Range Sum Query - Mutable

307. Range Sum Query - Mutable

Medium 1646 98 Add to List Share

Given an array `nums` and two types of queries where you should update the value of an index in the array, and retrieve the sum of a range in the array.

Implement the `NumArray` class:

- `NumArray(int[] nums)` Initializes the object with the integer array `nums`.
- `void update(int index, int val)` Updates the value of `nums[index]` to be `val`.
- `int sumRange(int left, int right)` Returns the sum of the subarray `nums[left, right]` (i.e., `nums[left] + nums[left + 1], ..., nums[right]`).

Example 1:

```
Input
["NumArray", "sumRange", "update", "sumRange"]
[[[1, 3, 5]], [0, 2], [1, 2], [0, 2]]
Output
[null, 9, null, 8]

Explanation
NumArray numArray = new NumArray([1, 3, 5]);
numArray.sumRange(0, 2); // return 9 = sum([1,3,5])
numArray.update(1, 2);   // nums = [1,2,5]
numArray.sumRange(0, 2); // return 8 = sum([1,2,5])
```

임의의 배열이 주어지고, 그 배열 내에서 arr[a] ~ arr[b] 까지의 합을 구하세요.

1. simple Ver

```
int sum = 0;
for(int i = a; i <= b; i++) {
    sum += arr[i];
}

static class NumArray {
    private int[] nums;

    public NumArray(int[] nums) {
      this.nums = nums;
    }

    public void update(int i, int val) {
      nums[i] = val;
```

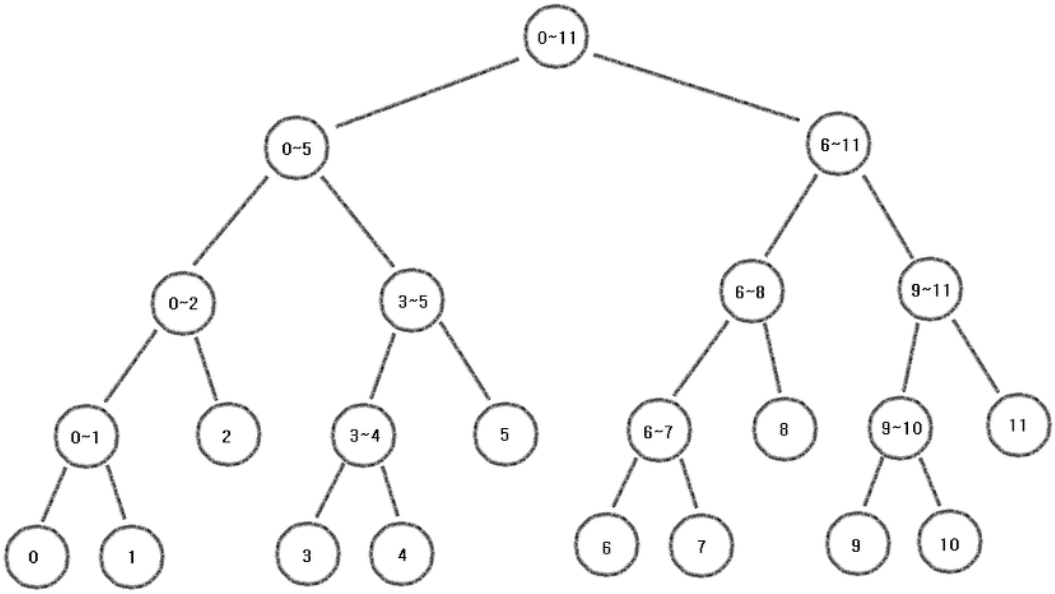
```
    }

    public int sumRange(int i, int j) {

      int sum = 0;

      for(int k = i; k <= j; k++) {
        sum += nums[k];
      }
      return sum;
    }
  }
```

