Range Queries
Class 1/2
Segment Trees

Range Queries

1e5 2e5

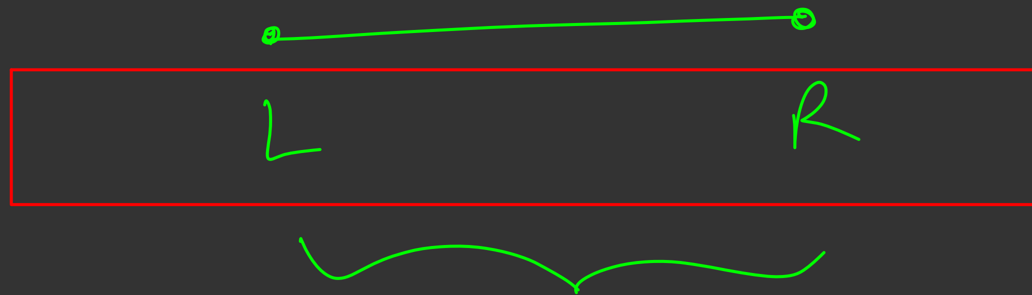
- **Example problem:** Given an array A of N (N<=1e5) elements and Q (Q<=1e5) queries, In each query you will be given two indices L and R, you need to output the sum of values of the array A from index L to R.

- **Example:**

  N=4, Q=1, A[]={3, 5, 2, 8}

  Query 1: 2 3

  Output: 7



$O(n)$

$Q \to 10^5$

$O(Q \cdot N)$

$pref(i) = sum(0 - i)$

$L, R \rightarrow pref(R) - pref(L-1)$

Brute force

Prefix Sum

1 2 3 1 2 1

1 3 6 7 9 10

$pref(R) - pref(L-1)$

$O(1)$

$O(1)$

$O(n)$

$O(Q)$

$O(n+Q)$

Range Queries

- This can be solved using a prefix sum array. But what if we have an update operation in the problem?

- **New statement:** Given an array A of N (N<=1e5) elements and Q (Q<=1e5) queries, In each query, you have to do one of two types of operations.

  Operation 1: You will be given two indices L and R, you need to output the sum of values of the array A from index L to R.

  Operation 2: You will be given index Pos and a value Val, you have to change the value at index Pos to the value Val. That is set A[Pos]=Val.

4 5 6 R

L

Q1 →

Q2 → Update

Q3 →

Q4 →

Range Queries

- **Example:**

  N=4, Q=3, A[]={3,5,2,8}

  Query 1: 1 2 3

  Output: 7

  Query 2: 2 2 10

  Query 3: 1 2 3

  Output: 12

  How to do this?

3 10 2 8

Let’s Learn Segment Trees!

- A segment tree can be used to solve this kind of problems!

- **Segment Tree:** Segment Tree is basically a binary tree used for storing intervals or segments. Each node in the Segment Tree represents an interval.

Leaf Nodes are the elements of the input array. Each internal node represents some merging of the leaf nodes. For this problem merging will mean addition.

- **Building the segment tree:**

An array representation of tree is used to represent Segment Trees. For each node at index i, the left child is at index 2*i, right child at 2*i+1.

As all the nodes have 2 children, the segment tree will be a full binary tree with N leaf nodes and N-1 internal Nodes. So the total number of nodes in the segment tree will be 2*N-1.

But if N is not a power of two, there will be some unused nodes. In that case, the approximate size of the segment tree is 4*N. So the segment tree array will be an array of size 4*N.

We start with the segment A[0 . . . n-1]. And every time we divide the current segment into two halves(if it has not yet become a segment of length 1), and then call the same procedure on both halves, and for each such segment, we store the sum in the corresponding node.

Segment Tree

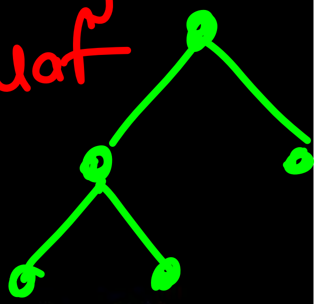
- **Code:**

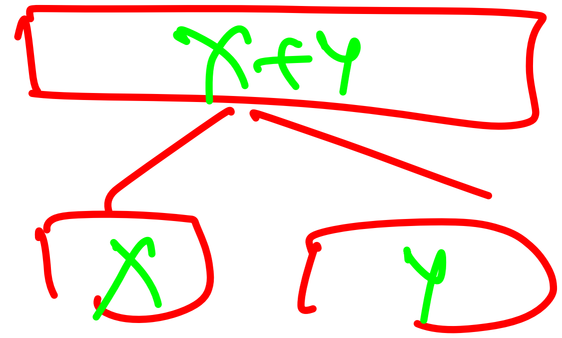
```
#define mxn 200005
int n, arr[mxn];
ll seg[4*mxn];

void build(int node, int st, int en){
    if(st==en){
        seg[node]=arr[st];
        return;
    }
    int mid=(st+en)/2;
    build(2*node,st,mid);
    build(2*node+1,mid+1,en);
    seg[node]=seg[2*node]+seg[2*node+1];
}
```

