Lazy Segment Trees

CS 491 - Competitive Programming

By Andrea Zhou

Last Lecture: Segment Trees

Objectives:

- Build a Segment Tree
- Query a Segment Tree

Consider the following data array

[85, 61, 75, 59, 49, 64, 50, 37]

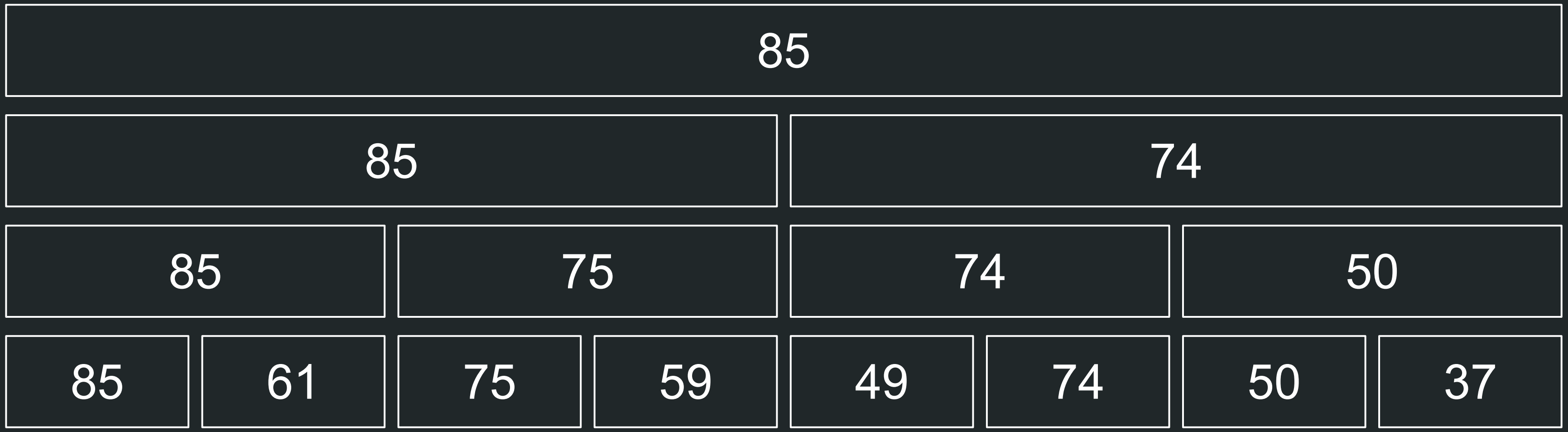
Consider the following data array

[85, 61, 75, 59, 49, 64, 50, 37]

If we want to perform range queries such as

- Find the maximum element in the range [5,6]
- Find the maximum element in the range [0,4]
- Find the maximum element in the range [3,7]
- ...

Segment Tree



85

85 74

85 75 74 50

85 61 75 59 49 74 50 37

But what if we wanted to change some of the data?

- Add 5 to elements in the range [4,7]
- Find the maximum element in the range [5,6]
- Find the maximum element in the range [0,4]
- Subtract 2 from elements in the range [6,6]
- Find the maximum element in the range [3,7]
- ...

But what if we wanted to change some of the data?

- Add 5 to elements in the range [4,7]
- Find the maximum element in the range [5,6]
- Find the maximum element in the range [0,4]
- Subtract 2 from elements in the range [6,6]
- Find the maximum element in the range [3,7]
- …

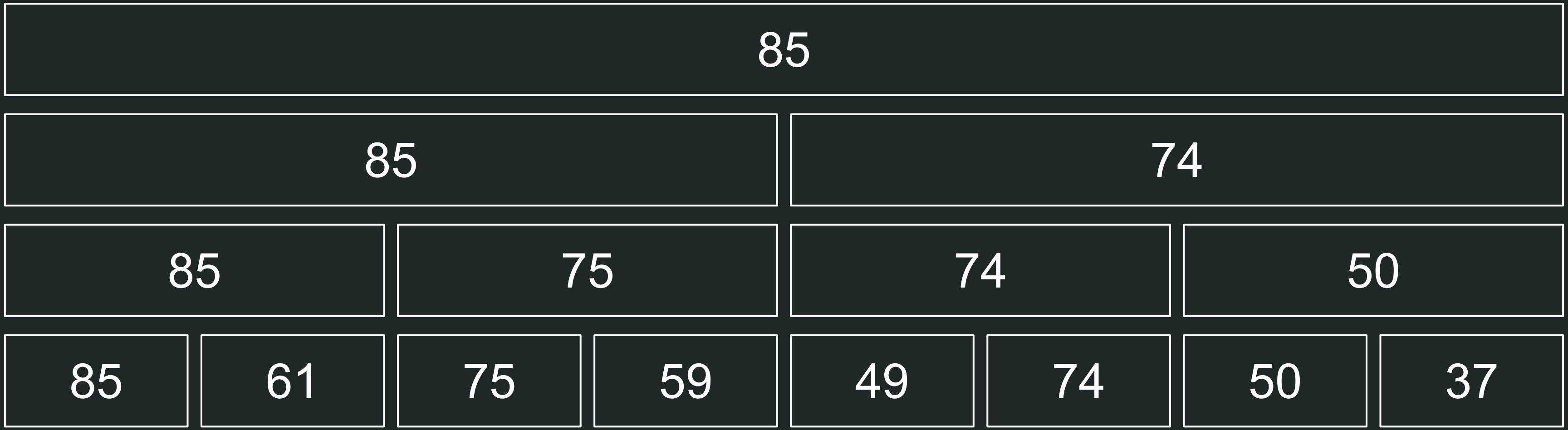
← might not return correct
max without updating ST

Today’s Lecture: Lazy Propagation in Segment Trees

Objectives:

- Perform updates to data while still being able to query the segment tree in log(n) time