2. 미리 합을 구하자

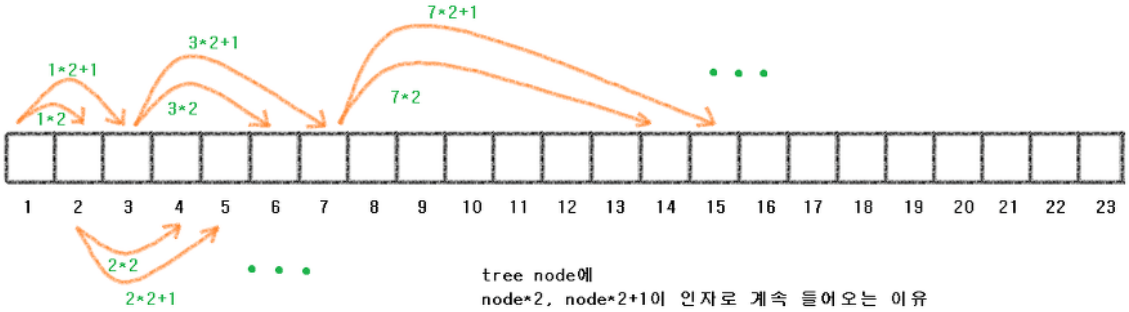
```
class NumArray {
    private int[] nums;
    private int[] sum;

    public NumArray(int[] nums) {

      this.nums = nums;

      sum = new int[(nums.length)];

      for(int i = 0; i < nums.length; i++) {

          if(i == 0) {
              sum[i] = nums[i];
              continue;
          }
          sum[i] = sum[i - 1] + nums[i];
      }
    }

    public void update(int i, int val) {
      int diff = val - nums[i];

      nums[i] = val;
      for(int k = i; k < nums.length; k++) {
        sum[k] += diff;
      }
    }

    public int sumRange(int i, int j) {
        if(i == 0) {
            return sum[j];
        }
        else {
            return sum[j] - sum[i - 1];
        }
    }
```

```
  }

// 이게 오히려 시간초과
// 업데이트 과정에서 N번 순회하는 탓
// O(N) 의 효율
```