4 · maxN

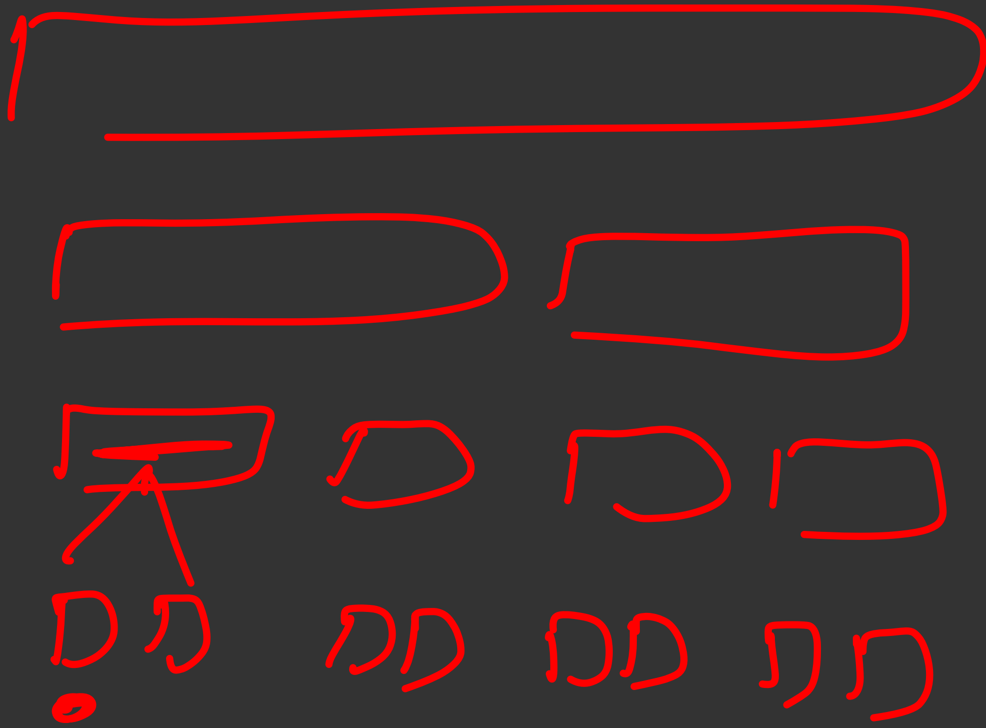
checking for leaf

start==end





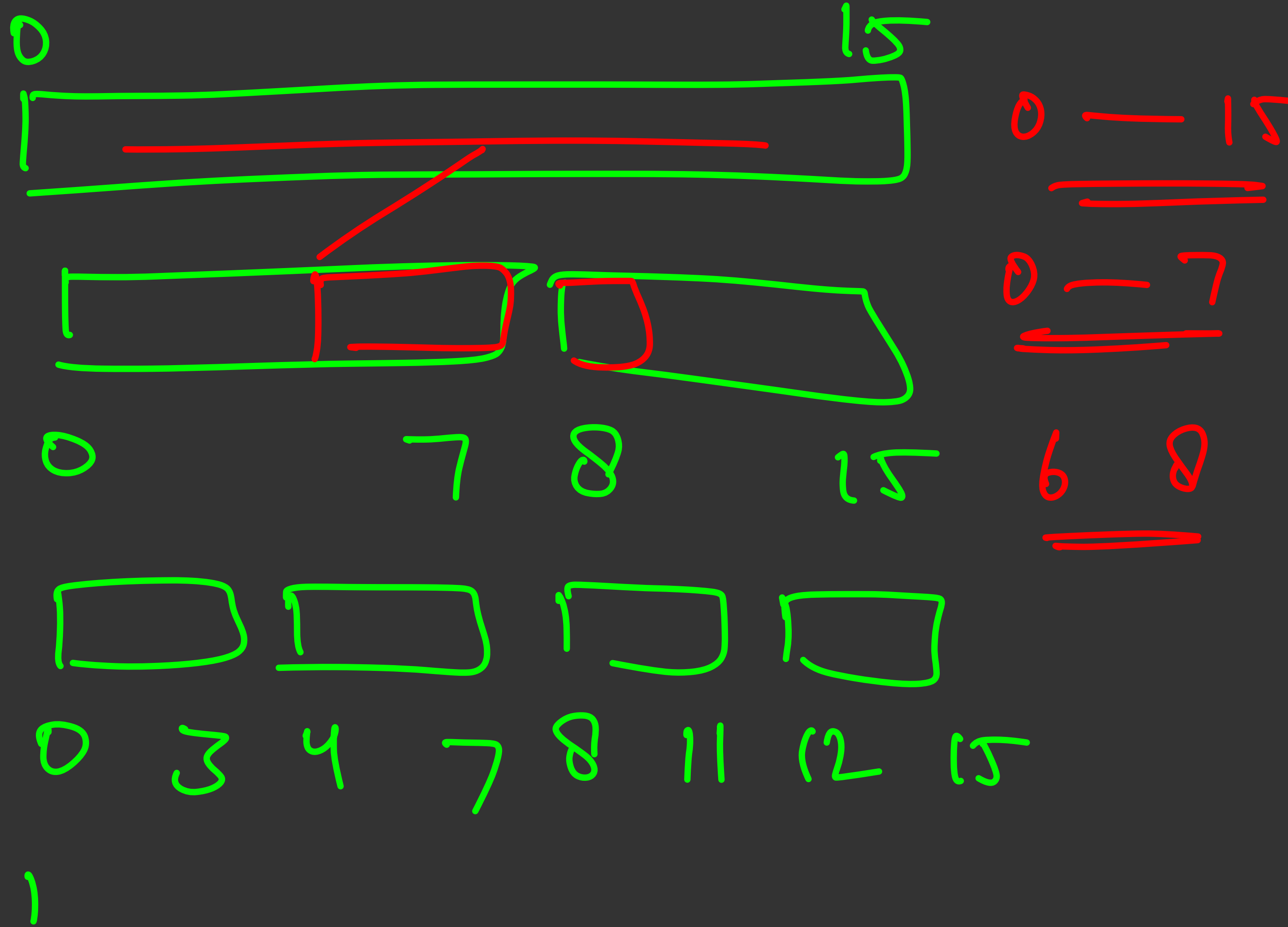
