

CS 5199 – Competition Programming and Problem Solving Seminar

Segment Tree

Raunak Kumar

Based on slides by Paul Liu, Kuba Karpierz, Bruno Vacherot, Raunak Kumar and Jason Chiu for CPSC 490 – Problem Solving in Computer Science at The University of British Columbia

2019/10/07

Cornell University

You are given an array A of n integers. We want to support the following operations

- Print the array in sorted order.
 - Solution: Just sort the array and print it.

You are given an array A of n integers. We want to support the following operations

- Print the array in sorted order.
 - Solution: Just sort the array and print it.
- Additionally, allow insertion/deletion of elements.
 - Solution: use a different data structure, like a binary search tree.

You are given an array A of n integers. We want to support the following operations

- Print the array in sorted order.
 - Solution: Just sort the array and print it.
- Additionally, allow insertion/deletion of elements.
 - Solution: use a different data structure, like a binary search tree.

We have a collection of elements and we would like to support some operations on these elements.

⇒ Often, the solution is to use specialized data structures.

Range Query

You are given an array A of n integers. How efficiently can you answer these queries?

Range Query

You are given an array A of n integers. How efficiently can you answer these queries?

- $\text{sum}(i, j) = A[i] + A[i+1] + \dots + A[j]$
- $\text{min}(i, j) = \min(A[i], \dots, A[j])$
- Median
- Mode

Range Query

You are given an array A of n integers. How efficiently can you answer these queries?

- $\text{sum}(i, j) = A[i] + A[i+1] + \dots + A[j]$
- $\text{min}(i, j) = \min(A[i], \dots, A[j])$
- Median
- Mode
- What if we allow updates?

Range Query with Updates

Given an array A of n integers, perform these queries efficiently:

- `update(i, x)`: set $A[i] = x$
- `sum(i, j)`: return $A[i] + A[i+1] + \dots + A[j]$

Potential Solutions

We could try the following:

- Option 1: Naively perform the 2 queries.
 - `update(i, x)` takes $O(1)$ time
 - `sum(i, j)` takes $O(n)$ time.

Potential Solutions

We could try the following:

- Option 1: Naively perform the 2 queries.
 - `update(i, x)` takes $O(1)$ time
 - `sum(i, j)` takes $O(n)$ time.
- Option 2: Precompute the sum in $O(n^2)$ intervals.
 - `update(i, x)` takes $O(n)$ time, to change $O(n)$ intervals
 - `sum(i, j)` takes $O(1)$ time

Potential Solutions

We could try the following:

- Option 1: Naively perform the 2 queries.
 - `update(i, x)` takes $O(1)$ time
 - `sum(i, j)` takes $O(n)$ time.
- Option 2: Precompute the sum in $O(n^2)$ intervals.
 - `update(i, x)` takes $O(n)$ time, to change $O(n)$ intervals
 - `sum(i, j)` takes $O(1)$ time

Can we do better? Can we do a different type of precomputation that allows us to **update** and **sum** fast?

Given an array A of n integers, answer these queries quickly:

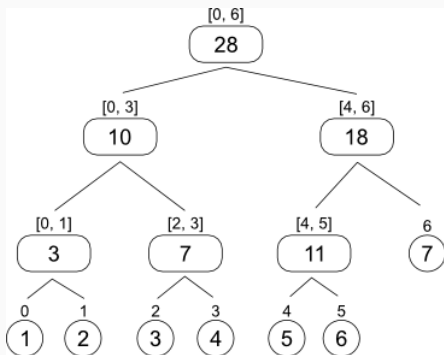
- `update(i, x)`: sets $A[i] = x$
- `sum(i, j)`: return $A[i] + \dots + A[j]$

Segment tree can solve this in $O(\log n)$ per query.

Segment Trees

Basic idea: add a lot of layers!

