Introduction to Segment Trees

Course: https://unacademy.com/a/i-p-c-intermediate-track

tanujkhattar@

Objective

- Point Update and Range Query problem
- Introduction to Segment Trees
    - Discuss via examples
    - Build
    - Query
    - Update
- Sparse Segment Trees
    - Definition
    - Update
    - Query
- Conclusion

Point Updates and Range Queries

1e5 * 1e5

$\boxed{1e10}$

1s 2s

- Given an array A of N elements, support two types of operations: 1e5
  - Point Update: Given i, x set A[i] = x. ✓ Set.
  - Range Query: Given [L, R] return Sum(A[i]), L <= i <= R. ⟸ [L, R]

1e5 ←

* Arrays.

Point Update : O(1) ✓

Range Query : O(N) ✓

⟹ O(log N)

* Prefix Sums.

Point Update : O(N) ✓ -> Recompute the whole PS.

Range Query : O(1) ✓ PS[R] - PS[L-1]

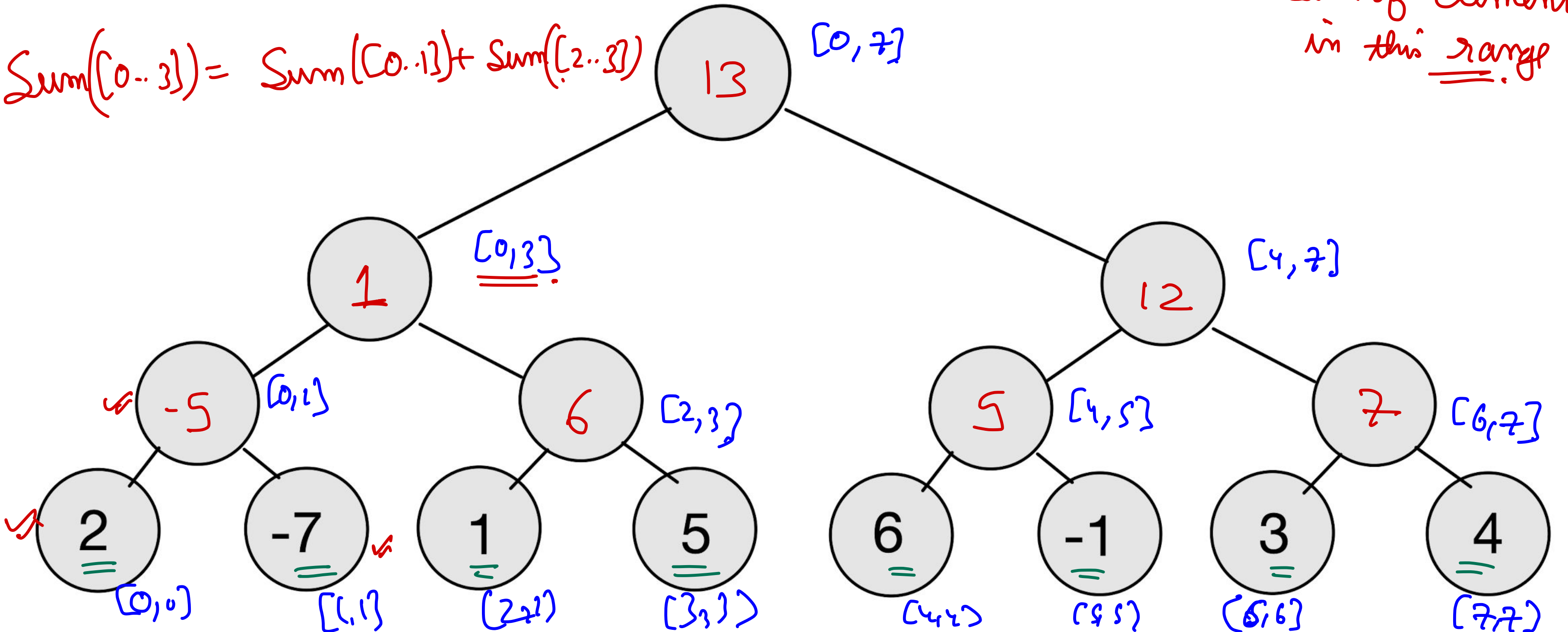
A: | 2 | -7 | 1 | 5 | 6 | -1 | 3 | 4 |

Introduction to Segment Trees

- A binary tree with each leaf corresponding to an element in the array.
- Every internal node represents a range in the original array.

Range Sum. ⇓ Sum of elements in this range.