Segment Tree



85

85 74

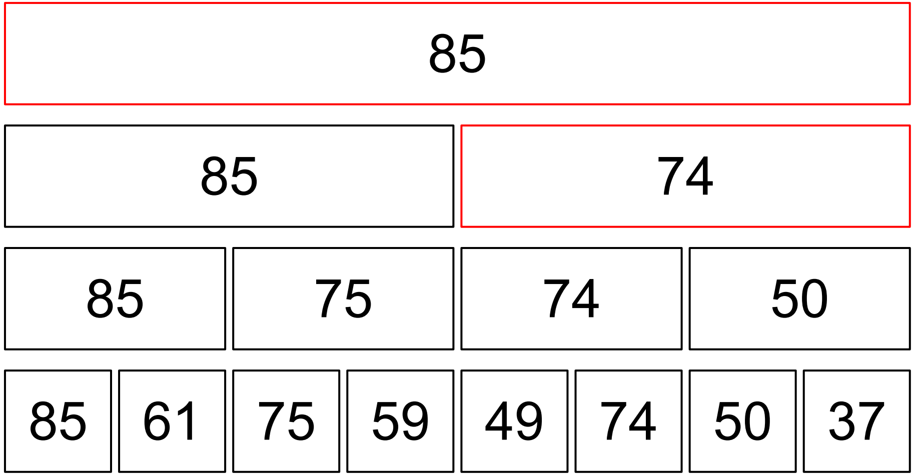
85 75 74 50

85 61 75 59 49 74 50 37

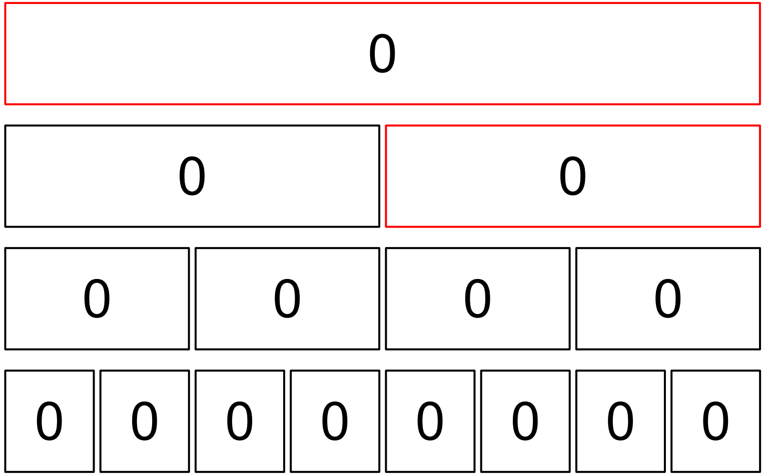
1. Update: Add 5 to elements in the range [4,7]
2. Query: Find the maximum element in the range [5,6]
3. …

1. “Add 5 to elements in the range [4,7]”

Segment Tree

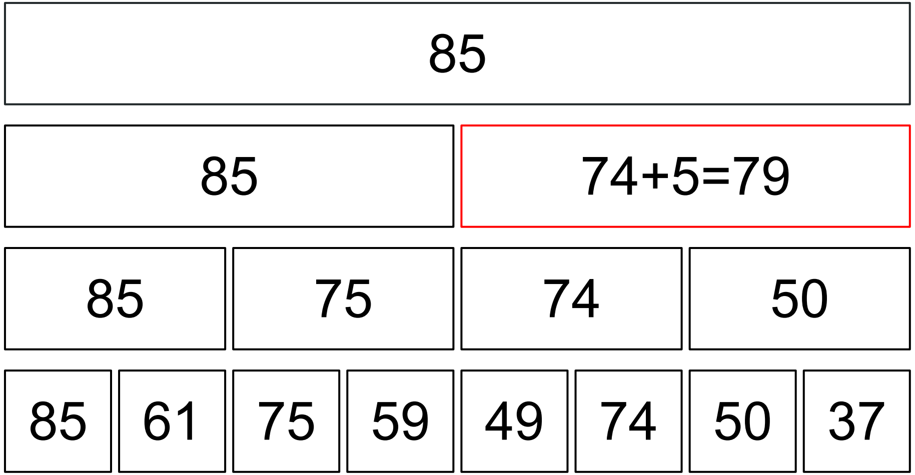


Lazy Tree



1. “Add 5 to elements in the range [4,7]”