세그먼트 트리로 구하는 구간 합

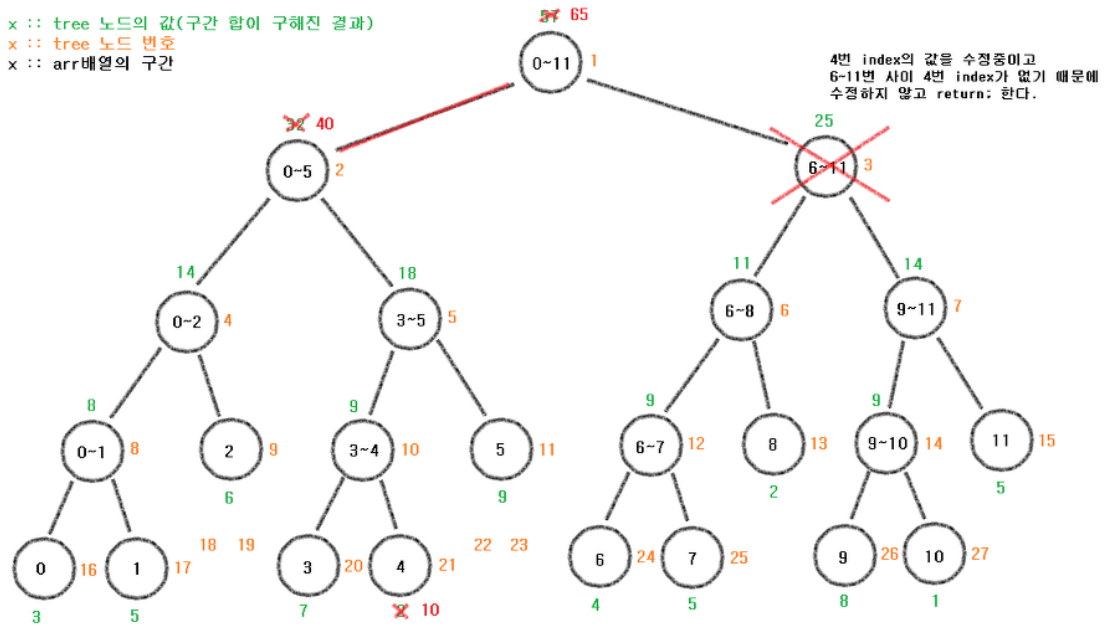


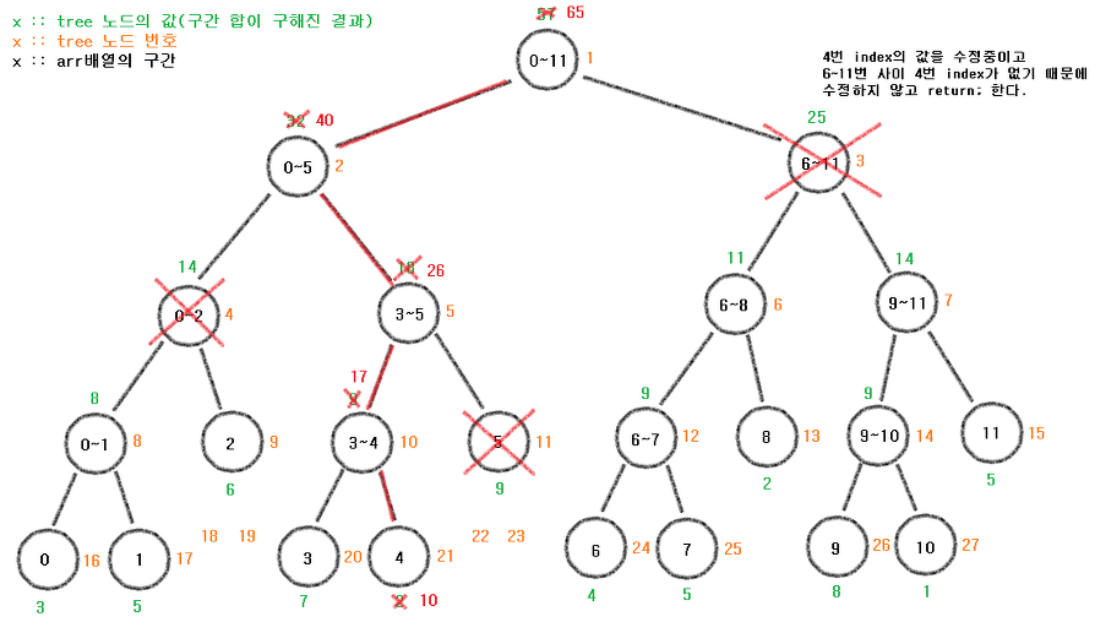

```
배열 형태로 부모-자식 관계를 표현 할때의 일반화

노드의 left child = 부모 node 값의 * 2
노드의 right child = 부모 node 값의 * 2 + 1
```



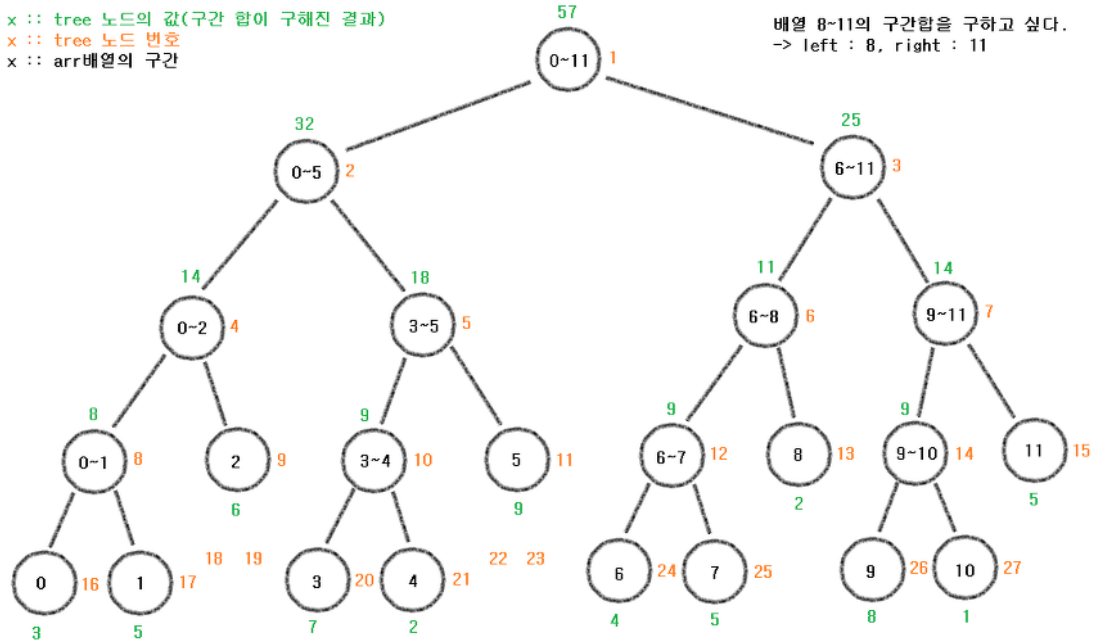
Update 과정

```
target num = 4 라고 하였을 때,

1. 루트에서부터 갖고 있는 start 와 end 사이에 타겟이 포함되어 있는지 확인한다.

2. 포함되어 있다면, child로 내려가 포함되는지 확인하며 리프노드 까지 진행한다.

3. 포함되어 있지 않다면, 갱신 값에 의해 변하지 않으므로 가지치기 한다.
```

