarrow \mathcal{O}(\log n)$
  - point updates
    - 1 node per layer $\Rightarrow \mathcal{O}(\log n)$



*: can be proven via induction

Segment trees — Range queries

range query(node, l, r, q_l, q_r)

  if $[l, r] \subseteq [q_l, q_r]$

    return node value

  $m = \lfloor (l + r)/2 \rfloor$

  res = 0

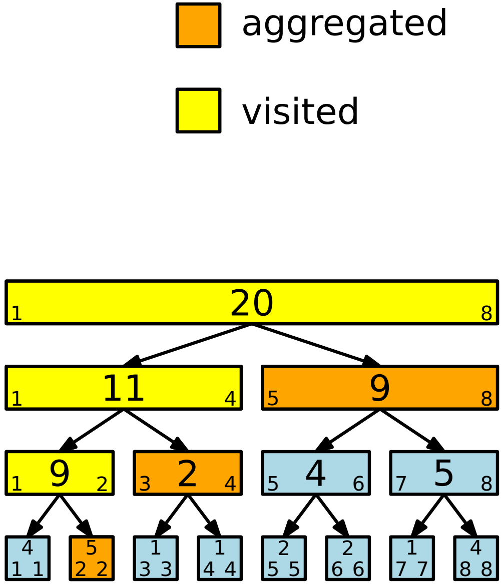
  if $[l, m] \cap [q_l, q_r] \neq \emptyset$

    res += **range query**(left child, l, m, q_l, q_r)

  if $[m + 1, r] \cap [q_l, q_r] \neq \emptyset$

    res += **range query**(right child, $m + 1, r, q_l, q_r$)

  return res



Segment trees — Range queries

range query(node, l, r, q_l, q_r)

  if $[l, r] \subseteq [q_l, q_r]$

    return node value

  $m = \lfloor (l + r)/2 \rfloor$

  res = 0

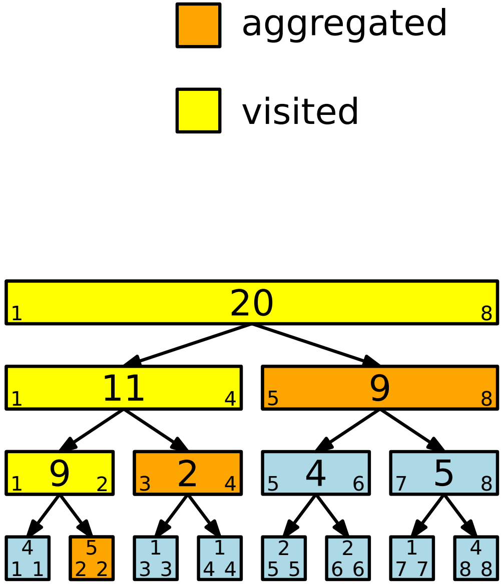
  if $[l, m] \cap [q_l, q_r] \neq \emptyset$

    res += **range query**(left child, l, m, q_l, q_r)

  if $[m + 1, r] \cap [q_l, q_r] \neq \emptyset$

    res += **range query**(right child, $m + 1, r, q_l, q_r$)

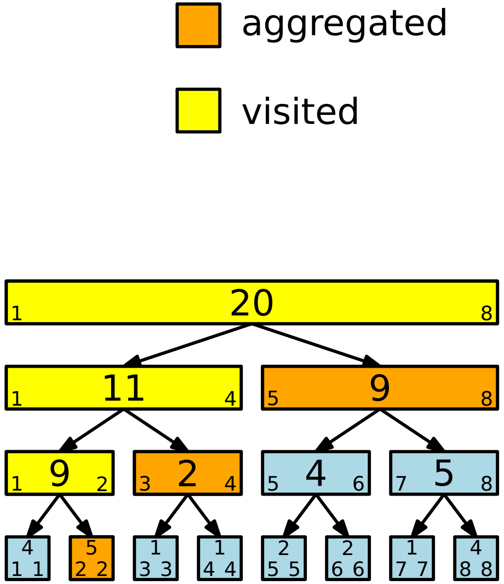
  return res



can we improve this code?