Segment Tree



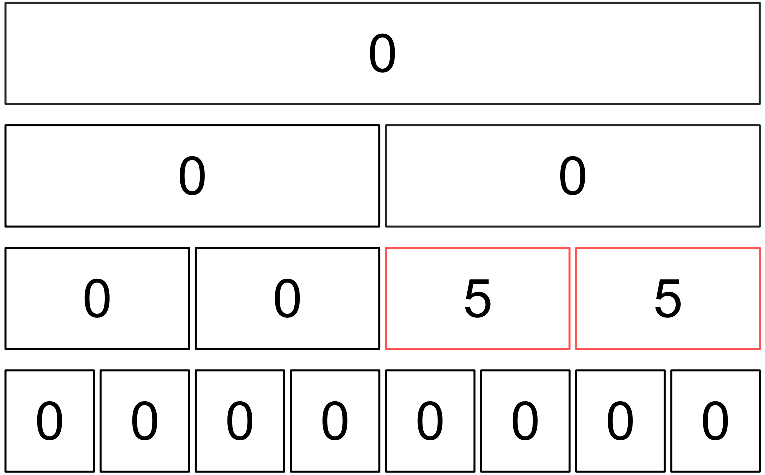
85

85 | 74+5=79

85 | 75 | 74 | 50

85 | 61 | 75 | 59 | 49 | 74 | 50 | 37

Lazy Tree



0

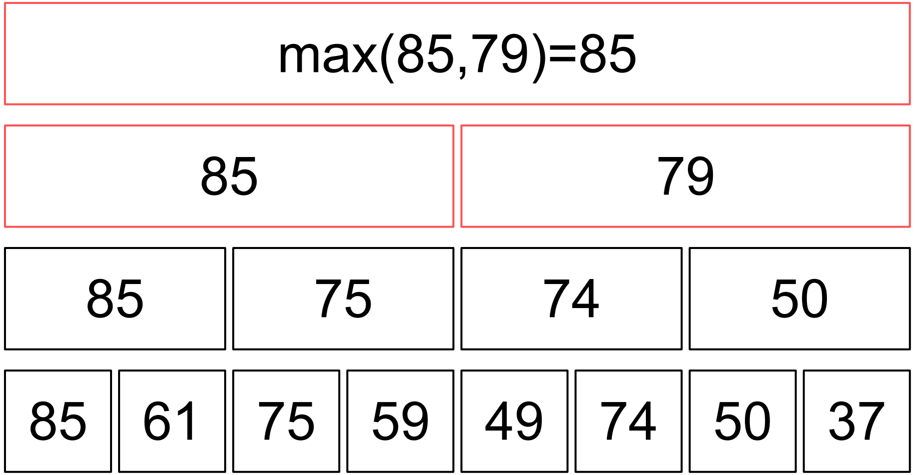
0 | 0

0 | 0 | 5 | 5

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

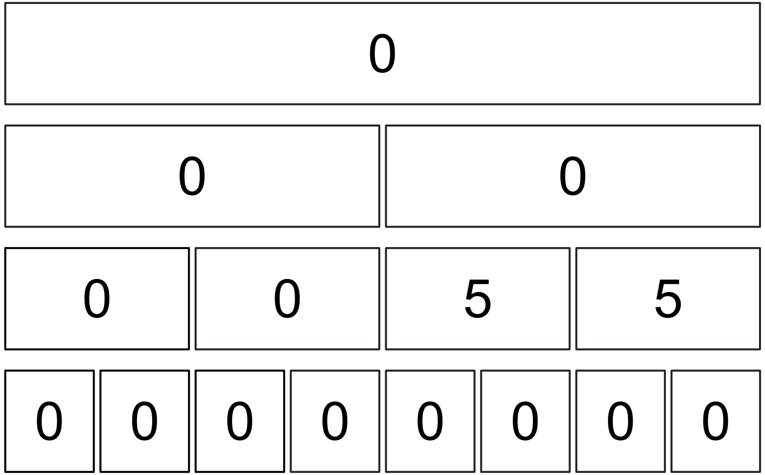
1. “Add 5 to elements in the range [4,7]”

Segment Tree



<table>
<tr><td colspan="8">max(85,79)=85</td></tr>
<tr><td colspan="4">85</td><td colspan="4">79</td></tr>
<tr><td colspan="2">85</td><td colspan="2">75</td><td colspan="2">74</td><td colspan="2">50</td></tr>
<tr><td>85</td><td>61</td><td>75</td><td>59</td><td>49</td><td>74</td><td>50</td><td>37</td></tr>
</table>

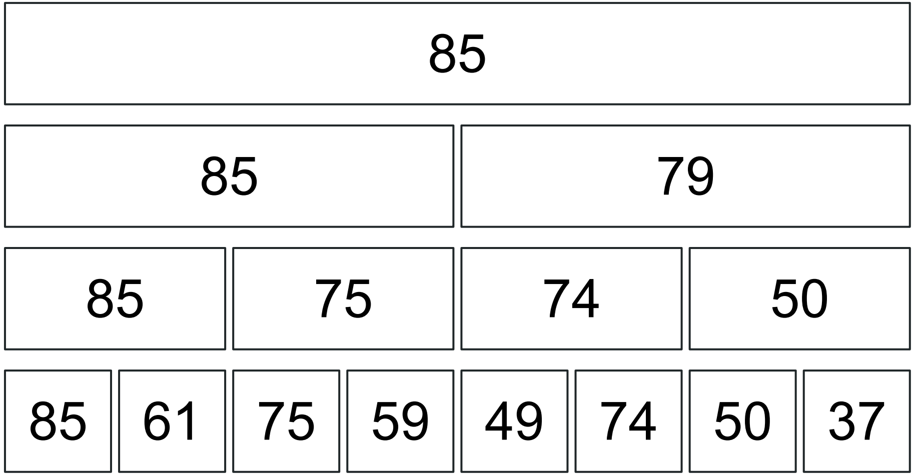
Lazy Tree



<table>
<tr><td colspan="8">0</td></tr>
<tr><td colspan="4">0</td><td colspan="4">0</td></tr>
<tr><td colspan="2">0</td><td colspan="2">0</td><td colspan="2">5</td><td colspan="2">5</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

2. “Find the maximum element in the range [5,6]”

Segment Tree



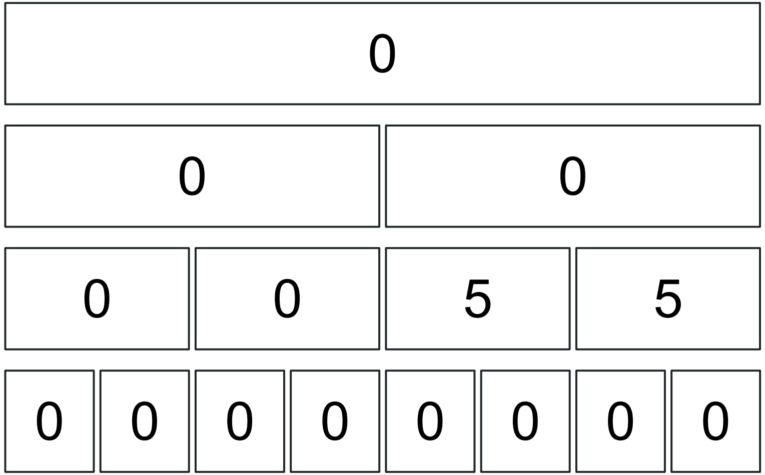
85

85 79

85 75 74 50

85 61 75 59 49 74 50 37

Lazy Tree



0

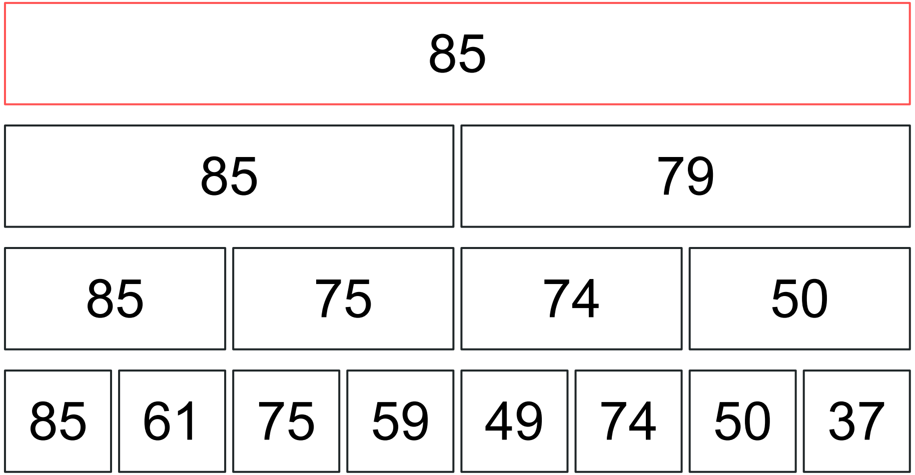
0 0

0 0 5 5

0 0 0 0 0 0 0 0

2. “Find the maximum element in the range [5,6]”