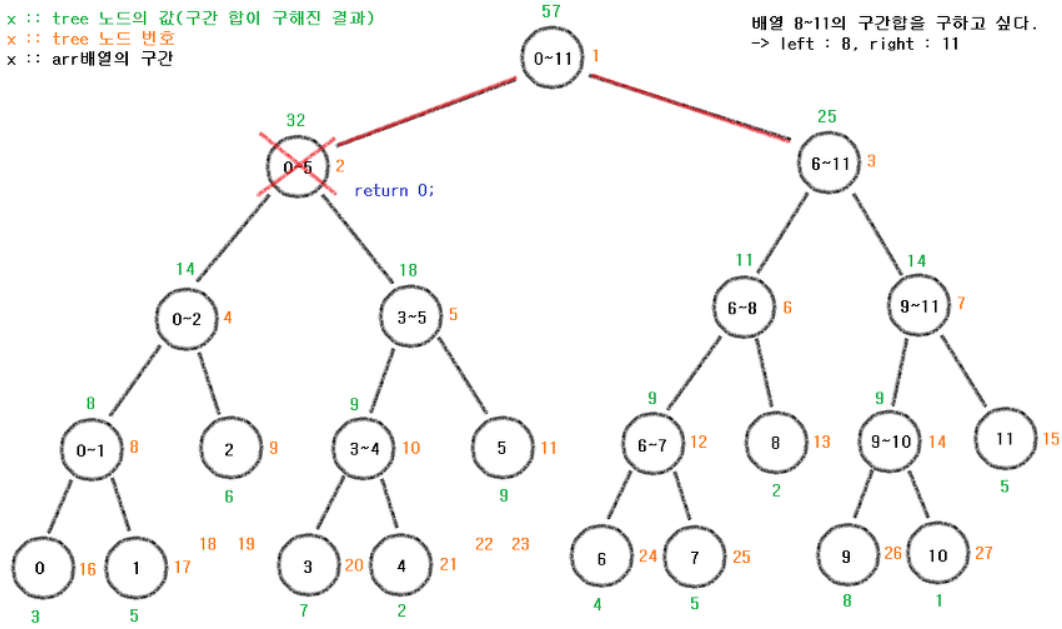



x :: tree 노드의 값(구간 합이 구해진 결과)
x :: tree 노드 번호
x :: arr배열의 구간
4번 index의 값을 수정중이고
6~11번 사이 4번 index가 없기 때문에
수정하지 않고 return; 한다.