- Segment tree: group every pair together, then every pair of pairs, then every pair of that, until you get 1 node



Source: <http://scvalex.github.io/articles/SegmentTree.html>

Segment Trees – Structure, Update

Structure

- Binary tree of $\log n$ layers, $1 + 2 + 4 + \dots + n = O(n)$ nodes
- Leaf node: represent one element
- Internal node: represent union of interval of left + right child

Point Update

- Update leaf
- Update ancestors

Segment Trees (Sum Query) – Structure, Update

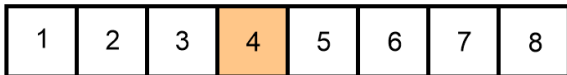
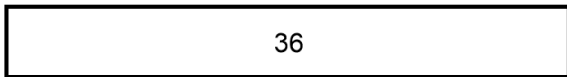
Structure

- Binary tree of $\log n$ layers, $1 + 2 + 4 + \dots + n = O(n)$ nodes
- Leaf node: store **value of one element**
- Internal node: store **sum of left + right child**

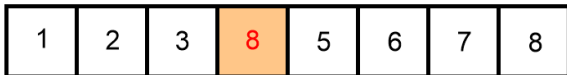
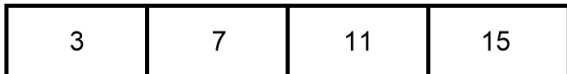
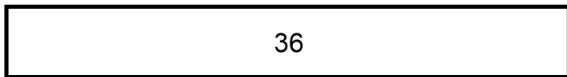
Point Update

- Update leaf: **change value of leaf node**
- Update ancestor: **re-compute sum of left + right child**

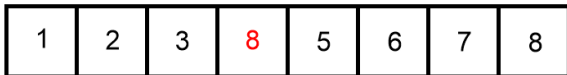
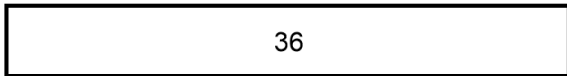
Segment Trees (Sum Query) – Point Update



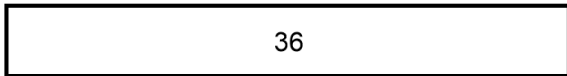
Segment Trees (Sum Query) – Point Update



Segment Trees (Sum Query) – Point Update



Segment Trees (Sum Query) – Point Update



Segment Trees (Sum Query) – Point Update

40

14	26
----	----

3	11	11	15
---	----	----	----

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

Range query

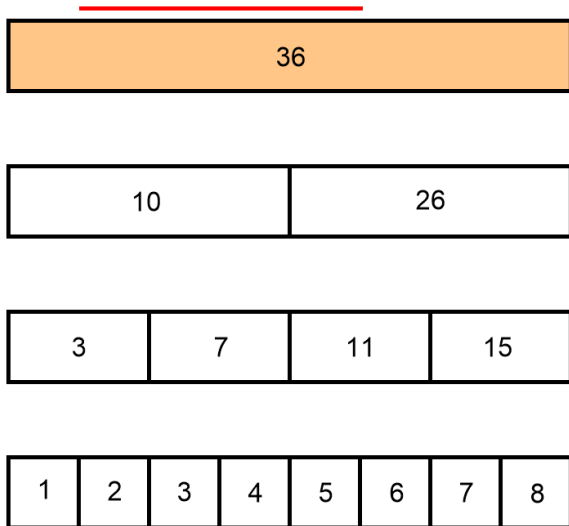
- Start at root and recurse down
- Suppose query for $[a, b]$, current node represent $[l, r]$
- Case 1: $[l, r] \subseteq [a, b]$, return value of current node
- Case 2: $[l, r] \not\subseteq [a, b]$, recurse then combine answer from children
 1. If $[l, m] \cap [a, b] \neq \emptyset$ recurse left child and get answer
 2. If $[m + 1, r] \cap [a, b] \neq \emptyset$ recurse right child and get answer
 3. Combine above two answers

Range query

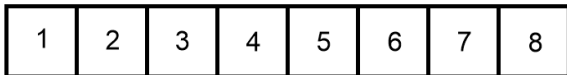
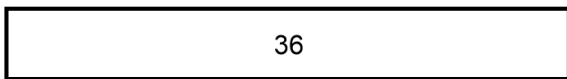
- Start at root and recurse down
- Suppose query for $[a, b]$, current node represent $[l, r]$
- Case 1: $[l, r] \subseteq [a, b]$, return value of current node
- Case 2: $[l, r] \not\subseteq [a, b]$, recurse then combine answer from children
 1. If $[l, m] \cap [a, b] \neq \emptyset$ recurse left child and get answer
 2. If $[m + 1, r] \cap [a, b] \neq \emptyset$ recurse right child and get answer
 3. Combine above two answers