Segment Tree



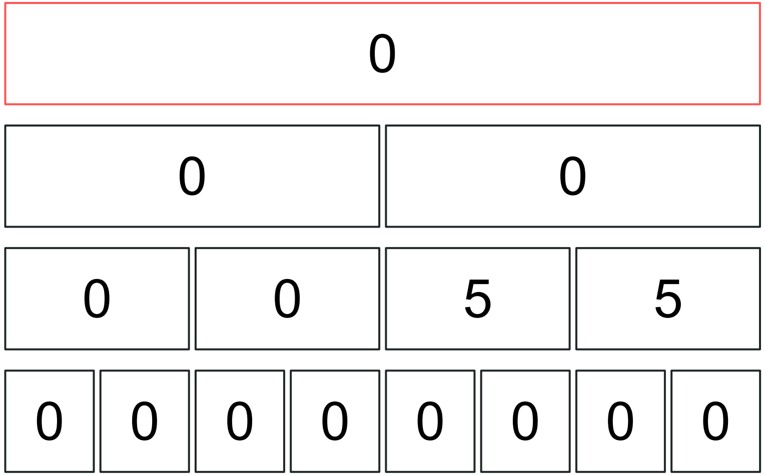
85

85 79

85 75 74 50

85 61 75 59 49 74 50 37

Lazy Tree



0

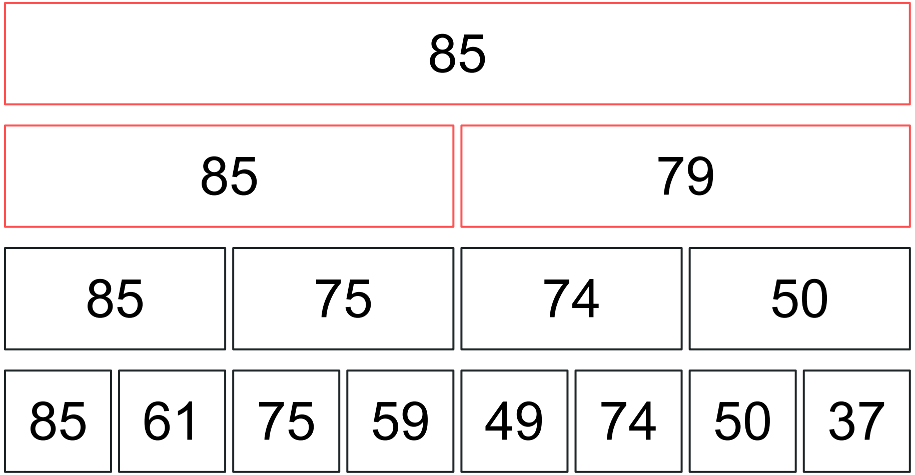
0 0

0 0 5 5

0 0 0 0 0 0 0 0

2. “Find the maximum element in the range [5,6]”

Segment Tree



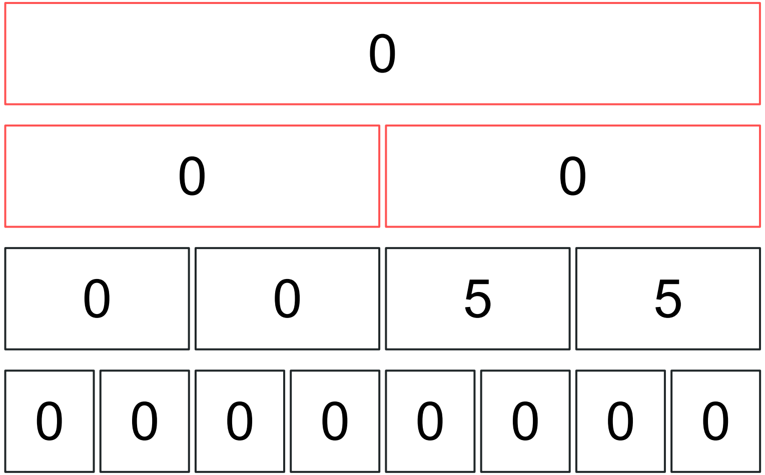
85

85 | 79

85 | 75 | 74 | 50

85 | 61 | 75 | 59 | 49 | 74 | 50 | 37

Lazy Tree



0

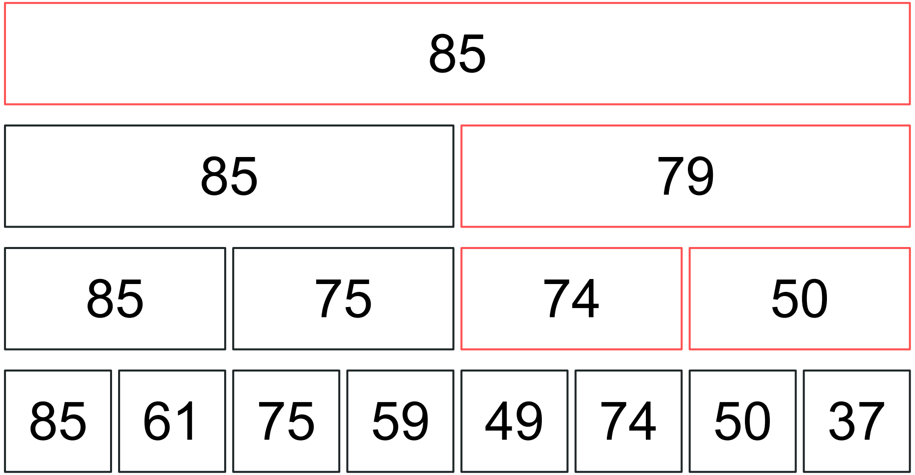
0 | 0

0 | 0 | 5 | 5

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

2. “Find the maximum element in the range [5,6]”