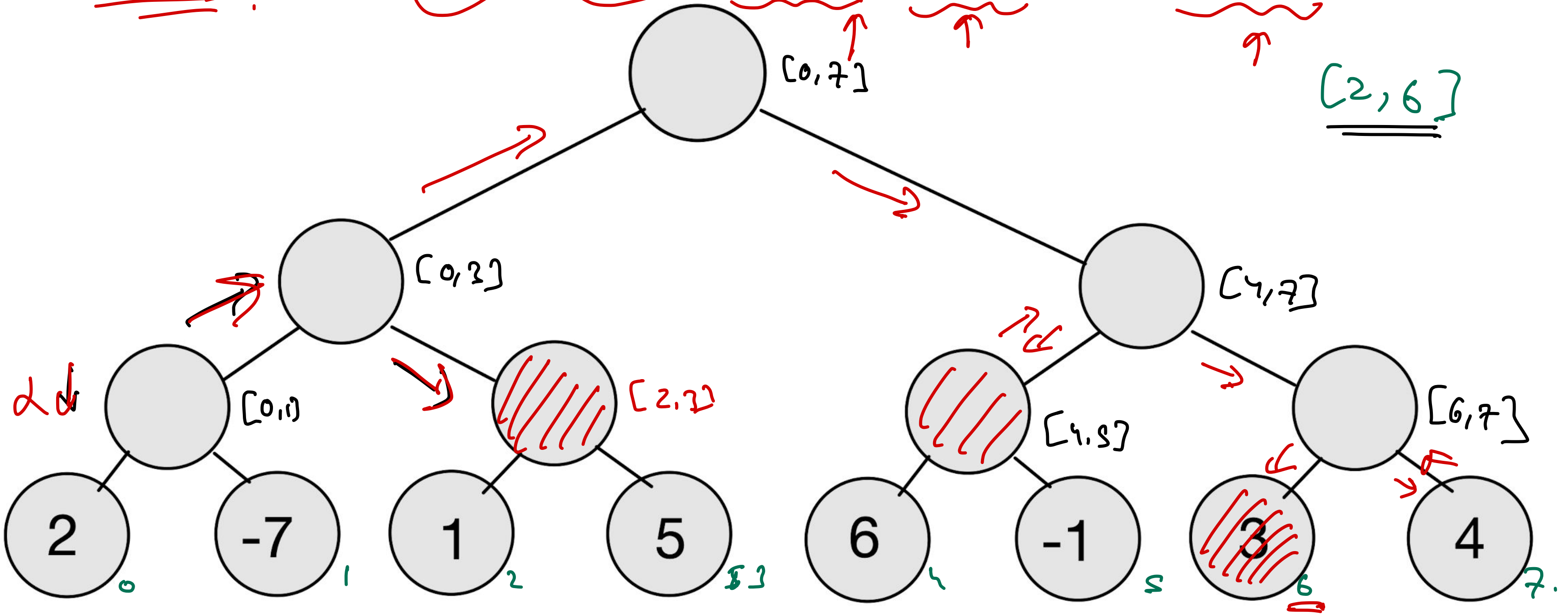
Sum([0..3]) = Sum([0..1]) + Sum([2..3])



Introduction to Segment Trees (eg: sum)

- Every internal node should store the “answer” for the range.
- Any range [L, R] can be broken down into at most LogN ranges
- Final answer = CombineAnswer(Range_1, Range_2, ..., Range_LogN)

“Segment Tree” ranges.