Sum query: add #1 and #2

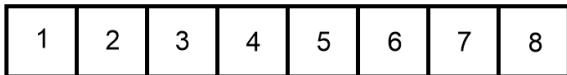
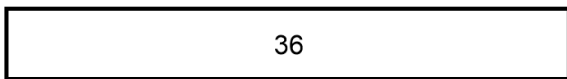
Segment Trees (Sum Query) – Range Query



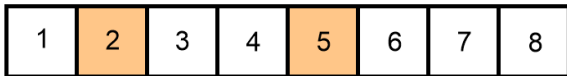
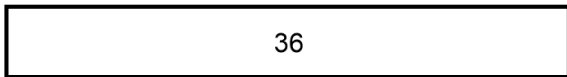
Segment Trees (Sum Query) – Range Query



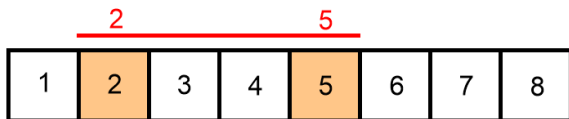
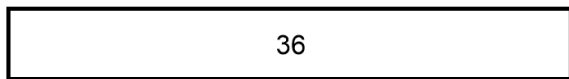
Segment Trees (Sum Query) – Range Query



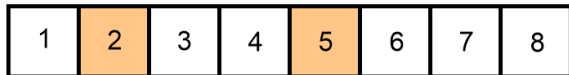
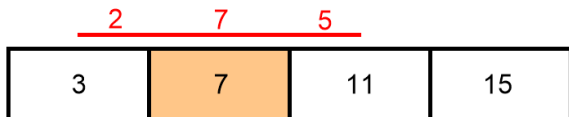
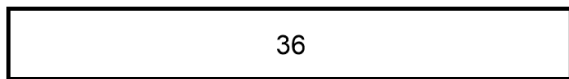
Segment Trees (Sum Query) – Range Query



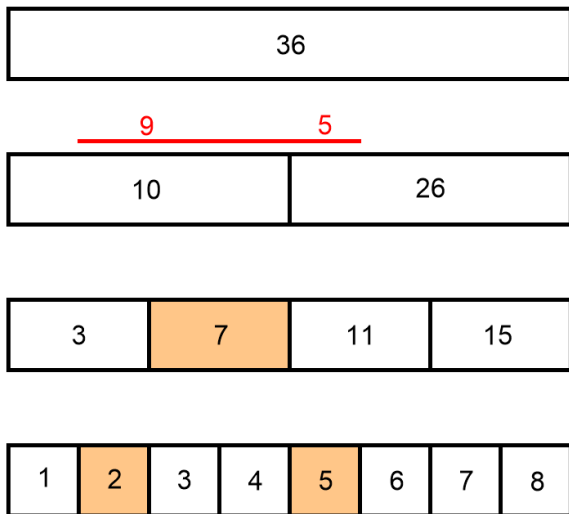
Segment Trees (Sum Query) – Range Query



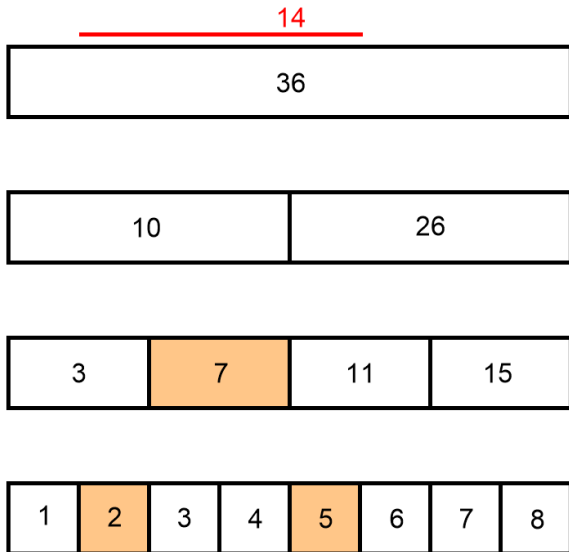
Segment Trees (Sum Query) – Range Query



Segment Trees (Sum Query) – Range Query



Segment Trees (Sum Query) – Range Query



Segment Tree - Construction

- You could call update on each element $\Rightarrow O(n \log n)$
- Better: fill from leaf upwards layer by layer $\Rightarrow O(n)$

Segment Tree - Implementation

Approach is similar to e.g. heap implementation

- Store the tree in an array $A[1 \dots 2n]$.
- Root = $A[1]$.
- Children of node $i = A[2i]$ and $A[2i + 1]$.
- Parent of node $i = i/2$.