Segment Tree



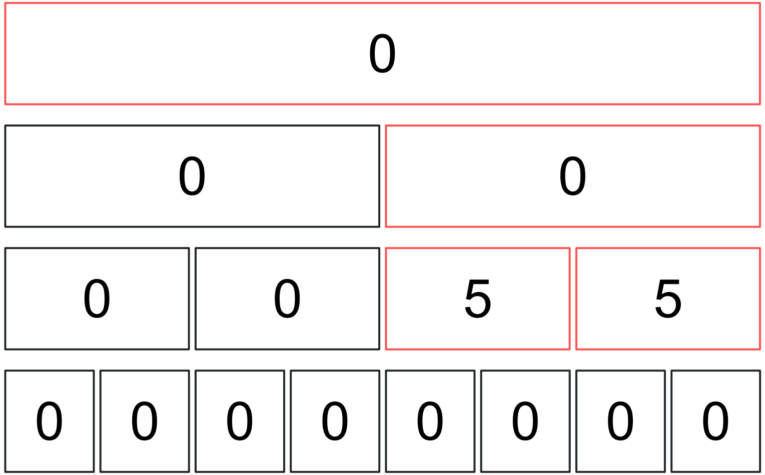
85

85 79

85 75 74 50

85 61 75 59 49 74 50 37

Lazy Tree



0

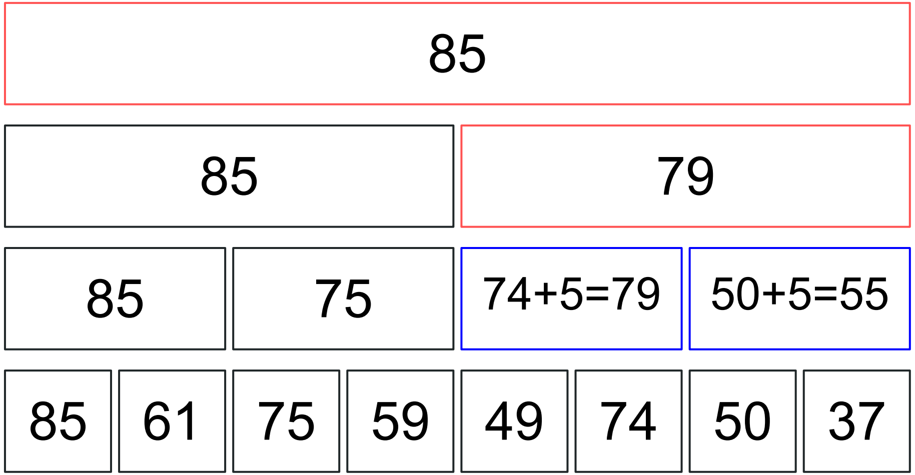
0 0

0 0 5 5

0 0 0 0 0 0 0 0

2. “Find the maximum element in the range [5,6]”

Segment Tree



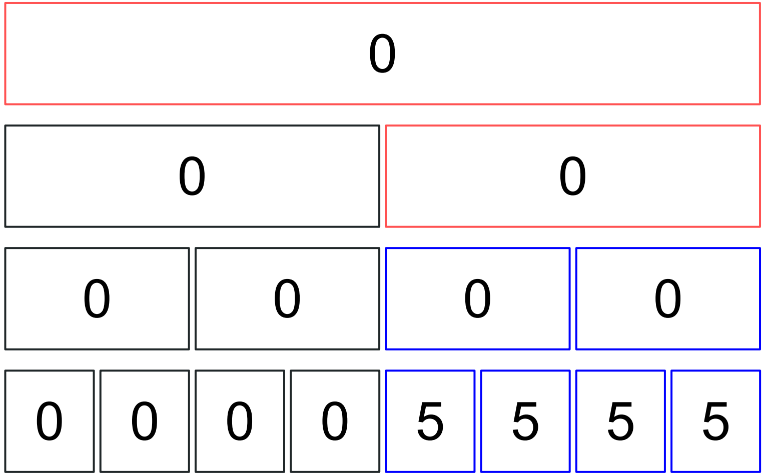
85

85 | 79

85 | 75 | 74+5=79 | 50+5=55

85 | 61 | 75 | 59 | 49 | 74 | 50 | 37

Lazy Tree



0

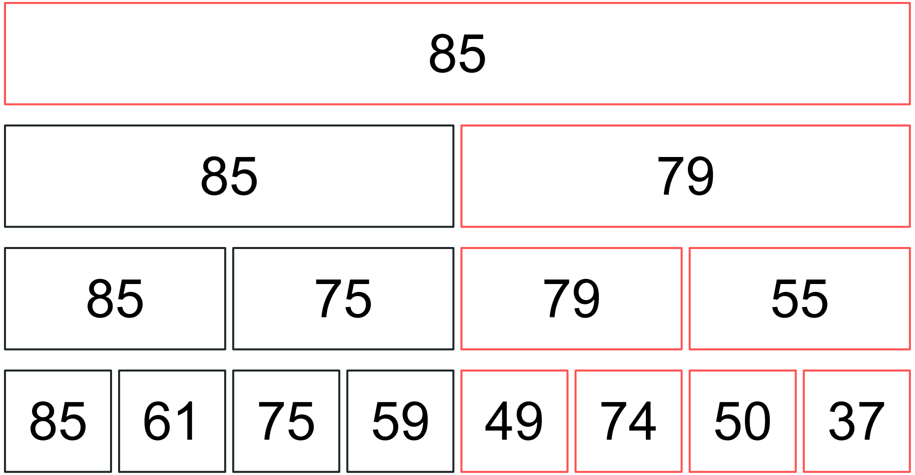
0 | 0

0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 5 | 5 | 5 | 5

2. “Find the maximum element in the range [5,6]”

Segment Tree



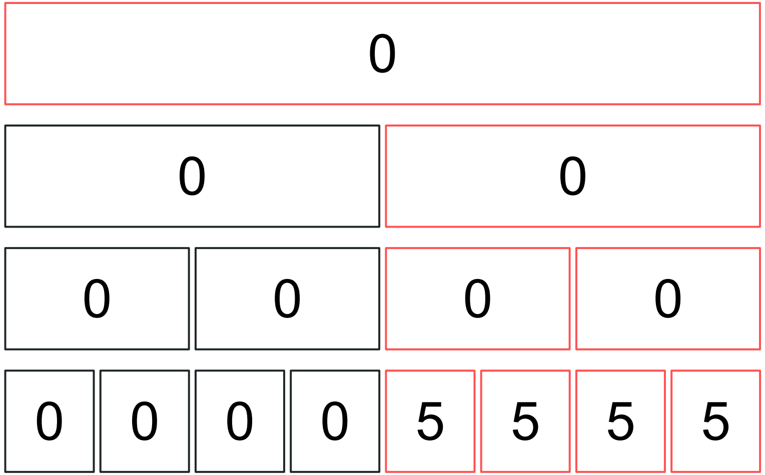
85

85 79

85 75 79 55

85 61 75 59 49 74 50 37

Lazy Tree



0

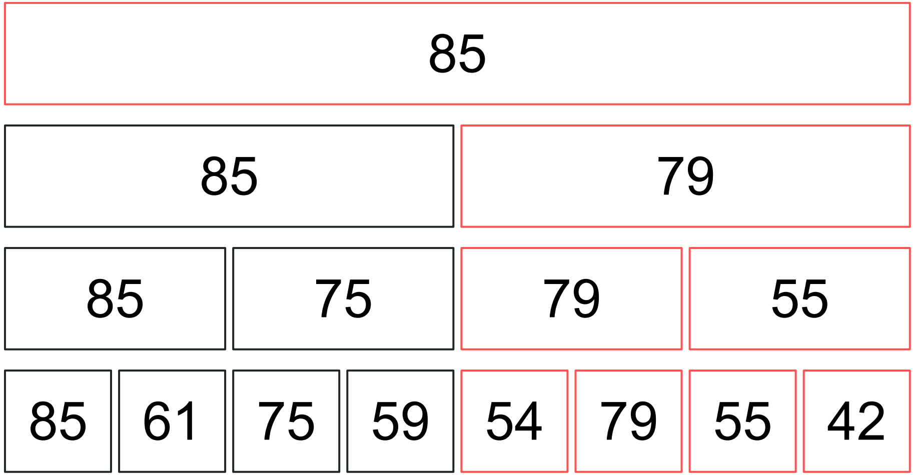
0 0

0 0 0 0

0 0 0 0 5 5 5 5

2. “Find the maximum element in the range [5,6]”