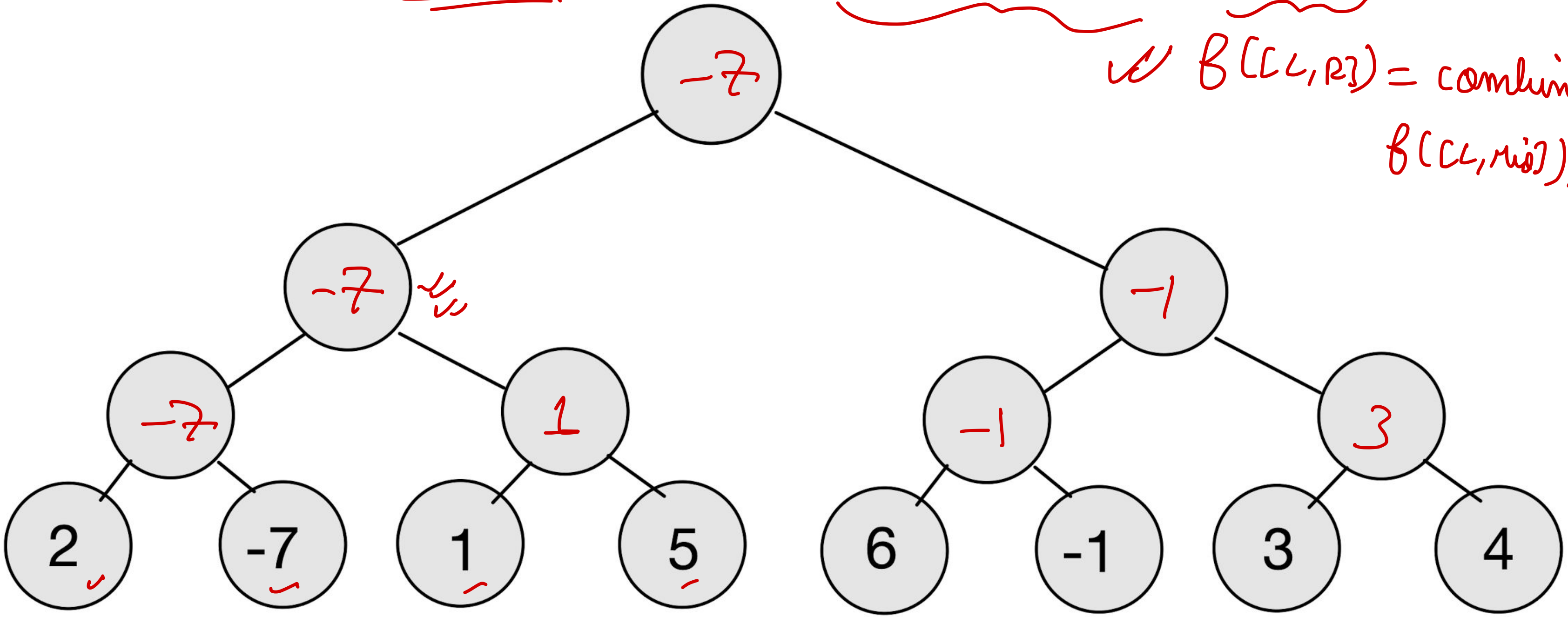


Introduction to Segment Trees (eg: min)

Every internal node will min of the range

- Every internal node should store the “answer” for the range.
- Any range [L, R] can be broken down into at most LogN ranges.
- Final answer = CombineAnswer(Range_1, Range_2, ..., Range_LogN)

f([L,R]) = combine(f([L,mid]), f([mid,R]))



Introduction to Segment Trees

A complete Binary has N − 1 internal nodes for N leaf nodes.

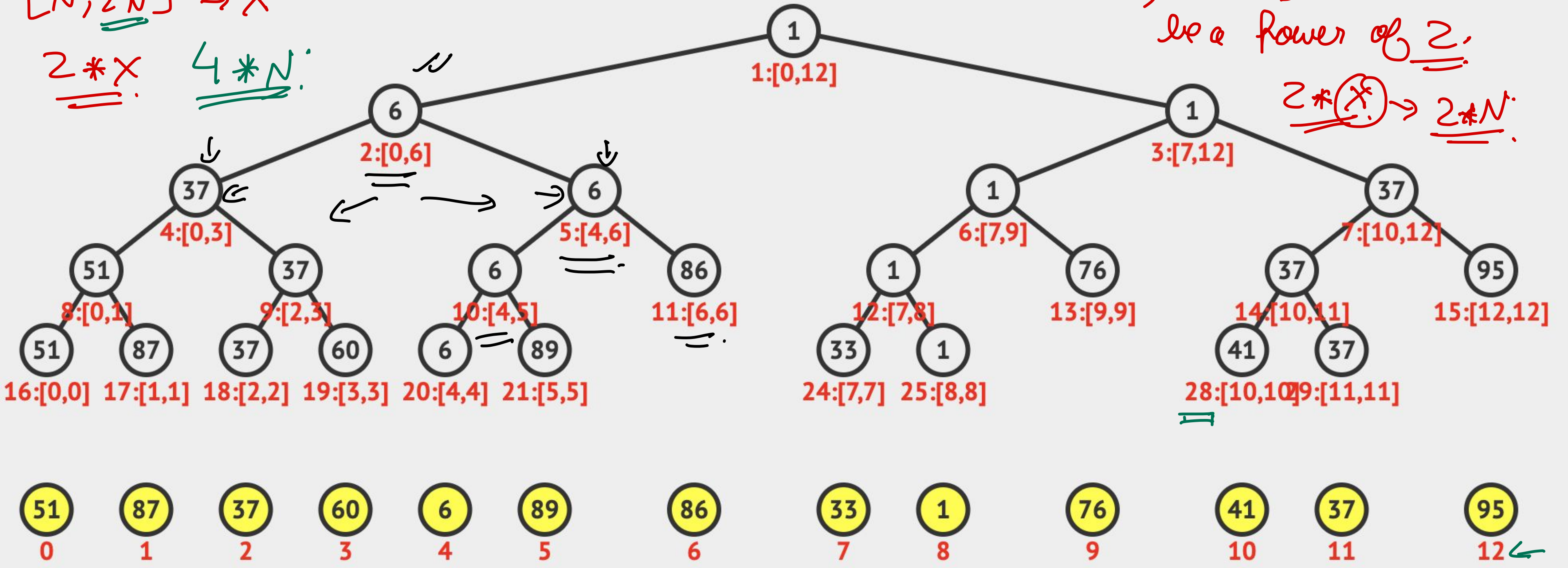
- See https://visualgo.net/en/segmenttree for more visualisations.
- **Q:** What is the tightest upper bound on number of nodes in a Segment Tree over array of length N? **A)** N **B)** 2 * N **C)** 4 * N **D)** N^2