Segment Tree - Initialize

```
1 // Set input size as some large power of 2.  
2 const int MAXN = 1 << 17;  
3  
4 // Store segment tree in a flat array.  
5 int T[2*MAXN];
```

Segment Tree - Build

```
1 void build(int A[MAXN]) {
2     // initialize leaf
3     for (int i = 0; i < MAXN; i++)
4         T[MAXN + i] = A[i];
5
6     // initialize internal nodes, bottom up
7     for (int i = MAXN-1; i > 0; i--) {
8         T[i] = T[2*i] + T[2*i+1];
9     }
10 }
```

Segment Tree - Point Update

```
1 void update(int x, int val) {
2     // Change the leaf node's value.
3     int v = MAXN + x;
4     T[v] = val;
5
6     // Propagate the change all the way to the root.
7     for (int i = v/2; i > 0; i /= 2)
8         T[i] = T[2*i] + T[2*i + 1];
9 }
```

Segment Tree - Range Query

```
1  int query(int x, int y, int i=1, int l=0, int r=MAXN-1)
2      // [l,r] is completely outside [x,y], return 0
3      if (x > r || y < l) return 0;
4
5      // [l,r] is completely in [x,y], return node value
6      if (x <= l && r <= y) return T[i];
7
8      // Otherwise, recurse on children.
9      return query(x, y, 2*i, l, (l+r)/2)
10         + query(x, y, 2*i+1, (l+r)/2+1, r);
11 }
```

Problem 1

Support the following operations on an array $A[1 \dots n]$

- *update* (same as before)
- *query*(l, r) returns the maximum sum subarray within this subrange.

Problem 1 - Solution

Consider a range $[l, r]$. How could we get the maximum sum subarray in this range?

- The answer lies entirely in the left child or entirely in the right.
- The answer could spans the left and right children.
- So $ans(l, r) = \max\{ans(left), ans(right), sum(across)\}$.

Problem 1 - Solution

Consider a range $[l, r]$. How could we get the maximum sum subarray in this range?

- The answer lies entirely in the left child or entirely in the right.
- The answer could spans the left and right children.
- So $ans(l, r) = \max\{ans(left), ans(right), sum(across)\}$.

How do we get the maximum sum subarray that goes across the left and right children?

Problem 1 - Solution

Consider a range $[l, r]$. How could we get the maximum sum subarray in this range?

- The answer lies entirely in the left child or entirely in the right.
- The answer could spans the left and right children.
- So $ans(l, r) = \max\{ans(left), ans(right), sum(across)\}$.

How do we get the maximum sum subarray that goes across the left and right children?

Clearly, we need to store more information in the nodes.

Problem 1 - Solution

Store these information in each node

- Sum of entire subrange contained in the node = $node.sum$
- Best prefix sum = $node.prefix$
- Best suffix sum = $node.suffix$
- Answer = $node.ans$

Update queries

- $node.sum = left.sum + right.sum$
- $node.prefix = \max(left.prefix, left.sum + right.prefix)$
- $node.suffix = \max(right.suffix, left.suffix + right.sum)$
- $node.ans = \max(left.ans, right.ans, left.suffix + right.prefix)$

Time Complexity: $O(\log n)$.

So far we have only been dealing with “point updates”, where we change only a single input element at a time.

Range Updates

So far we have only been dealing with “point updates”, where we change only a single input element at a time.

Suppose now we also need to be update an interval $[l, r]$ – for example, add 3 to every element in an interval. How do we do this?

Range Updates

So far we have only been dealing with “point updates”, where we change only a single input element at a time.

Suppose now we also need to be update an interval $[l, r]$ – for example, add 3 to every element in an interval. How do we do this?

Naive way: call *update* once for each element in the interval.

Time Complexity: $O(n \log n)$! Can we do better?