Segment Tree



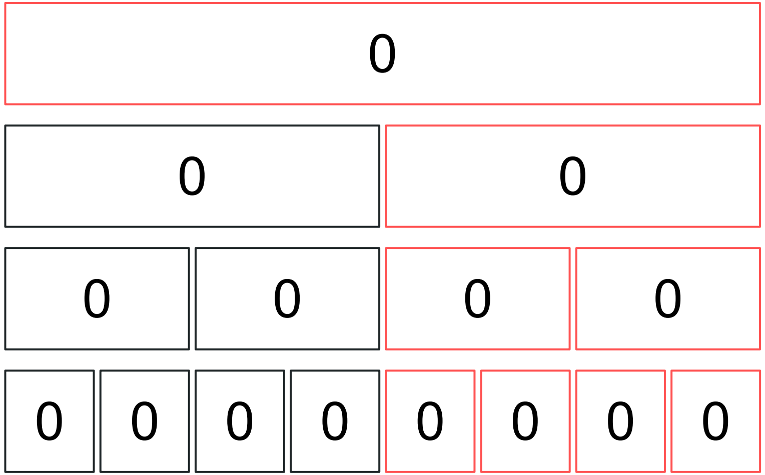
85

85 79

85 75 79 55

85 61 75 59 54 79 55 42

Lazy Tree



0

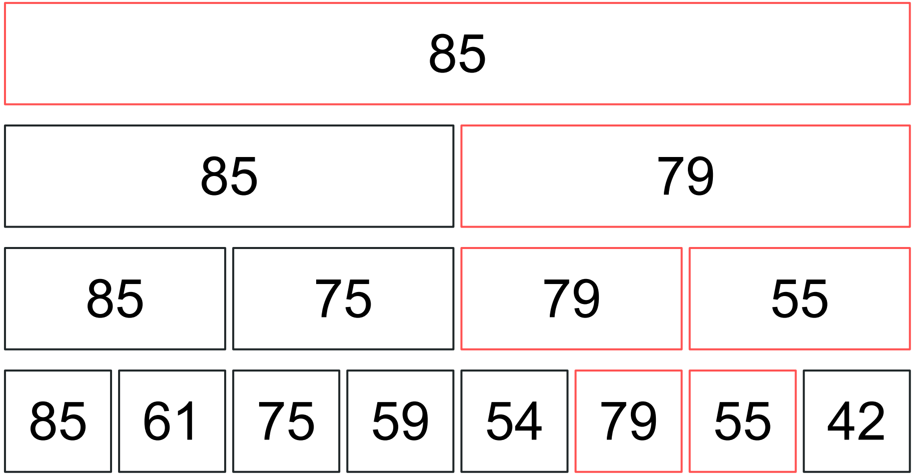
0 0

0 0 0 0

0 0 0 0 0 0 0 0

2. “Find the maximum element in the range [5,6]”