Segment trees — Range queries

range query(node, l, r, q_l, q_r)
    if $[l, r] \subseteq [q_l, q_r]$
        return node value
    $m = \lfloor (l + r)/2 \rfloor$
    res = 0
    if $[l, m] \cap [q_l, q_r] \neq \emptyset$
        res += **range query**(left child, l, m, q_l, q_r)
    if $[m + 1, r] \cap [q_l, q_r] \neq \emptyset$
        res += **range query**(right child, $m + 1, r, q_l, q_r$)
    return res



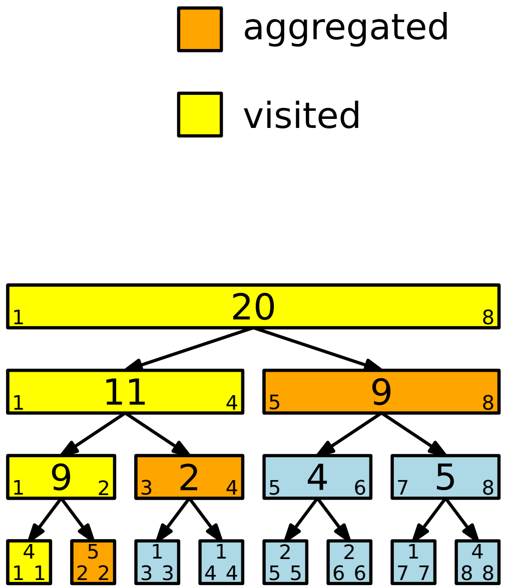
can we improve this code?

move range check into recursive call

(more visited nodes, but not asymptotically worse)

Segment trees — Range queries

range query(node, l, r, q_l, q_r)
  if $[l, r] \subseteq [q_l, q_r]$
    return node value
  $m = \lfloor (l + r)/2 \rfloor$
  res = 0
  if $[l, m] \cap [q_l, q_r] \neq \emptyset$
    res += **range query**(left child, l, m, q_l, q_r)
  if $[m + 1, r] \cap [q_l, q_r] \neq \emptyset$
    res += **range query**(right child, $m + 1, r, q_l, q_r$)
  return res

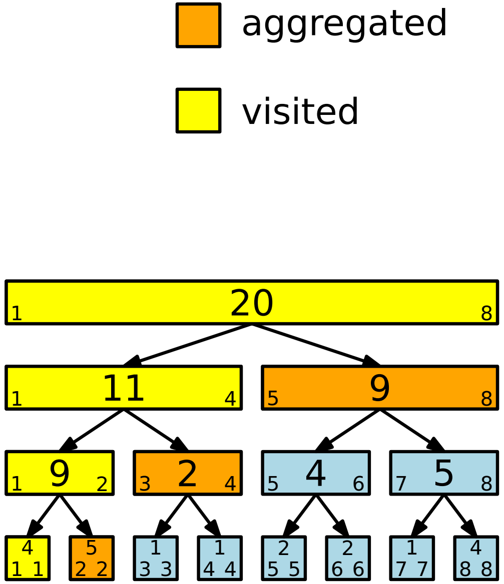


can we improve this code?

move range check into recursive call

(more visited nodes, but not asymptotically worse)

Segment trees — Range queries



range query(node, l, r, q_l, q_r)
  if $[l, r] \subseteq [q_l, q_r]$
    return node value
  if $[l, r] \cap [q_l, q_r] = \emptyset$
    return 0
  $m = \lfloor (l + r)/2 \rfloor$
  return range query(left child, l, m, q_l, q_r)+
    range query(right child, $m + 1, r, q_l, q_r$)

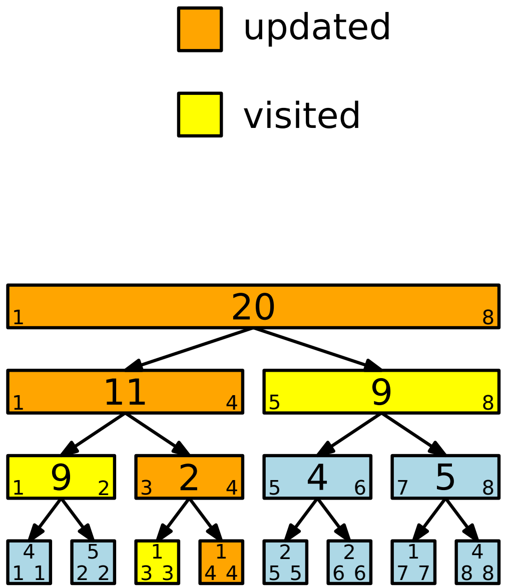
can we improve this code?

