Segment Trees & Binary Indexed Trees (Fenwick Trees) in DSA: Beginner to Advanced Guide

Subtitle: Learn how to efficiently handle range queries and updates on arrays for advanced problem solving.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Segment Trees and Binary Indexed Trees (Fenwick Trees) are advanced data structures used for **range queries and dynamic updates**. They are essential in problems like **range sum, minimum/maximum queries, and interval updates**.

Step 1: Segment Tree Basics

- **Complete binary tree representation** of array segments
- Each node stores information about a **segment** (sum, min, max)
- **Operations:**
  - Build tree → O(n)
  - Query → O(log n)
  - Update → O(log n)

Example: Array [1,3,5,7,9,11] → segment tree stores sums for ranges

Step 2: Building a Segment Tree

```
function buildTree(arr) {
  let n = arr.length;
  let tree = Array(2*n).fill(0);
  for(let i=0;i<n;i++) tree[n+i] = arr[i];
  for(let i=n-1;i>0;i--) tree[i] = tree[i<<1] + tree[i<<1|1];
  return tree;
}
```

Step 3: Range Sum Query

```
function query(tree, n, left, right) {
  left += n; right += n;
```

```
  let sum = 0;
  while(left <= right) {
    if(left % 2 === 1) sum += tree[left++];
    if(right % 2 === 0) sum += tree[right--];
    left = left>>1; right = right>>1;
  }
  return sum;
}
```

Step 4: Update Element

```
function update(tree, n, index, value) {
  index += n; tree[index] = value;
  while(index > 1) {
    index = index >> 1;
    tree[index] = tree[index<<1] + tree[index<<1|1];
  }
}
```

Step 5: Binary Indexed Tree (Fenwick Tree) Basics

- Supports **prefix sum queries** efficiently
- Array-based, uses **least significant bit (LSB)**
- **Operations:**
  - Update → O(log n)
  - Prefix sum query → O(log n)

Example:

```
function updateBIT(BIT, n, index, val) {
  for(let i=index;i<=n;i+=i&-i) BIT[i]+=val;
}


function queryBIT(BIT, index) {
```

```
  let sum = 0;

  for(let i=index;i>0;i-=i&-i) sum+=BIT[i];

  return sum;

}
```

Step 6: Common Problems Using Segment Trees & BIT

1. Range sum / range minimum queries
2. Dynamic array updates
3. Inversion count in an array
4. Maximum in a range with updates
5. Frequency counting with range queries

Step 7: Mini Exercises

1. Build segment tree for sum queries
2. Update array element and query sum
3. Count inversions using BIT
4. Range maximum query with updates

Step 8: Common Mistakes

- Off-by-one errors in indexing
- Forgetting to propagate updates
- Using segment tree instead of BIT for simple prefix sums
- Not handling 0-based vs 1-based indexing

Step 9: Practice Plan for Segment Trees & BIT

Day 1:

- Build segment tree, sum queries → 5 problems

Day 2:

- Update queries, max/min range → 5 problems

Day 3:

- Binary Indexed Tree, prefix sums → 5 problems

Day 4:

- Inversion count, frequency count → 5–7 problems

Day 5–6:

- Mixed segment tree/BIT problems → 10 problems

Key Takeaways

- Segment Trees → flexible for **range queries and updates**
- BIT → lightweight alternative for **prefix sums and frequency problems**
- Essential for **competitive programming and complex DSA problems**
- Practice builds **index manipulation, recursion, and update/query intuition**

Visit haas.dev for more DSA segment tree and BIT guides, problem sets, and interview preparation resources.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app