# Lazy Segment Trees

## CS 491 – Competitive Programming

Dr. Mattox Beckman

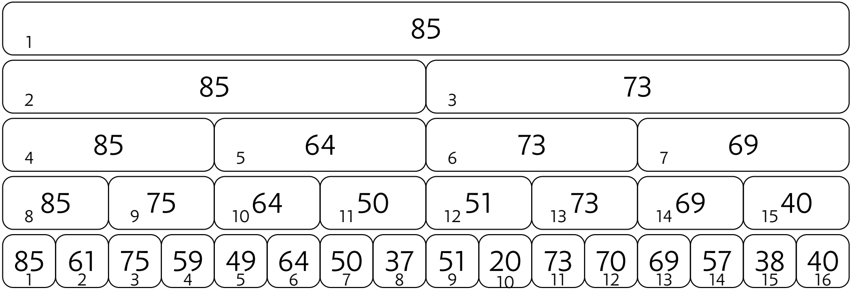
University of Illinois at Urbana-Champaign
Department of Computer Science

Spring 2023

# Objectives

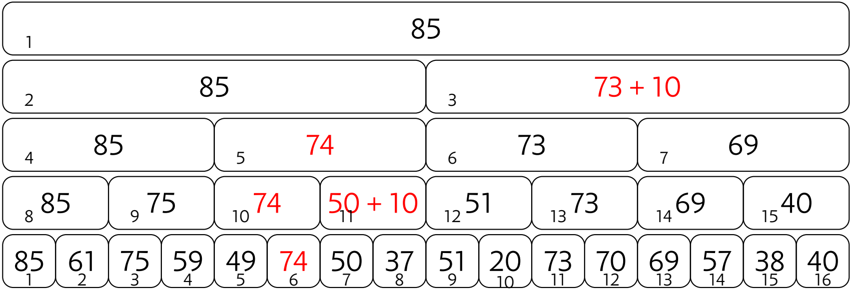
- Use lazy updating to perform updates to data in $\mathcal{O}(\log_2 n)$ time.

# Segment Trees, Example from last time.



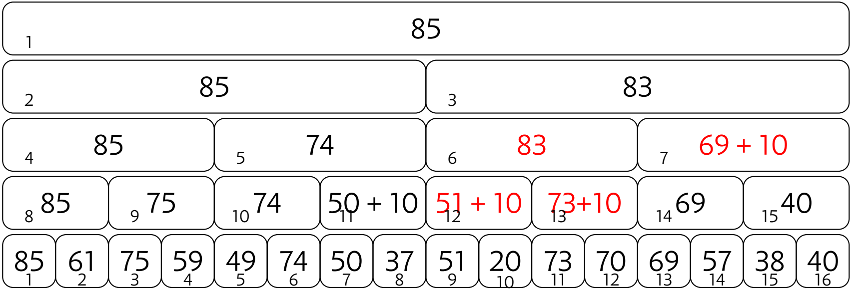

- What if we want to add 10 to elements 6 (64) to 16 (40)?

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




- What if we want to add 10 to elements 6 (64) to 16 (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 decends through a node.
- What if I ask about nodes 9 and 10 (51 and 20) now?

# Segment Trees, Example from last time, pt 3




- What if I ask about nodes 9 and 10 (51 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

```
class SegmentTree {
  private:
    int n; // size
    vi A, st, lazy;
    int l(int p) { return p<<1; } // go left
    int r(int p) { return (p<<1)+1; } // go right

    int conquer(int a, int b) {
      if (a == -1) return b; // corner case
      if (b == -1) return a;
      return min(a, b); // RMQ
    }
```

# Building

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

# Code for Searching

- L and R give you the bounds with respect to the orignal array.
- i and j give you the bounds for the query

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

# Range Update

```
void update(int p, int L, int R, int i, int j, int val) { /
  propagate(p, L, R); // lazy propagation
  if (i > j) return;
  if ((L >= i) && (R <= j)) { // found the segment
    lazy[p] = val; // update this
    propagate(p, L, R); // lazy propagation
  } else {
    int m = (L+R)/2;
    update(l(p), L , m, i , min(m, j), val);
    update(r(p), m+1, R, max(i, m+1), j , val);
    int lsub = (lazy[l(p)] != -1) ? lazy[l(p)] : st[l(p)];
    int rsub = (lazy[r(p)] != -1) ? lazy[r(p)] : st[r(p)];
    st[p] = (lsub <= rsub) ? st[l(p)] : st[r(p)]; } }
```

# Propagation

```
void propagate(int p, int L, int R) {
  if (lazy[p] != -1) { // has a lazy flag
    st[p] = lazy[p]; // [L..R] has same value
    if (L != R) // not a leaf
      lazy[l(p)] = lazy[r(p)] = lazy[p]; // propagate downw
    else // L == R, a single index
      A[L] = lazy[p]; // time to update this
    lazy[p] = -1; // erase lazy flag
  } }
```