move range check into recursive call

(more visited nodes, but not asymptotically worse)

Segment trees — Point updates

point update(node, l, r, i, v)
  if $i \notin [l, r]$
    return node value
  if $l = r$
    node value += v
    return node value
  $m = \lfloor (l + r)/2 \rfloor$
  node value = **point update**(left child, l, m, i, v)+
        point update(right child, $m + 1, r, i, v$)
  return node value



Segment trees — Storage

- do not use `new` and pointers when implementing segment trees!*

*: except for persistent and implicit segment trees, which are far outside the scope of this lecture

Segment trees — Storage

- do not use `new` and pointers when implementing segment trees!*
  - using arrays/vectors is both faster and easier to debug

*: except for persistent and implicit segment trees, which are far outside the scope of this lecture

Segment trees — Storage

- do not use `new` and pointers when implementing segment trees![*]
  - using arrays/vectors is both faster and easier to debug
- how do we map a binary tree into an array?

*: except for persistent and implicit segment trees, which are far outside the scope of this lecture

Segment trees — Storage

- do not use `new` and pointers when implementing segment trees!*
  - using arrays/vectors is both faster and easier to debug
- how do we map a binary tree into an array? heap indexing!

*: except for persistent and implicit segment trees, which are far outside the scope of this lecture

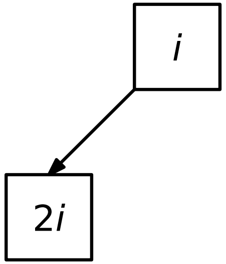
Segment trees — Storage

- do not use `new` and pointers when implementing segment trees!*
  - using arrays/vectors is both faster and easier to debug
- how do we map a binary tree into an array? heap indexing!
  - for a node with index *i*

*: except for persistent and implicit segment trees, which are far outside the scope of this lecture

Segment trees — Storage

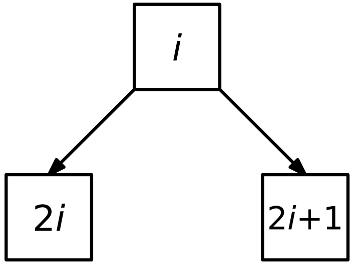
- do not use `new` and pointers when implementing segment trees!*
  - using arrays/vectors is both faster and easier to debug
- how do we map a binary tree into an array? heap indexing!
  - for a node with index i
    - left child is $2i$



*: except for persistent and implicit segment trees, which are far outside the scope of this lecture

Segment trees — Storage

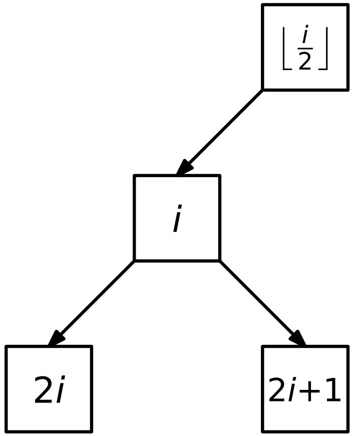
- do not use `new` and pointers when implementing segment trees!*
  - using arrays/vectors is both faster and easier to debug
- how do we map a binary tree into an array? heap indexing!
  - for a node with index i
    - left child is $2i$
    - right child is $2i + 1$



*: except for persistent and implicit segment trees, which are far outside the scope of this lecture

Segment trees — Storage

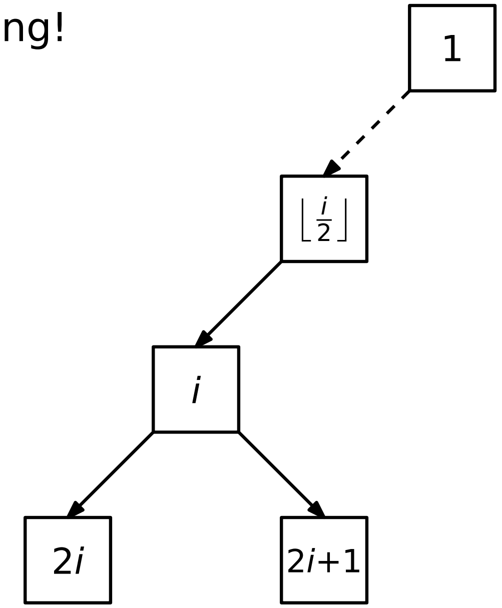
- do not use `new` and pointers when implementing segment trees!*
  - using arrays/vectors is both faster and easier to debug
- how do we map a binary tree into an array? heap indexing!
  - for a node with index i
    - left child is $2i$
    - right child is $2i + 1$
    - parent is $\left\lfloor \frac{i}{2} \right\rfloor$