- **Time Complexity:** O(N) (We visit all the 2*N-1 nodes once)



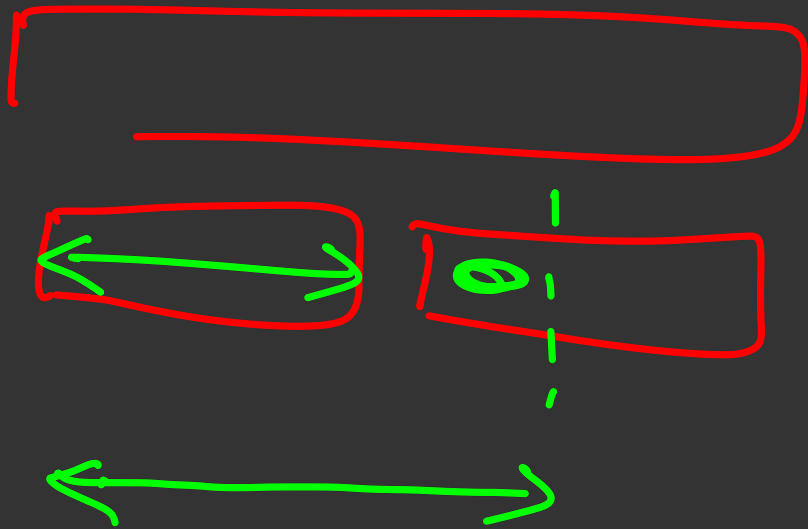
Sum(current node) = arr(start)

Dfs → O(n)