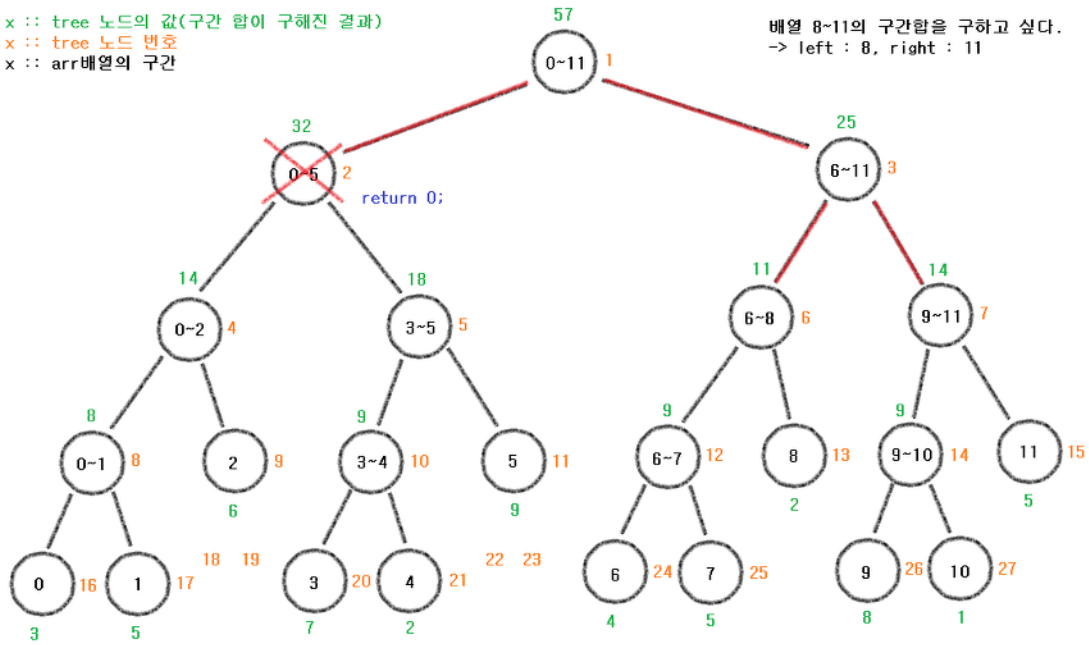
합 구간


x :: tree 노드의 값(구간 합이 구해진 결과)
x :: tree 노드 번호
x :: arr배열의 구간
배열 8~11의 구간합을 구하고 싶다.
-> left : 8, right : 11

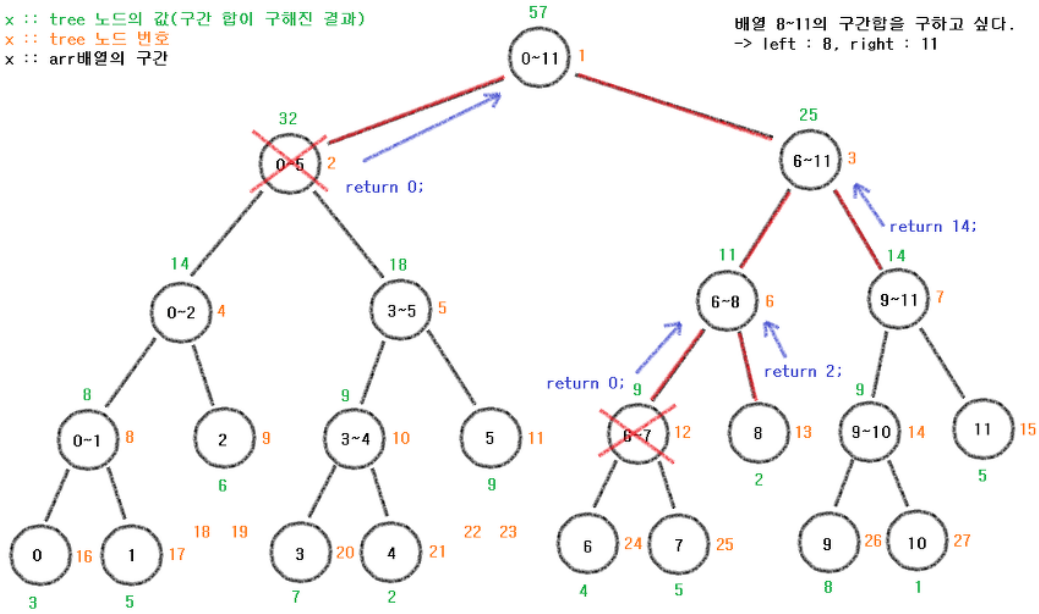
범위가 포함되지 않으므로 0 리턴



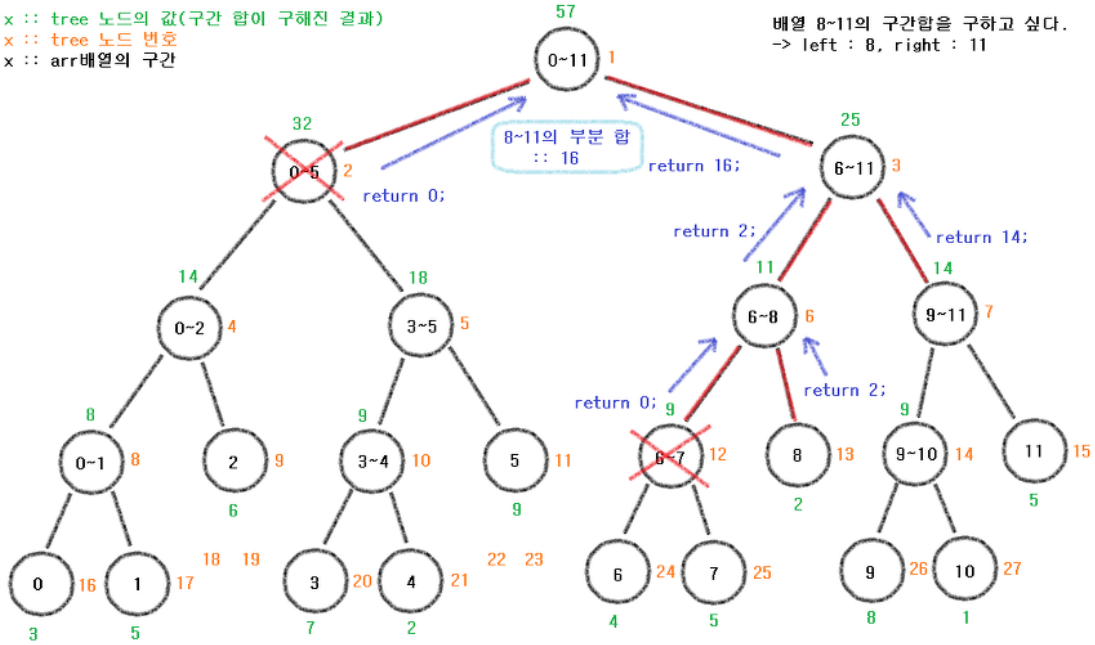
포함은 되지만 넓은 범위가 포함되어 값을 구할 수 없다면 자식 노드에 물어본다.