*: except for persistent and implicit segment trees, which are far outside the scope of this lecture

Segment trees — Storage

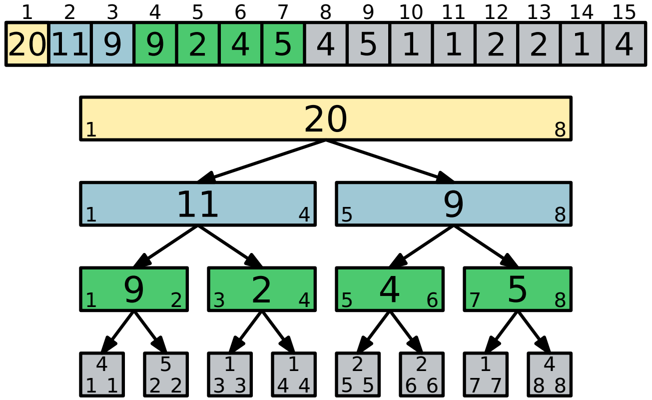
- do not use `new` and pointers when implementing segment trees!*
  - using arrays/vectors is both faster and easier to debug
- how do we map a binary tree into an array? heap indexing!
  - for a node with index i
    - left child is $2i$
    - right child is $2i + 1$
    - parent is $\left\lfloor \frac{i}{2} \right\rfloor$
  - root is 1

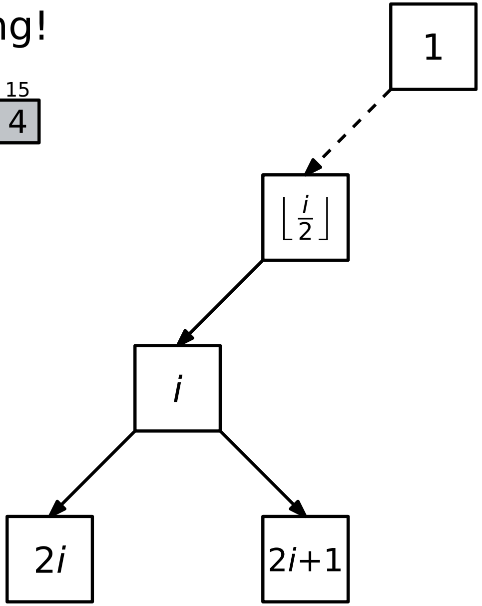


*: except for persistent and implicit segment trees, which are far outside the scope of this lecture

Segment trees — Storage

- do not use `new` and pointers when implementing segment trees!*
  - using arrays/vectors is both faster and easier to debug
- how do we map a binary tree into an array? heap indexing!
  - for a node with index i
    - left child is $2i$
    - right child is $2i+1$
    - parent is $\left\lfloor \frac{i}{2} \right\rfloor$
  - root is 1