Lazy Propagation

Basic idea

- From range query: range $[l, r]$ = union of ranges represented by $O(\log n)$ nodes, so we can just update value at these nodes!
- Idea: at these nodes, add a “todo” variable saying every leaf in the subtree needs to be updated by x , don't recurse further
- Update ancestors as usual

Lazy Propagation

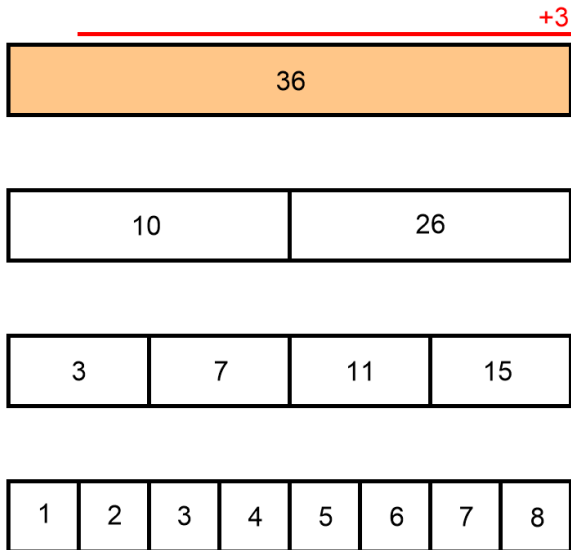
Basic idea

- From range query: range $[l, r]$ = union of ranges represented by $O(\log n)$ nodes, so we can just update value at these nodes!
- Idea: at these nodes, add a “todo” variable saying every leaf in the subtree needs to be updated by x , don't recurse further
- Update ancestors as usual

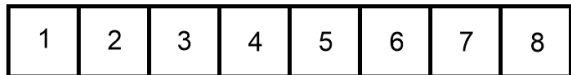
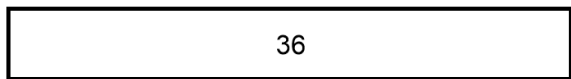
What if end point of next update/query interval lands in the middle of a node with a “todo”?

- Before recursing on children, push the “todo” down one level
- After recursing on children, continue as usual

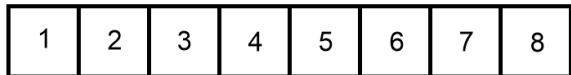
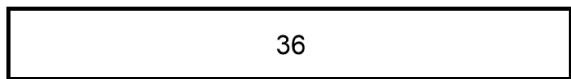
Lazy Propagation – Lazy Update



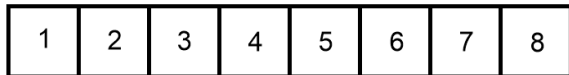
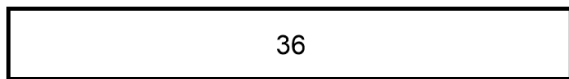
Lazy Propagation – Lazy Update



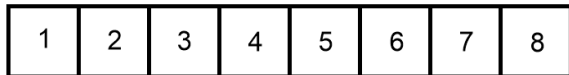
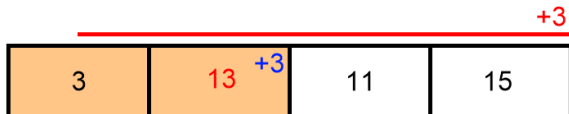
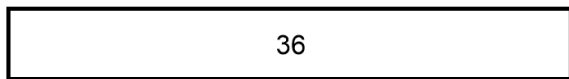
Lazy Propagation – Lazy Update



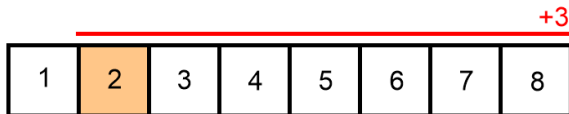
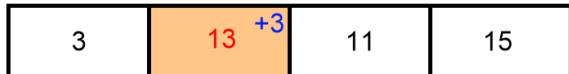
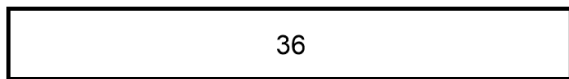
Lazy Propagation – Lazy Update



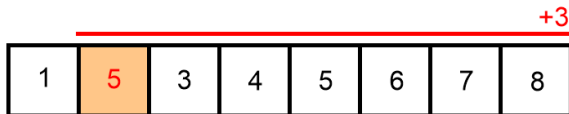
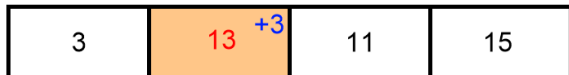
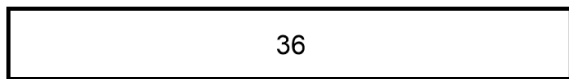
Lazy Propagation – Lazy Update