Segment Tree



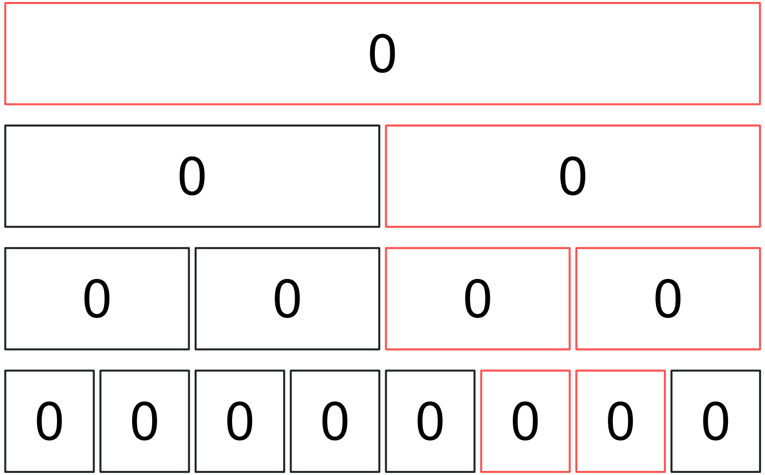
85

85 79

85 75 79 55

85 61 75 59 54 79 55 42

Lazy Tree



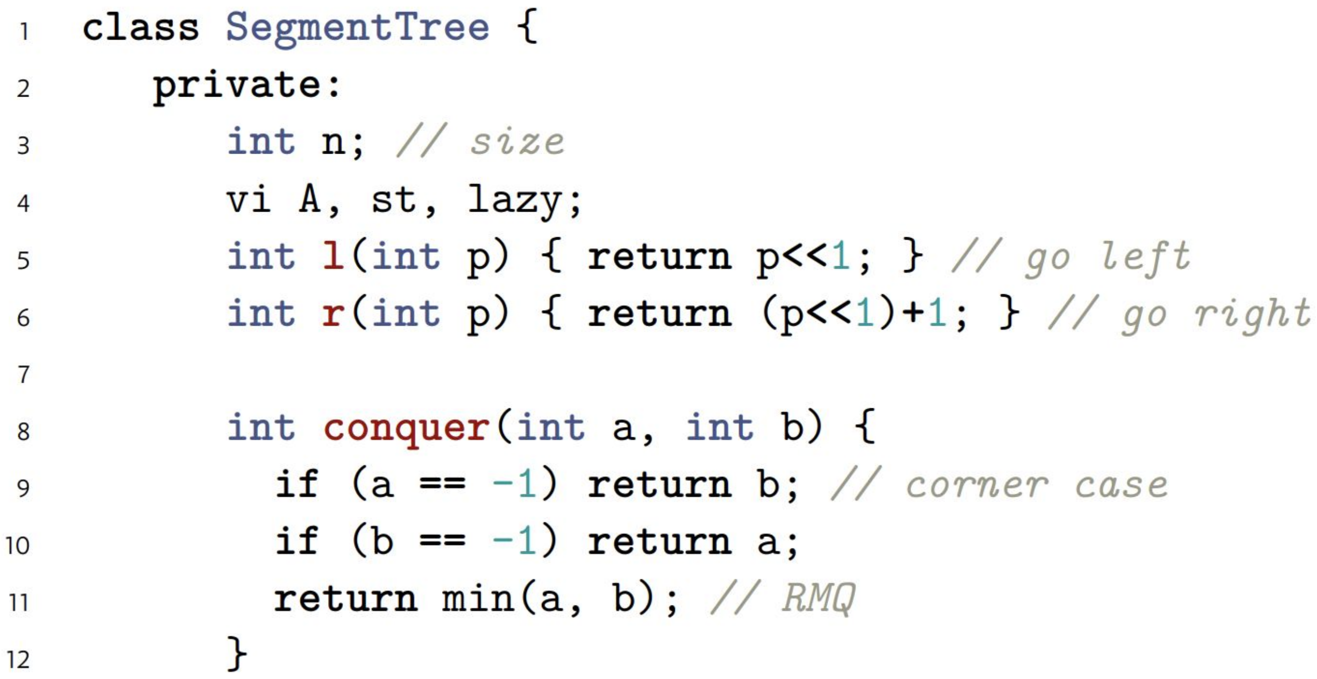
0

0 0

0 0 0 0

0 0 0 0 0 0 0 0

CP4 Segment Tree Class

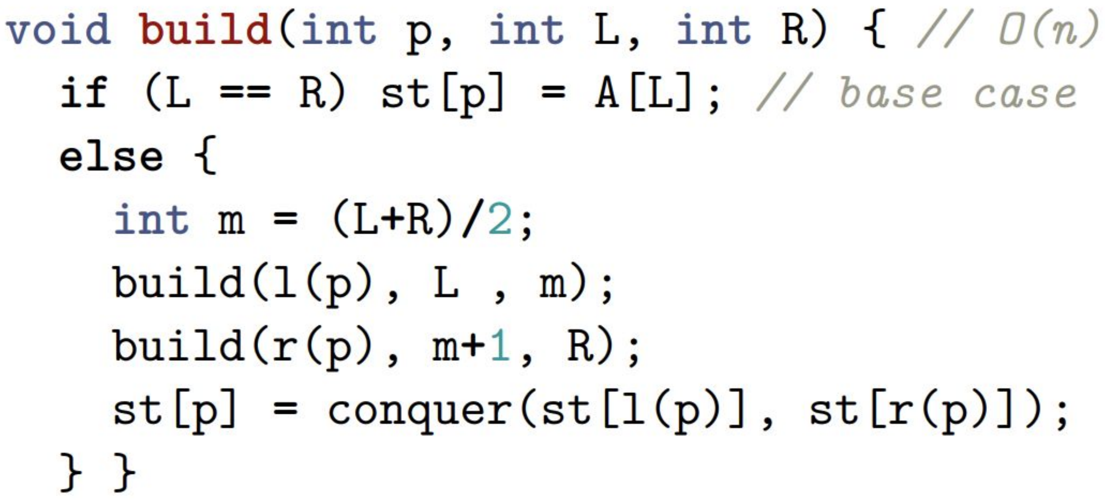


```cpp
class SegmentTree {
  private:
    int n; // size
    vi A, st, lazy;
    int l(int p) { return p<<1; } // go left
    int r(int p) { return (p<<1)+1; } // go right

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

Code provided from Dr. Mattox Beckman

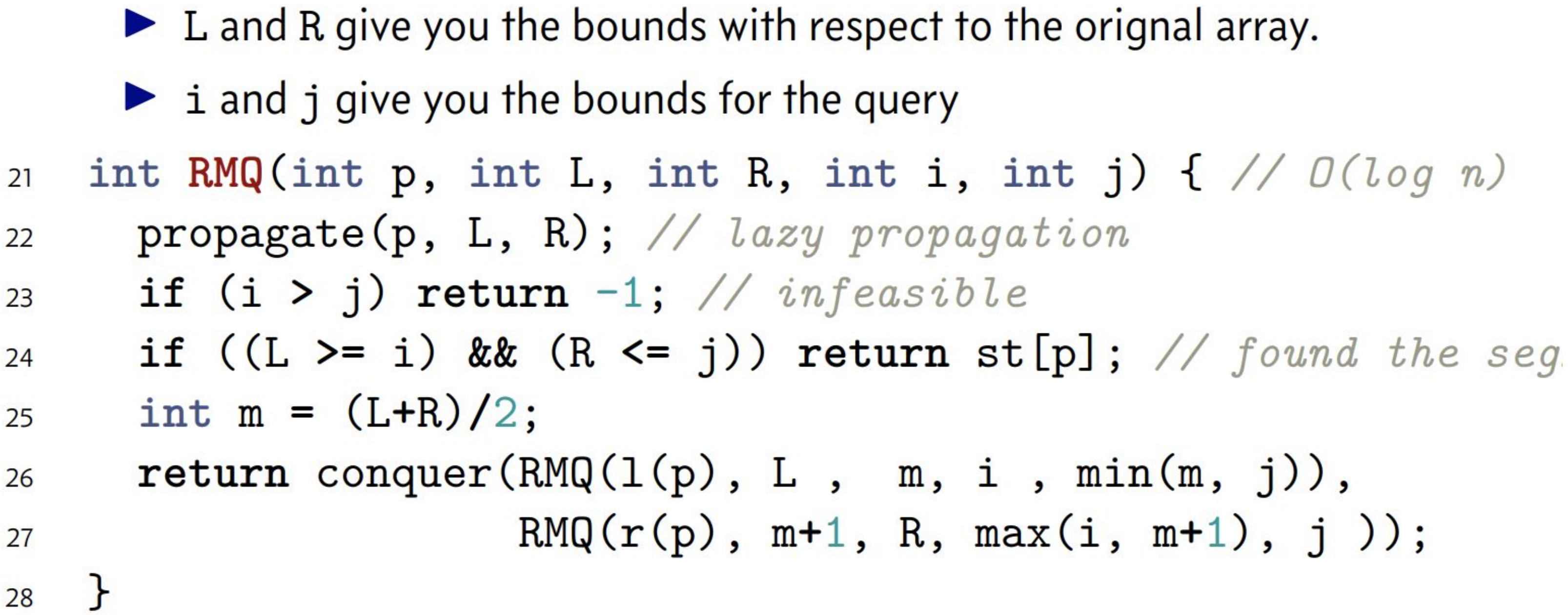
Building the Segment Tree