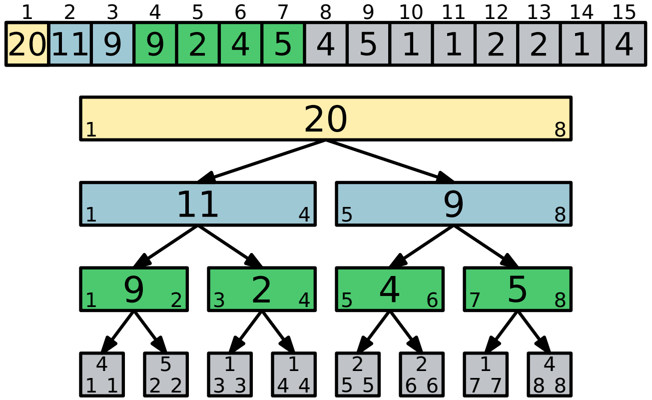


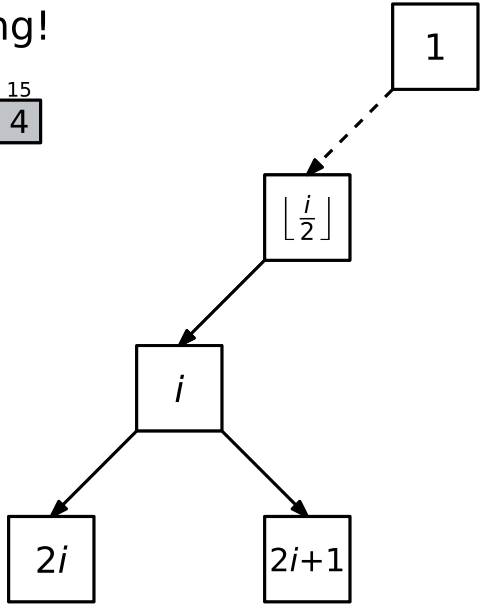


*: except for persistent and implicit segment trees, which are far outside the scope of this lecture

Segment trees — Storage

- do not use `new` and pointers when implementing segment trees!*
  - using arrays/vectors is both faster and easier to debug
- how do we map a binary tree into an array? heap indexing!
  - for a node with index i
    - left child is $2i$
    - right child is $2i+1$
    - parent is $\left\lfloor \frac{i}{2} \right\rfloor$
  - root is 1
- we need space for $\leq 2^{\lceil \log n \rceil + 1} \leq 4n$ nodes





*: except for persistent and implicit segment trees, which are far outside the scope of this lecture

Segment trees — Conclusion

- pros/cons

Segment trees — Conclusion

- pros/cons
  - + $\mathcal{O}(\log n)$ for all our operations

Segment trees — Conclusion

- pros/cons
  - + $\mathcal{O}(\log n)$ for all our operations
  - + almost universally applicable. we only need ways to

Segment trees — Conclusion

- pros/cons
  - + $\mathcal{O}(\log n)$ for all our operations
  - + almost universally applicable. we only need ways to
    - calculate aggregate from a single element

Segment trees — Conclusion

- pros/cons
  - + $\mathcal{O}(\log n)$ for all our operations
  - + almost universally applicable. we only need ways to
    - calculate aggregate from a single element
    - combine aggregate of adjacent ranges

Segment trees — Conclusion

- pros/cons
  - + $\mathcal{O}(\log n)$ for all our operations
  - + almost universally applicable. we only need ways to
    - calculate aggregate from a single element
    - combine aggregate of adjacent ranges
  - + incredibly flexible and extensible (lazy propagation, 2d, persistent, implicit)

Segment trees — Conclusion

- pros/cons
  - + $\mathcal{O}(\log n)$ for all our operations
  - + almost universally applicable. we only need ways to
    - calculate aggregate from a single element
    - combine aggregate of adjacent ranges
  - + incredibly flexible and extensible (lazy propagation, 2d, persistent, implicit)
  - + low memory overhead ($4n$, $2n$ possible)

Segment trees — Conclusion

- pros/cons
  - + $\mathcal{O}(\log n)$ for all our operations
  - + almost universally applicable. we only need ways to
    - calculate aggregate from a single element
    - combine aggregate of adjacent ranges
  - + incredibly flexible and extensible (lazy propagation, 2d, persistent, implicit)
  - + low memory overhead ($4n$, $2n$ possible)
  - + fast compared to other search trees (red-black trees, treaps, $\ldots$)

Segment trees — Conclusion

- pros/cons
  - + $\mathcal{O}(\log n)$ for all our operations
  - + almost universally applicable. we only need ways to
    - calculate aggregate from a single element
    - combine aggregate of adjacent ranges
  - + incredibly flexible and extensible (lazy propagation, 2d, persistent, implicit)
  - + low memory overhead ($4n$, $2n$ possible)
  - + fast compared to other search trees (red-black trees, treaps, $\ldots$)
  - – no way to reorder/insert/delete elements (see: treaps)

