Segment Trees

1. Creation of Segment Tree

```
public class CreationOfST {
    static int tree[];
    public static void init(int n){
        tree = new int[4*n];
    }
    public static int buildST(int arr[], int i, int start, int end){
        if (start == end) {
            tree[i] = arr[start];
            return arr[start];
        }
        int mid = (start + end)/2;
        buildST(arr, 2*i+1, start, mid);    //left subtree
        buildST(arr, 2*i+2, mid+1, end);    //Right subtree
        tree[i] = tree[2*i+1] + tree[2*i+2];
        return tree[i];
    }
    public static void main(String[] args) {
        int arr[] = {1, 2, 3, 4, 5, 6, 7, 8};
        int n = arr.length;
        init(n);
        buildST(arr, 0, 0, n-1);
        for(int i=0; i<tree.length; i++){
            System.out.print(tree[i] + " ");
        }
    }
}
```

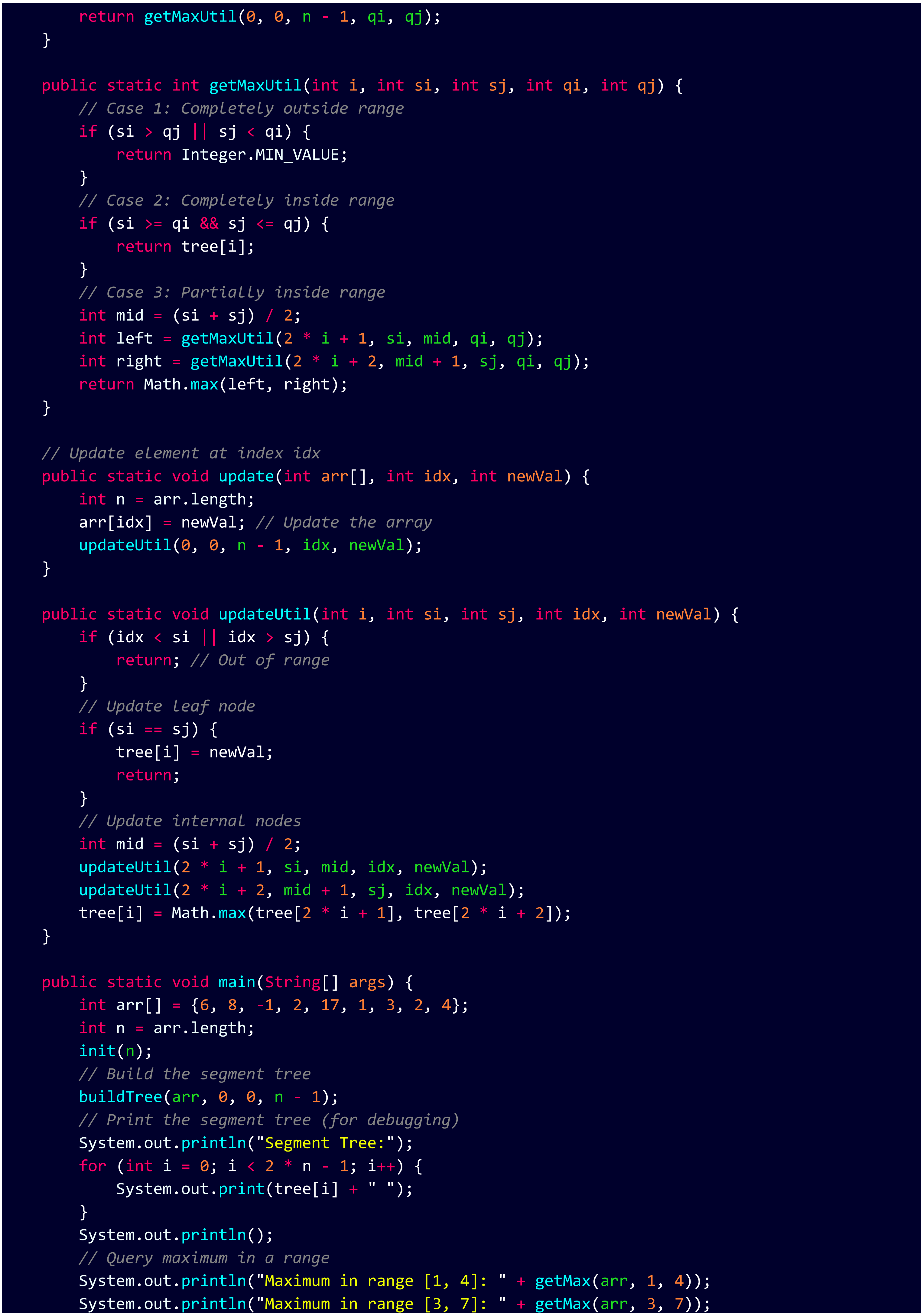
2. Max Element Queries

```
public class MaxElementQueries {
    static int tree[];

    // Initialize the segment tree
    public static void init(int n) {
        tree = new int[4 * n];
    }

    // Build the segment tree
    public static void buildTree(int arr[], int i, int start, int end) {
        if (start == end) {
            tree[i] = arr[start];
            return;
        }
        int mid = (start + end) / 2;
        buildTree(arr, 2 * i + 1, start, mid); // Left subtree
        buildTree(arr, 2 * i + 2, mid + 1, end); // Right subtree
        tree[i] = Math.max(tree[2 * i + 1], tree[2 * i + 2]);
    }

    // Output maximum for the subarray [qi...qj]
    public static int getMax(int arr[], int qi, int qj) {
        int n = arr.length;
```



```java
        return getMaxUtil(0, 0, n - 1, qi, qj);
    }

    public static int getMaxUtil(int i, int si, int sj, int qi, int qj) {
        // Case 1: Completely outside range
        if (si > qj || sj < qi) {
            return Integer.MIN_VALUE;
        }
        // Case 2: Completely inside range
        if (si >= qi && sj <= qj) {
            return tree[i];
        }
        // Case 3: Partially inside range
        int mid = (si + sj) / 2;
        int left = getMaxUtil(2 * i + 1, si, mid, qi, qj);
        int right = getMaxUtil(2 * i + 2, mid + 1, sj, qi, qj);
        return Math.max(left, right);
    }

    // Update element at index idx
    public static void update(int arr[], int idx, int newVal) {
        int n = arr.length;
        arr[idx] = newVal; // Update the array
        updateUtil(0, 0, n - 1, idx, newVal);
    }