n


0
15
0 — 15
0 — 7
6
8
0
7
8
15
0
3
4
7
8
11
12
15
1

$O(1)$

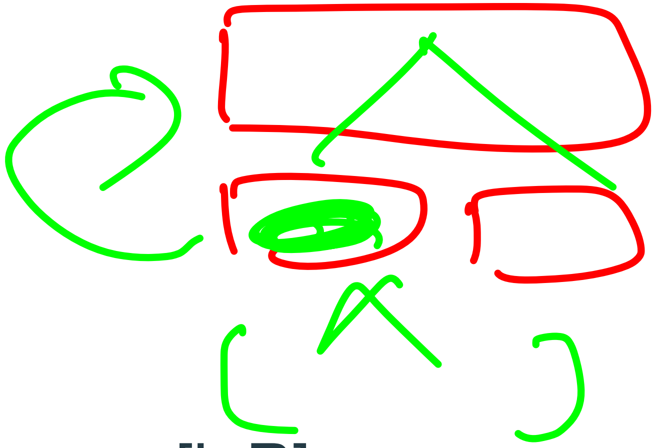


Y, N

L, R

$O(1)$

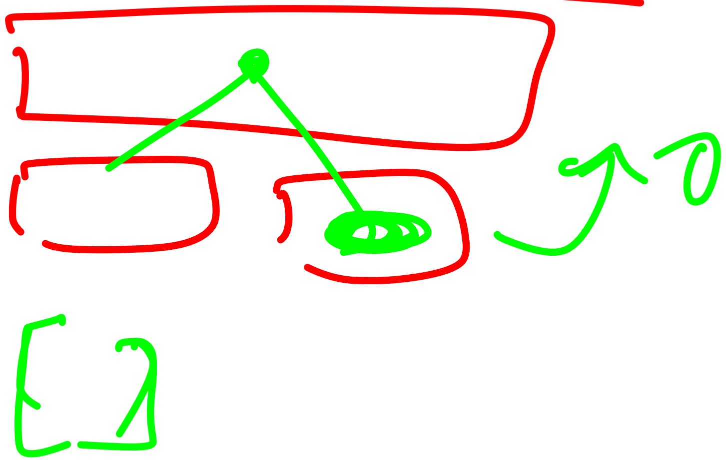
Segment Tree



- **Querying:**

To make a query on a segment tree on the range [L,R], we recurse on the tree starting from the root and check if the interval represented by the node is completely in the range from L to R.

If the interval represented by a node is completely in the range from L to R, return that nodes value.



Segment Tree

- **Code:**

```
ll query(int node, int st, int en, int l, int r){
    if(st>=l && en<=r) return seg[node];
    if(en<l orr st>r) return 0;
    int mid=(st+en)/2;
    return query(2*node, st, mid, l, r)
           +query(2*node+1, mid+1, en, l, r);
}
```

1 → completely overlapping

O(logn).4

O(4logn) → O(logn)

- **Time Complexity:** O(log(N)) (At each level, we will visit at most 4 nodes)

Segment Tree

Build , Update, Range Query

- **Updating in a segment tree:**

O(n) O(log n) O(log n)

We can update the value of an index of the array using a segment tree. Like the segment tree construction and query operations, the update can also be done recursively.

Each level of a Segment Tree forms a partition of the array. Therefore an element A[Pos] only contributes to one segment from each level. Thus only O(log(N)) vertices need to be updated.

O(Q log n + N)

Segment Tree

- **Updating in a segment tree:**

We start the update function from the root. We recursively call the update function with one of the two child vertices (the one that contains A[Pos] in its segment).

When we get to the leaf node, we update A[Pos] with Val and return. And after that the vertices in the path to this leaf node recomputes their corresponding sum value, similar to how it is done in the build function (that is as the sum of its two children).

Segment Tree

- **Code:**

```
void update(int node, int st, int en, int pos, int val){
    if(st==en){
        seg[node]=arr[pos]=val;
        return;
    }
    int mid=(st+en)/2;
    if(pos<=mid) update(2*node, st, mid, pos, val);
    else update(2*node+1, mid+1, en, pos, val);
    seg[node]=seg[2*node]+seg[2*node+1];
}
```

- **Time Complexity:** O(log(N))

Segment Tree

Range min, max Fenwick

Range gcd, xor & or

- **Other Queries:**

Some other types of range queries can be solved using a segment tree as well. For example, range minimum/maximum queries, range GCD queries etc.

1. Range minimum/maximum: For querying range minimum/maximum, we will store the minimum/maximum of its two children in the corresponding node. Similarly we will take minimum/maximum of its two children when updating and querying.

Segment Tree

- **Other Queries:**

  2. Range GCD: For querying range GCD, we will store the GCD of its two children in the corresponding node. Similarly we will take GCD of its two children when updating and querying.

  3. Index of minimum/maximum element query: For this we can store a pair of values for each node, which will correspond to the minimum/maximum in that segment and the index of that value, when merging two nodes, the pair for the current node will be the pair among its two children which has the minimum/maximum value.

Some More Problems

- **Problem 1:**

  Given an array A of N (N<=2e5) elements and a list of positions, your task is to remove elements from the list at given positions and report the removed elements.

  Example:

  N=5, A[]={2, 6, 1 ,4 ,2}, list[]={3, 1, 3, 1, 1}

  Output: 1 2 2 6 4

  Any ideas?

Some More Problems

- **Problem 2:**

  Second Maximum in a Range Queries

  Any ideas?