Segment trees — Conclusion

- pros/cons
  - \+ $\mathcal{O}(\log n)$ for all our operations
  - \+ almost universally applicable. we only need ways to
    - calculate aggregate from a single element
    - combine aggregate of adjacent ranges
  - \+ incredibly flexible and extensible (lazy propagation, 2d, persistent, implicit)
  - \+ low memory overhead ($4n$, $2n$ possible)
  - \+ fast compared to other search trees (red-black trees, treaps, ...)
  - – no way to reorder/insert/delete elements (see: treaps)
    - often not a problem if we know all elements upfront

Overview

	query		update	
	point	range	point	range
difference array[†]	–	–	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$	–	–
sqrt decomposition	$\mathcal{O}(1)$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(\sqrt{n})$	–
segment tree	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)^{\S}$

*: requires invertible operation
†: as base for other DS, enables range updates at the price of range queries
§: requires lazy propagation

Overview

	query		update	
	point	range	point	range
difference array†	–	–	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–
prefix array	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$	–	–
sqrt decomposition	$\mathcal{O}(1)$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(\sqrt{n})$	–
segment tree	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)^\S$
treap	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)^\S$

*: requires invertible operation
†: as base for other DS, enables range updates at the price of range queries
§: requires lazy propagation

Overview

	query		update		walk your dog and make you breakfast	
	point	range	point	range	point	range
difference array†	–	–	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$	–	–
input array	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	–	–	–
prefix array	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$	–	–	–	–
sqrt decomposition	$\mathcal{O}(1)$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(\sqrt{n})$	–	–	–
segment tree	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)^{\S}$	–	–
treap	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)^{\S}$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

$*$: requires invertible operation
†: as base for other DS, enables range updates at the price of range queries
§: requires lazy propagation

Advanced stuff for the interested

- https://cp-algorithms.com/data_structures/segment_tree.html
  - everything you ever wanted to know about segment trees
  - basics, lazy propagation, 2d, persistent, implicit, . . .

Advanced stuff for the interested

- https://cp-algorithms.com/data_structures/segment_tree.html
  - everything you ever wanted to know about segment trees
  - basics, lazy propagation, 2d, persistent, implicit, ...
- https://codeforces.com/blog/entry/18051
  - iterative (non-recursive) segment trees

```
int query(int l, int r) {
  int res = 0;
  for (l += k, r += k; l < r;
       l /= 2, r /= 2) {
    if (l & 1) res += d[l++];
    if (r & 1) res += d[--r];
  }
  return res;
}

void update(int i, int diff) {
  d[i + k] += diff;
  for (int j = i + k; j /= 2;)
    d[j] = d[2*j] + d[2*j+1];
}
```

Advanced stuff for the interested

- https://cp-algorithms.com/data_structures/segment_tree.html
  - everything you ever wanted to know about segment trees
  - basics, lazy propagation, 2d, persistent, implicit, ...
- https://codeforces.com/blog/entry/18051
  - iterative (non-recursive) segment trees
- https://cp-algorithms.com/data_structures/fenwick.html
  - dual fenwick tree trick and much more

```
int query(int l, int r) {
  int res = 0;
  for (l += k, r += k; l < r;
       l /= 2, r /= 2) {
    if (l & 1) res += d[l++];
    if (r & 1) res += d[--r];
  }
  return res;
}

void update(int i, int diff) {
  d[i + k] += diff;
  for (int j = i + k; j /= 2;)
    d[j] = d[2*j] + d[2*j+1];
}
```

Advanced stuff for the interested

- https://cp-algorithms.com/data_structures/segment_tree.html
  - everything you ever wanted to know about segment trees
  - basics, lazy propagation, 2d, persistent, implicit, ...
- https://codeforces.com/blog/entry/18051
  - iterative (non-recursive) segment trees
- https://cp-algorithms.com/data_structures/fenwick.html
  - dual fenwick tree trick and much more
- https://cp-algorithms.com/geometry/convex_hull_trick.html
  - really crazy segment tree application (convex hull of ranges of linear functions)

```
int query(int l, int r) {
  int res = 0;
  for (l += k, r += k; l < r;
       l /= 2, r /= 2) {
    if (l & 1) res += d[l++];
    if (r & 1) res += d[--r];
  }
  return res;
}

void update(int i, int diff) {
  d[i + k] += diff;
  for (int j = i + k; j /= 2;)
    d[j] = d[2*j] + d[2*j+1];
}
```