    public static void updateUtil(int i, int si, int sj, int idx, int newVal) {
        if (idx < si || idx > sj) {
            return; // Out of range
        }
        // Update leaf node
        if (si == sj) {
            tree[i] = newVal;
            return;
        }
        // Update internal nodes
        int mid = (si + sj) / 2;
        updateUtil(2 * i + 1, si, mid, idx, newVal);
        updateUtil(2 * i + 2, mid + 1, sj, idx, newVal);
        tree[i] = Math.max(tree[2 * i + 1], tree[2 * i + 2]);
    }

    public static void main(String[] args) {
        int arr[] = {6, 8, -1, 2, 17, 1, 3, 2, 4};
        int n = arr.length;
        init(n);
        // Build the segment tree
        buildTree(arr, 0, 0, n - 1);
        // Print the segment tree (for debugging)
        System.out.println("Segment Tree:");
        for (int i = 0; i < 2 * n - 1; i++) {
            System.out.print(tree[i] + " ");
        }
        System.out.println();
        // Query maximum in a range
        System.out.println("Maximum in range [1, 4]: " + getMax(arr, 1, 4));
        System.out.println("Maximum in range [3, 7]: " + getMax(arr, 3, 7));
        // Update an element
        update(arr, 2, 10); // Update index 2 to 10
        System.out.println("After update:");
        for (int i = 0; i < 2 * n - 1; i++) {
            System.out.print(tree[i] + " ");
        }
        System.out.println();
        // Query again after update
        System.out.println("Maximum in range [1, 4]: " + getMax(arr, 1, 4));
        System.out.println("Maximum in range [3, 7]: " + getMax(arr, 3, 7));
    }
}
```

3. Query on Segment Tree

```java
public class QueryOnST {
    static int tree[];
    public static void init(int n){
        tree = new int[4*n];
    }
    public static int buildST(int arr[], int i, int start, int end){
        if (start == end) {
            tree[i] = arr[start];
            return arr[start];
        }
        int mid = (start + end)/2;
        buildST(arr, 2*i+1, start, mid);    //left subtree
        buildST(arr, 2*i+2, mid+1, end);    //Right subtree
        tree[i] = tree[2*i+1] + tree[2*i+2];
        return tree[i];
    }
    public static int getSumUtil(int i, int si, int sj, int qi, int qj){
        if (qj <= si || qi >= sj) {     //Non-Overlapping
            return 0;
        } else if(si >= qi && sj <= qj) {   //Complete Overlap
            return tree[i];
        } else {    //partial overlapping
            int mid = (si + sj) / 2;
            int left = getSumUtil(2*i+1, si, mid, qi, qj);