[N, 2N] → X

2 * X 4 * N.

[N, 2 * N] → There will be a power of 2.

2 * (X) → 2 * N.



Build

- Takes O(N) time because ST has O(N) nodes and each node is visited once.

```
int ST[4 * N], A[N];
#define lc (x << 1)
#define rc (x << 1)|1
void build(int x = 1, int l = 1, int r = N + 1) {
  if (l == r - 1) return void(ST[x] = A[l]);
  int mid = (l + r) / 2;
  build(lc, l, mid);
  build(rc, mid, r);
  ST[x] = combine(ST[lc], ST[rc]);
}
```

[l, r) ∈ x.

→ left child. → 2x

→ Right child. 2x+1

x=1, 2*x, 2x+1

[5, 6)

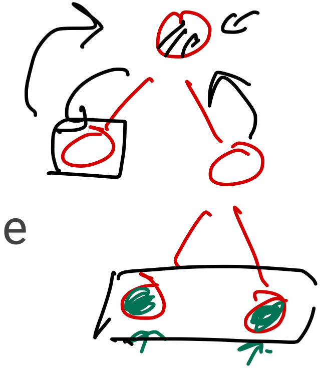
[l, mid)   [mid, r)

x : [l, r)
lc : [l, mid)
rc : [mid, r)

RightAns => min (Left ans, RightAns)

∞

Query

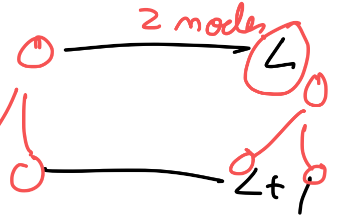


- Start with the root node, and for every node check whether the range represented by this node lies completely within the Query Range
- If yes, return the answer stored at this node.
- If no, recursively fetch the answer from left and right child of the node, combine and return the complete answer.
- **Claim:** The query runs in O(logN) time.

```
int query(int L, int R, int x = 1, int l = 1, int r = N - 1) {
1) if (l >= R || r <= L) return 0;   If no intersection, return 0
2) if (l >= L && r <= R) return ST[x];
3) int mid = (l + r) / 2; }
   return combine(query(L, R, lc, l, mid), query(L, R, rc, mid, r));
}
```

+ / min / max. ⇐ return 0 will change.