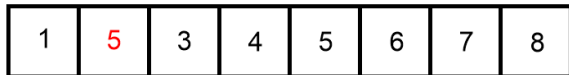
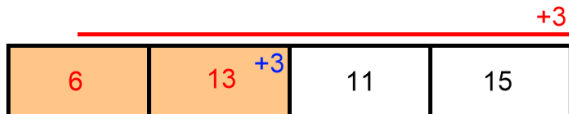
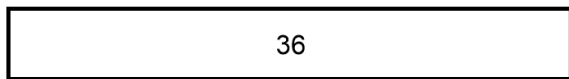
Lazy Propagation – Lazy Update



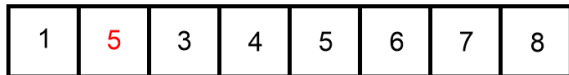
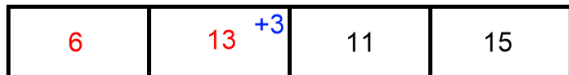
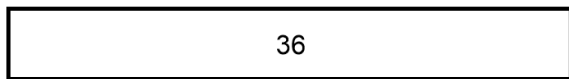
Lazy Propagation – Lazy Update



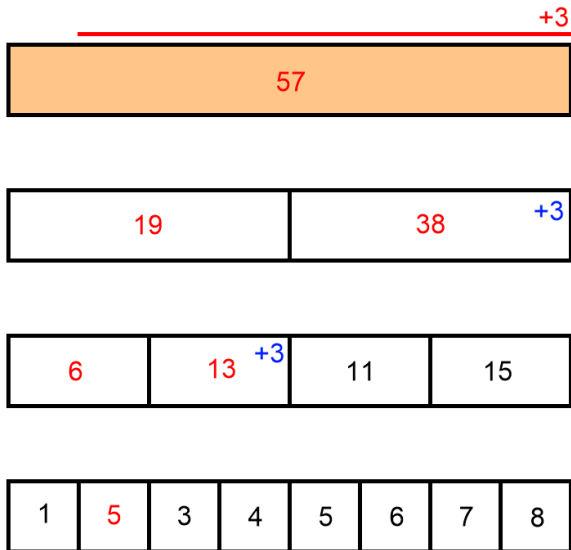
Lazy Propagation – Lazy Update



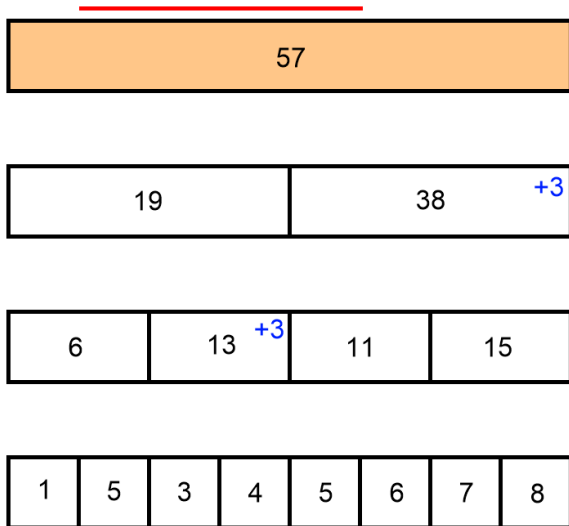
Lazy Propagation – Lazy Update



Lazy Propagation – Lazy Update



Lazy Propagation – Range Query



Lazy Propagation – Range Query

57

19	38 +3
----	-------

6	13 +3	11	15
---	-------	----	----

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

Lazy Propagation – Range Query

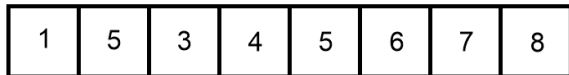
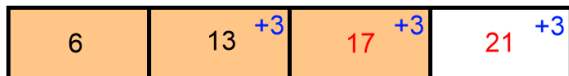
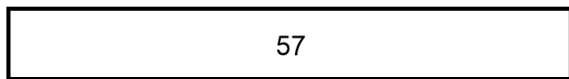
57

