Prefix Sum and Segment Tree

Prefix Sum

- You are given an array arr of n integers. You need to efficiently answer multiple queries of the following types:
- Range Sum Query: Given two indices l and r, find the sum of elements in the range [l, r] (inclusive).

Prefix Sum

- Declare a new array prefixSum[] of the same size as the input array
- Run a for loop to traverse the input array
- For each index add the value of the current element and the previous value of the prefix sum array
-

Prefix Sum

```cpp
// Function to compute prefix sums
std::vector<int> prefixSum(const std::vector<int>& arr) {
    int n = arr.size();
    std::vector<int> prefix(n);

    // First element of prefix sum array is the same as the original array
    prefix[0] = arr[0];

    // Compute the prefix sum
    for (int i = 1; i < n; ++i) {
        prefix[i] = prefix[i - 1] + arr[i];
    }

    return prefix;
}
```

Prefix Sum drawbacks

Sum query is o(1) speed but updating takes o(n)

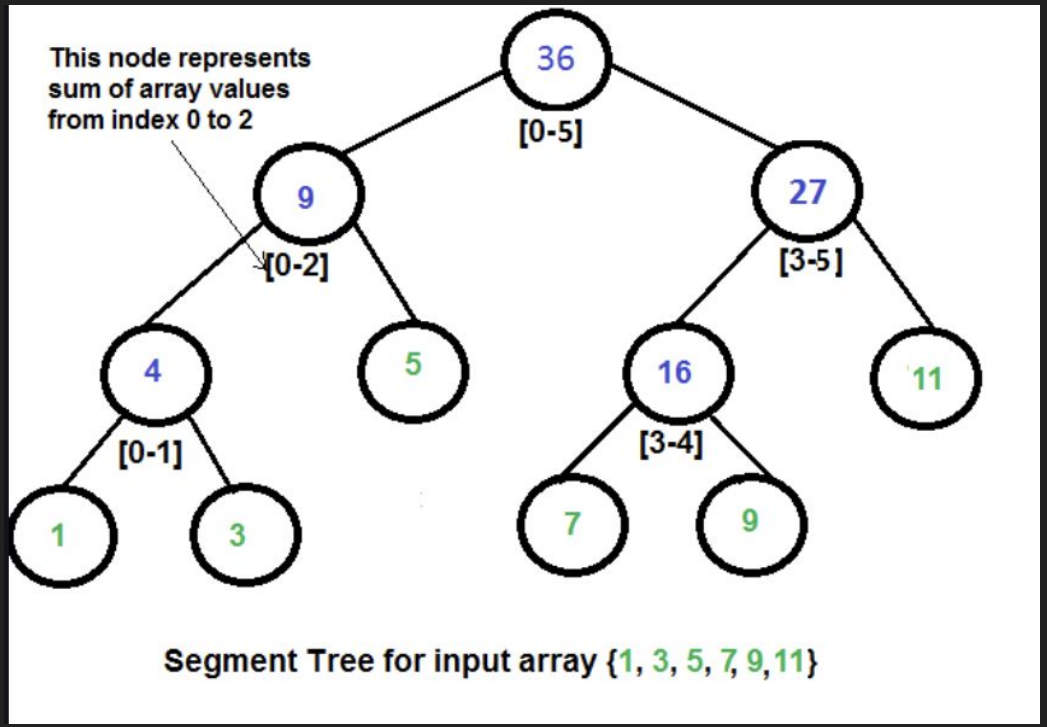
Segment tree

You are given an array arr of n integers. You need to efficiently answer multiple queries of the following types:

Range Sum Query: Given two indices l and r, find the sum of elements in the range [l, r] (inclusive).

Point Update: Given an index i and a value x, update the element at index i to x

Segment tree



Segment tree - build

```
void build(int a[], int v, int tl, int tr) {
    if (tl == tr) {
        t[v] = a[tl];
    } else {
        int tm = (tl + tr) / 2;
        build(a, v*2, tl, tm);
        build(a, v*2+1, tm+1, tr);
        t[v] = t[v*2] + t[v*2+1];
    }
}
```

Segment tree - sum

```
int sum(int v, int tl, int tr, int l, int r) {
    if (l > r)
        return 0;
    if (l == tl && r == tr) {
        return t[v];
    }
    int tm = (tl + tr) / 2;
    return sum(v*2, tl, tm, l, min(r, tm))
           + sum(v*2+1, tm+1, tr, max(l, tm+1), r);
}
```

Segment tree - update

```
void update(int v, int tl, int tr, int pos, int new_val) {
    if (tl == tr) {
        t[v] = new_val;
    } else {
        int tm = (tl + tr) / 2;
        if (pos <= tm)
            update(v*2, tl, tm, pos, new_val);
        else
            update(v*2+1, tm+1, tr, pos, new_val);
        t[v] = t[v*2] + t[v*2+1];
    }
}
```

Segment tree with lazy propagation

- You are given an array arr of length n. You need to perform the following operations efficiently:

  Range Update: Add a given value val to all elements in a range [l, r].

  Range Query: Return the sum of all elements in a range [l, r]

Segment tree with lazy propagation

- Update: When you update a range, instead of directly modifying all nodes, you store the update in the lazy array for the affected segment tree nodes.
- This means the update will only be applied when it's absolutely necessary (i.e., when querying that segment or updating its children).
- Query: When you query a range, before accessing the value of a node, you first check if there are any pending updates in the lazy array. If so, you apply the update to that node and propagate the pending update to its child nodes (if necessary), then continue the query.