

# Lazy Segment Trees

CS 491 – Competitive Programming

**Dr. Mattox Beckman**

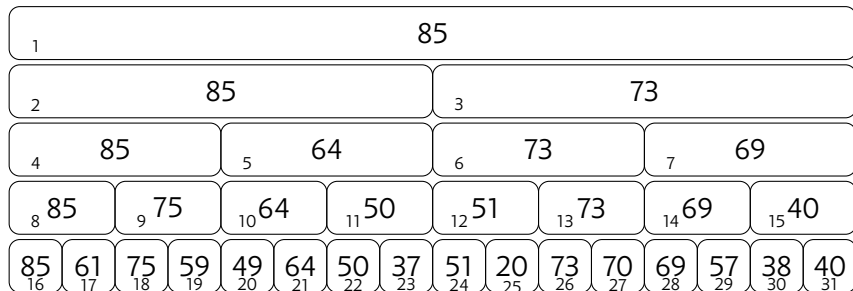
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN  
DEPARTMENT OF COMPUTER SCIENCE

Spring 2023

# Objectives

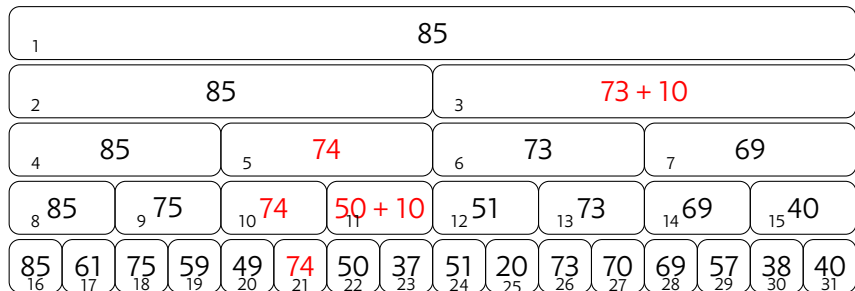
- ▶ Use lazy updating to perform updates to data in  $\mathcal{O}(\log_2 n)$  time.

## Segment Trees, Example from last time.



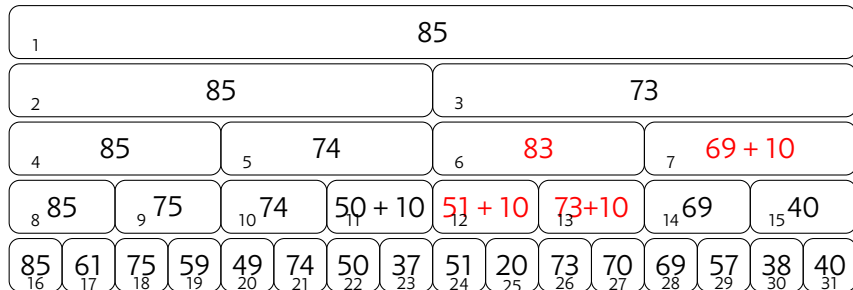
▶ What if we want to add 10 to elements 5 (64) to 15 (40)?

## Segment Trees, Example from last time, pt 2.



- ▶ What if we want to add 10 to elements 5 (64) to 15 (40)?
- ▶ The action is similar to a query.
  - ▶ If a node is completely contained, we annotate it.
  - ▶ If a node is a leaf, we update it.
  - ▶ Propagation occurs whenever a query descends through a node.
- ▶ What if I ask about nodes 8 and 9 (50 and 20) now?

## Segment Trees, Example from last time, pt 3



- ▶ What if I ask about nodes 8 and 9 (50 and 20) now?
  - ▶ Previous update instructions get propagated down the tree....
  - ▶ Answer is 61.

# The Segment Tree Class

- ▶ CP 4 has a much nicer segment tree implementation

```
1 class SegmentTree {
2     private:
3         int n; // size
4         vi A, st, lazy;
5         int l(int p) { return p<<1; } // go left
6         int r(int p) { return (p<<1)+1; } // go right
7
8         int conquer(int a, int b) {
9             if (a == -1) return b; // corner case
10            if (b == -1) return a;
11            return min(a, b); // RMQ
12        }
```

# Building

```
13 void build(int p, int L, int R) { // O(n)
14     if (L == R) st[p] = A[L]; // base case
15     else {
16         int m = (L+R)/2;
17         build(l(p), L , m);
18         build(r(p), m+1, R);
19         st[p] = conquer(st[l(p)], st[r(p)]);
20     }
```

## Code for Searching

```
21 int RMQ(int p, int L, int R, int i, int j) { // O(log n)
22     propagate(p, L, R); // lazy propagation
23     if (i > j) return -1; // infeasible
24     if ((L >= i) && (R <= j)) return st[p]; // found the segment
25     int m = (L+R)/2;
26     return conquer(RMQ(l(p), L, m, i, min(m, j)),
27                   RMQ(r(p), m+1, R, max(i, m+1), j));
28 }
```



## Range Update

```
29 void update(int p, int L, int R, int i, int j, int val) {
30     propagate(p, L, R); // lazy propagation
31     if (i > j) return;
32     if ((L >= i) && (R <= j)) { // found the segment
33         lazy[p] = val; // update this
34         propagate(p, L, R); // lazy propagation
35     } else {
36         int m = (L+R)/2;
37         update(l(p), L, m, i, min(m, j), val);
38         update(r(p), m+1, R, max(i, m+1), j, val);
39         int lsub = (lazy[l(p)] != -1) ? lazy[l(p)] : st[l(p)];
40         int rsub = (lazy[r(p)] != -1) ? lazy[r(p)] : st[r(p)];
41         st[p] = (lsub <= rsub) ? st[l(p)] : st[r(p)]; } }
```

# Propagation

```
42 void propagate(int p, int L, int R) {
43     if (lazy[p] != -1) { // has a lazy flag
44         st[p] = lazy[p]; // [L..R] has same value
45         if (L != R) // not a leaf
46             lazy[l(p)] = lazy[r(p)] = lazy[p]; // propagate down
47         else // L == R, a single index
48             A[L] = lazy[p]; // time to update this
49         lazy[p] = -1; // erase lazy flag
50     } }
```