$N, N/2, N/2^2 \ldots$ $N/2^{\log N} \approx 1$

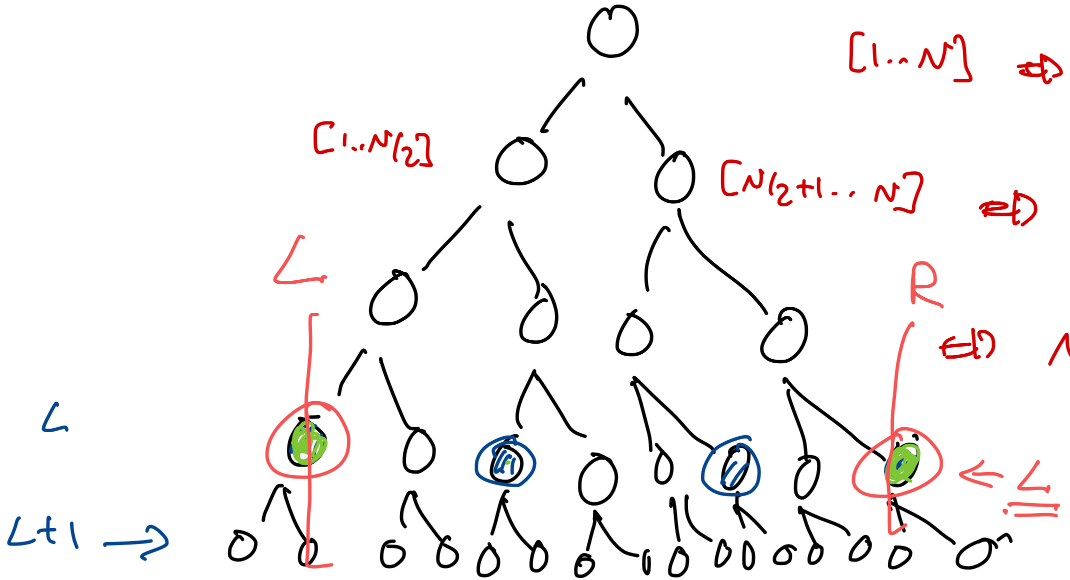


Visit 4 nodes.

Query Complexiy

Claim: The query runs in O(logN) time.

Proof: The Segment tree has O(logN) levels and at every level, we expand at-most 2 nodes (leftmost and rightmost). Therefore, we visit at-most 4 nodes at any level. Since time spent per node is O(1), total time is O(logN).



[1..N] ⇒ N elements

O(logN) Levels

⇒ N/2 element

⇒ N/4 elem.

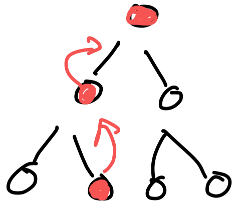
1 element.

$4 * \log N$

O(logN)

Point Update

O(log N)



O(log N)

- Takes O(logN) time since only nodes lying on a path from root to affected leaf will get affected. Hence no. of affected nodes are only O(logN).

```
void point_update(int pos, int val, int x = 1, int l = 1, int r = N - 1) {
 if (pos < l || pos >= r) return;
 if (l == r - 1) {
    ST[x] = val;
    A[pos] = val;
    return;
 }
 int mid = (l + r) / 2;
 update(pos, val, lc, l, mid);
 update(pos, val, rc, mid, r);
 ST[x] = combine(ST[lc], ST[rc]);
}
```

Have you already reached a leaf node?

Sparse Segment Trees

- Let A be an empty array of 1e9 ([1, 1e9]) elements, initially all 0. Let there be Q (<= 1e5) queries of the form:
  - Point Update: Given pos, v - set A[pos] = v (1 <= pos <= 1e9)
  - Range Query: Given [L, R] - return Sum(A[i]), L <= i <= R.

Way-1 Coordinate Compression (Offline)

- Since number of distinct positions (updated or queried) is bounded by the input size (2 * Q), we can read all queries offline and map the integers to range [1, 2e5]
- Works only if processing the queries offline is allowed.

Way-2: Sparse Segment Trees

- Allocate the segment tree nodes only when needed (i.e. during a point update).
- During a Query, if a child doesn't exist, the range represented by that child is 0.
- Need total Qlog(MAX) nodes in the tree, where MAX is the size of the range.

Way-2: Sparse Segment Trees

```
int L[Q * LOGN], R[Q * LOGN], ST[Q * LOGN], blen;
// sparse segtree. range sum, initially 0
int update(int pos, int add, int l, int r, int id) {
  if (pos < l || pos >= r) return id;
  if (!id) id = ++blen;
  if (l == r - 1) {
    ST[id] += add;
    return id;
  }
  int m = l + (r - l) / 2;
  L[id] = update(pos, add, l, m, L[id]);
  R[id] = update(pos, add, m, r, R[id]);
  ST[id] = combine(ST[L[id]], ST[R[id]]);
  return id;
}
```

Conclusion

- There are many more variations in segment trees.
  - Lazy Segment Trees for range updates
  - Merge Sort Trees
  - 2D Segment Trees.
  - Persistent Segment Trees
  - Etc.
- Segment trees are really powerful and are one of the most used DS in Competitive Programming.