            int right = getSumUtil(2*i+2, mid+1, sj, qi, qj);
            return left + right;
        }
    }
    public static int getSum(int arr[], int qi, int qj){
        int n = arr.length;
        return getSumUtil(0, 0, n-1, qi, qj);
    }
    public static void main(String[] args) {
        int arr[] = {1, 2, 3, 4, 5, 6, 7, 8};
        int n = arr.length;
        init(n);
        buildST(arr, 0, 0, n-1);
        System.out.println(getSum(arr, 2, 5));  //18
    }
}
```

4. Update on Segment Tree

```java
public class UpdateOnST {
    static int tree[];

    // Initialize the segment tree
    public static void init(int n) {
        tree = new int[4 * n];
    }

    // Build the segment tree
    public static int buildST(int arr[], int i, int start, int end) {
        if (start == end) {
            tree[i] = arr[start];
            return arr[start];
        }
        int mid = (start + end) / 2;
        int left = buildST(arr, 2 * i + 1, start, mid);    // Left subtree
        int right = buildST(arr, 2 * i + 2, mid + 1, end); // Right subtree
        tree[i] = left + right; // Combine results
        return tree[i];
    }

    // Utility function to update the tree
    public static void updateUtil(int i, int si, int sj, int idx, int diff) {
        if (idx > sj || idx < si) {
            return; // Out of range
        }
        tree[i] += diff; // Update current node
        if (si != sj) {
            int mid = (si + sj) / 2;
            updateUtil(2 * i + 1, si, mid, idx, diff);     // Update left subtree
            updateUtil(2 * i + 2, mid + 1, sj, idx, diff); // Update right subtree
        }
    }

    // Public function to handle updates
    public static void update(int arr[], int idx, int newVal) {
        int n = arr.length;
        int diff = newVal - arr[idx];
        arr[idx] = newVal; // Update the array
        updateUtil(0, 0, n - 1, idx, diff); // Update the segment tree
    }

    public static void main(String[] args) {
        int arr[] = {1, 2, 3, 4, 5, 6, 7, 8};
        int n = arr.length;
        init(n);
        buildST(arr, 0, 0, n - 1);

        // Print the tree (only meaningful elements)
        System.out.println("Segment Tree before update:");
        for (int i = 0; i < 2 * n - 1; i++) {
            System.out.print(tree[i] + " ");
        }

        // Perform an update
        update(arr, 4, 10);
        System.out.println("\nSegment Tree after update:");
        for (int i = 0; i < 2 * n - 1; i++) {
            System.out.print(tree[i] + " ");
        }
    }
}
```