2555. Maximize Win From Two Segments

My Submissions (/contest/biweekly-contest-97/problems/maximize-win-from-two-segments/submissions/) Back to Contest (/contest/biweekly-contest-97/)

User Accepted:	1440
User Tried:	4552
Total Accepted:	1506
Total Submissions:	10321
Difficulty:	Medium

There are some prizes on the **X-axis**. You are given an integer array `prizePositions` that is **sorted in non-decreasing order**, where `prizePositions[i]` is the position of the i^{th} prize. There could be different prizes at the same position on the line. You are also given an integer `k`.

You are allowed to select two segments with integer endpoints. The length of each segment must be `k`. You will collect all prizes whose position falls within at least one of the two selected segments (including the endpoints of the segments). The two selected segments may intersect.

- For example if `k = 2`, you can choose segments `[1, 3]` and `[2, 4]`, and you will win any prize i that satisfies `1 <= prizePositions[i] <= 3` or `2 <= prizePositions[i] <= 4`.

Return *the* ***maximum*** *number of prizes you can win if you choose the two segments optimally*.

Example 1:

```
Input: prizePositions = [1,1,2,2,3,3,5], k = 2
Output: 7
Explanation: In this example, you can win all 7 prizes by selecting two segments [1, 3] and [3, 5].
```

Example 2:

```
Input: prizePositions = [1,2,3,4], k = 0
Output: 2
Explanation: For this example, one choice for the segments is [3, 3] and [4, 4], and you will be able to get 2 prizes.
```

Constraints:

- `1 <= prizePositions.length <=` 10^5
- `1 <= prizePositions[i] <=` 10^9
- `0 <= k <=` 10^9
- `prizePositions` is sorted in non-decreasing order.

Discuss (https://leetcode.com/problems/maximize-win-from-two-segments/discuss)

JavaScript

```
function Bisect() {
    return { insort_right, insort_left, bisect_left, bisect_right }
    function insort_right(a, x, lo = 0, hi = null) {
        lo = bisect_right(a, x, lo, hi);
        a.splice(lo, 0, x);
    }
    function bisect_right(a, x, lo = 0, hi = null) { // > upper_bound
        if (lo < 0) throw new Error('lo must be non-negative');
        if (hi == null) hi = a.length;
        while (lo < hi) {
            let mid = parseInt((lo + hi) / 2);
            a[mid] > x ? hi = mid : lo = mid + 1;
        }
        return lo;
    }
    function insort_left(a, x, lo = 0, hi = null) {
        lo = bisect_left(a, x, lo, hi);
        a.splice(lo, 0, x);
    }
    function bisect_left(a, x, lo = 0, hi = null) { // >= lower_bound
        if (lo < 0) throw new Error('lo must be non-negative');
        if (hi == null) hi = a.length;
        while (lo < hi) {
            let mid = parseInt((lo + hi) / 2);
            a[mid] < x ? lo = mid + 1 : hi = mid;
        }
        return lo;
    }
}

function TreeMap(g) {
    let ts = [], m = new Map(), bisect = new Bisect();
    initialize();
    return { set, get, firstKey, lastKey, keys, pollFirstEntry, pollLastEntry, ceilingKey, higherKey, lowerKey,
floorKey, ceilingEntry, higherEntry, lowerEntry, floorEntry, remove, has, size, findKth, clear, show };
    function initialize() {
        if (g) {
            for (const [k, v] of g) {
                if (!m.has(k)) bisect.insort_right(ts, k);
                m.set(k, v);
            }
        }
    }
    function set(k, v) {
        if (!m.has(k)) bisect.insort_right(ts, k); // ts has no duplicates/unique key
        m.set(k, v); // update key with most recent value
    }
    function get(k) {
        return m.get(k);
    }
    function keys() {
        return ts;
    }
    function firstKey() {
        return ts[0];
    }
    function lastKey() {
        return ts[ts.length - 1];
    }
    function pollFirstEntry() {
        let k = ts[0], v = m.get(k);
        ts.splice(0, 1);
        m.delete(k);
        return [k, v];
    }
    function pollLastEntry() {
        let k = ts.pop(), v = m.get(k);
        m.delete(k);
        return [k, v];
    }
    function ceilingKey(e) { // >= lower_bound
        let idx = bisect.bisect_right(ts, e);
        let res = ts[idx - 1] == e ? e : ts[bisect.bisect_right(ts, e)];
        return res == undefined ? null : res;
    }
    function higherKey(e) { // > upper_bound
        let idx = bisect.bisect_right(ts, e);
        let res = ts[idx] > e ? ts[idx] : ts[bisect.bisect_right(ts, e) + 1];
        return res == undefined ? null : res;
    }
    function floorKey(e) { // <=
        let idx = bisect.bisect_left(ts, e);
        let res = ts[idx] == e ? e : ts[bisect.bisect_left(ts, e) - 1];
        return res == undefined ? null : res;
    }
    function lowerKey(e) { // <
        let idx = bisect.bisect_left(ts, e);
        let res = ts[idx] < e ? ts[idx] : ts[bisect.bisect_left(ts, e) - 1];
        return res == undefined ? null : res;
    }
    function data(k) {
        return k == null ? null : { key: k, value: m.get(k) }
    }
    function ceilingEntry(k) {
        return data(ceilingKey(k));
    }
    function higherEntry(k) {
        return data(higherKey(k));
    }
    function floorEntry(k) {
        return data(floorKey(k));
    }
    function lowerEntry(k) {
        return data(lowerKey(k));
    }
    function remove(e) {
        let idx = bisect.bisect_left(ts, e);
        if (ts[idx] == e) ts.splice(idx, 1);
        m.delete(e);
    }
    function has(e) {
        return m.has(e);
    }
    function size() {
        return ts.length;
    }
    function findKth(k) {
        let cnt = 0;
        for (const x of ts) {
            let occ = m.get(x);
            if (cnt + occ < k) {
                cnt += occ;
            } else {
                return x;
            }
        }
    }
    function clear() {
        ts = [];
        m.clear();
    }
    function show() {
        let res = new Map();
        for (const x of ts) res.set(x, m.get(x));
        return res;
    }
}

const maximizeWin = (a, k) => {
    let m = new TreeMap(), res = 0, pre = 0;
    for (let i = 0; i < a.length; i++) m.set(a[i], i + 1);
    for (const x of m.keys()) {
        let l = x - k, r = x + k, xLastIdx = m.get(x) || 0;
        let floorR = m.floorKey(r);
        let floorRLastIdx = floorR == null ? 0 : m.get(floorR);
        let lowerX = m.lowerKey(x);
        let lowerXLastIdx = lowerX == null ? 0 : m.get(lowerX);
        let lowerL = m.lowerKey(l);
        let lowerLLastIdx = lowerL == null ? 0 : m.get(lowerL);
        res = Math.max(res, pre + floorRLastIdx - lowerXLastIdx);
        pre = Math.max(pre, xLastIdx - lowerLLastIdx);
    }
    return res;
};
```

Custom Testcase Use Example Testcases

Run Submit

Submission Result: Accepted (/submissions/detail/891620267/) More Details (/submissions/detail/891620267/)

Share your acceptance!