```
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 provided from Dr. Mattox Beckman

Querying

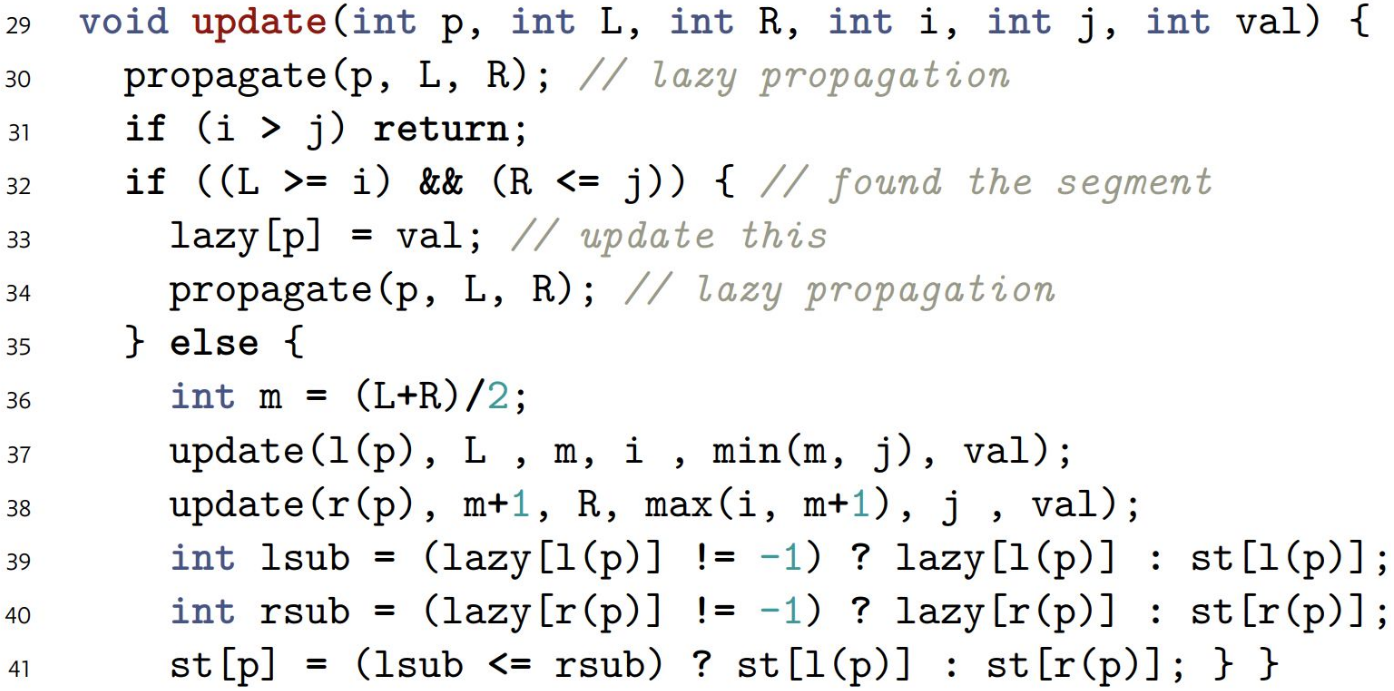


- L and R give you the bounds with respect to the orignal array.
- i and j give you the bounds for the query

```
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 ));
}
```

Code provided from Dr. Mattox Beckman

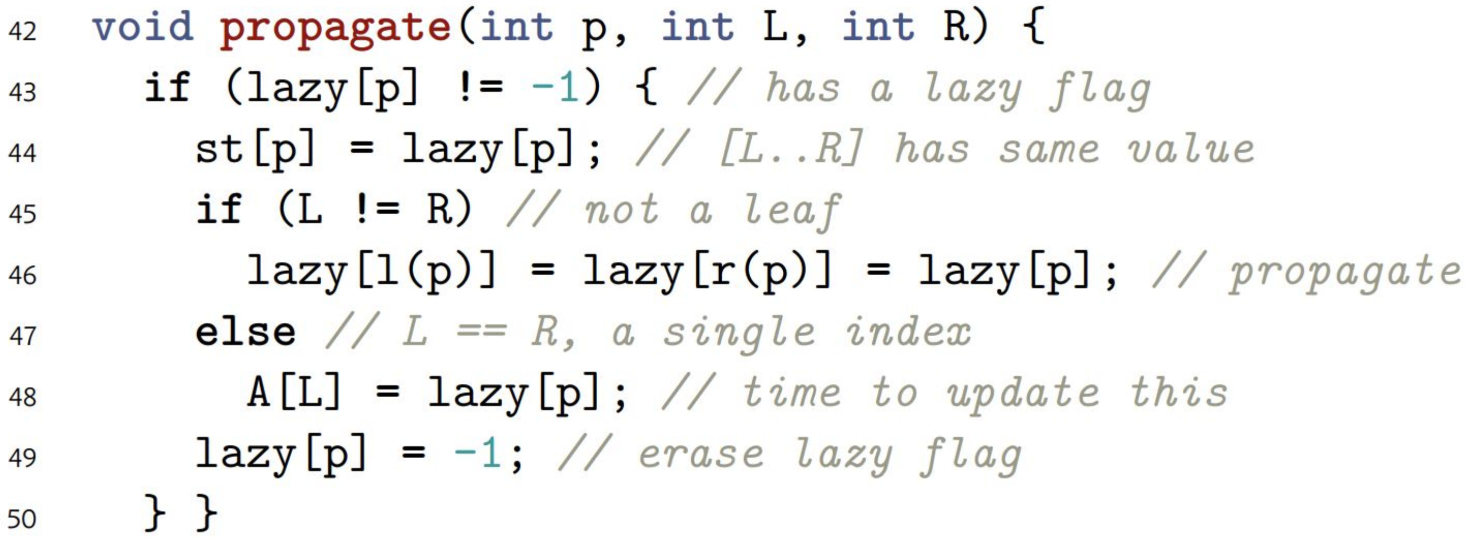
Updating



```
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)]; } }
```

Code provided from Dr. Mattox Beckman

Propagating



```
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
    else // L == R, a single index
      A[L] = lazy[p]; // time to update this
    lazy[p] = -1; // erase lazy flag
} }
```

Code provided from Dr. Mattox Beckman