구간에 포함되면 자신의 값을 다시 리턴해주고



재귀로 다시 돌려받은 리턴 값의 합이 최종 합이 된다.


x :: tree 노드의 값(구간 합이 구해진 결과)
x :: tree 노드 번호
x :: arr배열의 구간
배열 8~11의 구간합을 구하고 싶다.
-> left : 8, right : 11
8~11의 부분 합
:: 16
return 16;
return 0;
return 2;
return 14;
return 0;
return 2;
0~11
0~5
6~11
0~2
3~5
6~8
9~11
0~1
2
3~4
5
6~7
8
9~10
11
0
1
3
4
6
7
9
10

```
// 세용님 코드

static class NumArray {
        int[] origin;
        int[] sumArr;
        Map<Integer, Integer> diffRecord;

        public NumArray(int[] nums) {
            origin = nums.clone();
            sumArr = new int[nums.length];
            for (int i = 0; i < nums.length; i++) {
                if (i == 0) {
                    sumArr[i] = nums[i];
                    continue;
                }
                sumArr[i] = sumArr[i - 1] + nums[i];
            }

            diffRecord = new HashMap<>();
        }

        public void update(int idx, int val) {
            int diff = val - origin[idx];

            if (diffRecord.containsKey(idx)) {
                int newDiff = diffRecord.get(idx) + diff;

                if (newDiff == 0)
                    diffRecord.remove(idx);
                else
                    diffRecord.put(idx, newDiff);

            } else {
                diffRecord.put(idx, diff);
            }
            origin[idx] = val;
```

```
        }

        public int sumRange(int i, int j) {
            int sum;
            if (i == 0)
                sum = sumArr[j];
            else
                sum = sumArr[j] - sumArr[i - 1];

            int diffSum = 0;
            for (int x : diffRecord.keySet()) {
                if (i <= x && x <= j) {
                    diffSum += diffRecord.get(x);
                }
            }
            return sum + diffSum;
        }
    }
```

```
// 홈페이지 솔루션
static class NumArray {
    private int[] nums;
    private int[] sum;

    public NumArray(int[] nums) {

      this.nums = nums;

      sum = new int[(nums.length * 4)];

      for (int i = nums.length, j = 0; i < 2 * nums.length; i++, j++)
        sum[i] = nums[j];
      for (int i = nums.length - 1; i > 0; --i)
        sum[i] = sum[i * 2] + sum[i * 2 + 1];
    }

    public void update(int i, int val) {
      i += nums.length;
      sum[i] = val;
      while (i > 0) {
        int left = i;
        int right = i;
        if (i % 2 == 0) {
          right = i + 1;
        } else {
          left = i - 1;
        }
        // parent is updated after child is updated
        sum[i / 2] = sum[left] + sum[right];
        i /= 2;
      }
    }
```

```
    public int sumRange(int i, int j) {
      i += nums.length;
      j += nums.length;

      int result = 0;
      while (i <= j) {
        if ((i % 2) == 1) {
          result += sum[i];
          i++;
        }
        if ((j % 2) == 0) {
          result += sum[j];
          j--;
        }
        i /= 2;
        j /= 2;
      }
      return result;
    }
  }
```