19	38
----	----

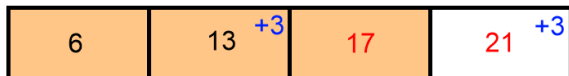
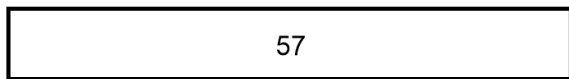
6	13 ⁺³	17 ⁺³	21 ⁺³
---	------------------	------------------	------------------

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

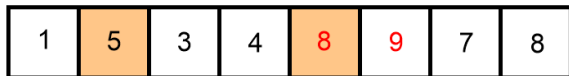
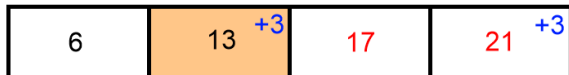
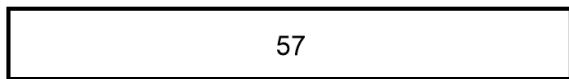
Lazy Propagation – Range Query



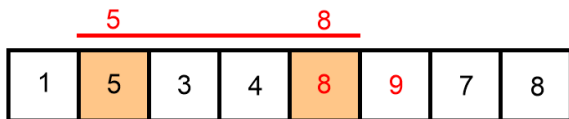
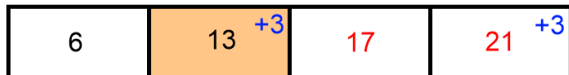
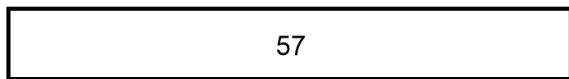
Lazy Propagation – Range Query



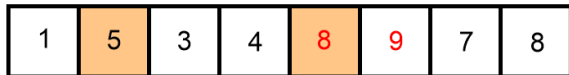
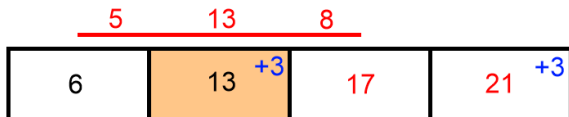
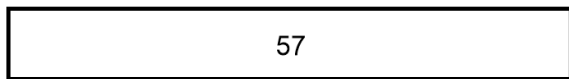
Lazy Propagation – Range Query



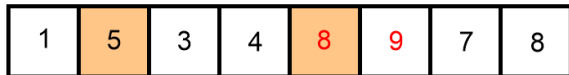
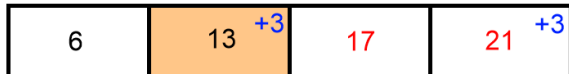
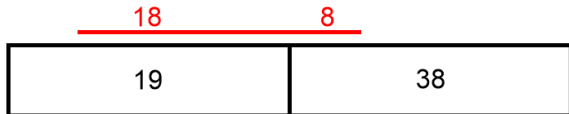
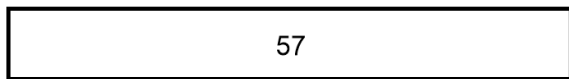
Lazy Propagation – Range Query



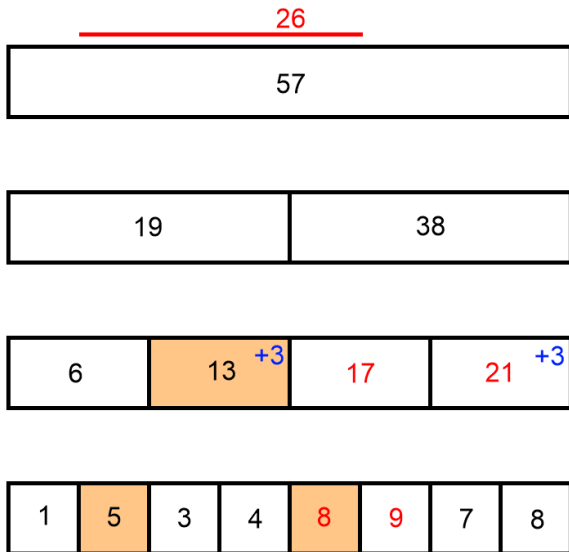
Lazy Propagation – Range Query



Lazy Propagation – Range Query



Lazy Propagation – Range Query



Feedback form: <https://forms.gle/HqAufcsFUBNN7yfi7>