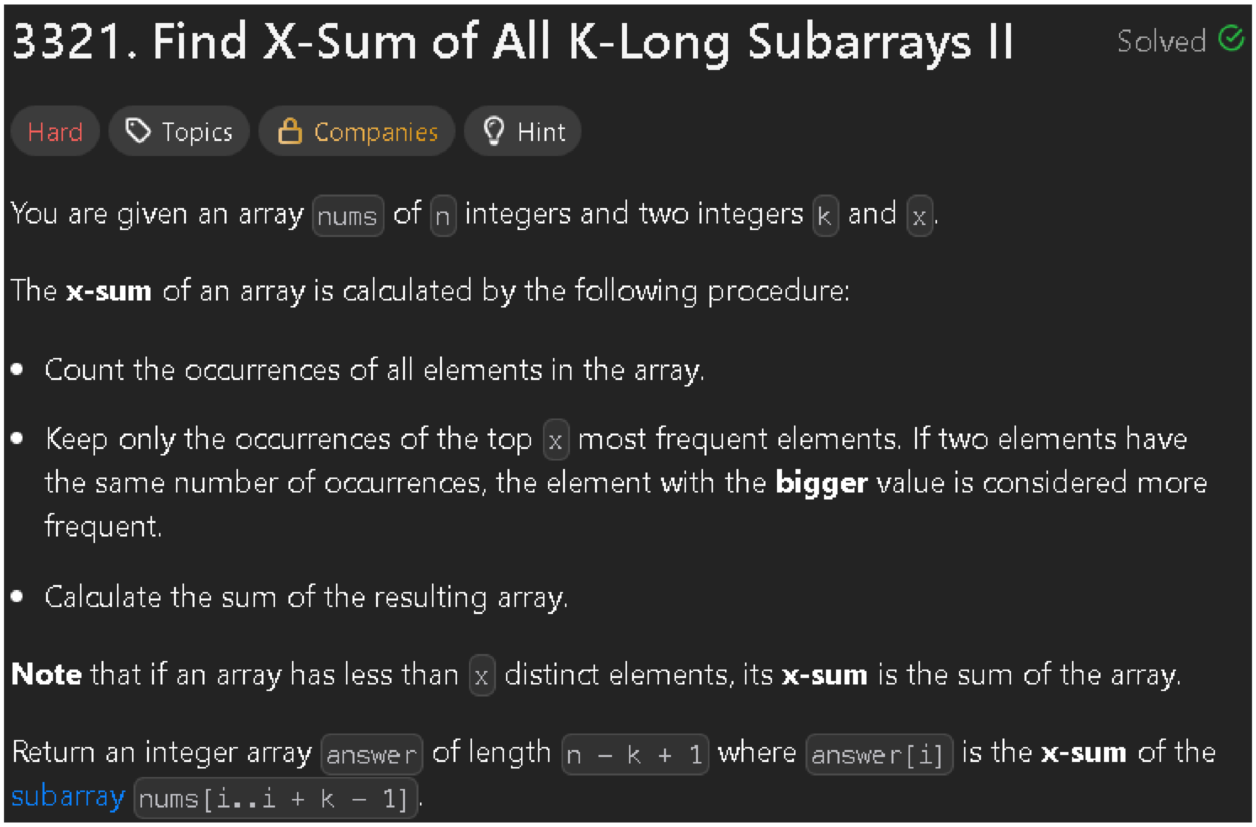


3321. Find X-Sum of All K-Long Subarrays II

Solved ✓

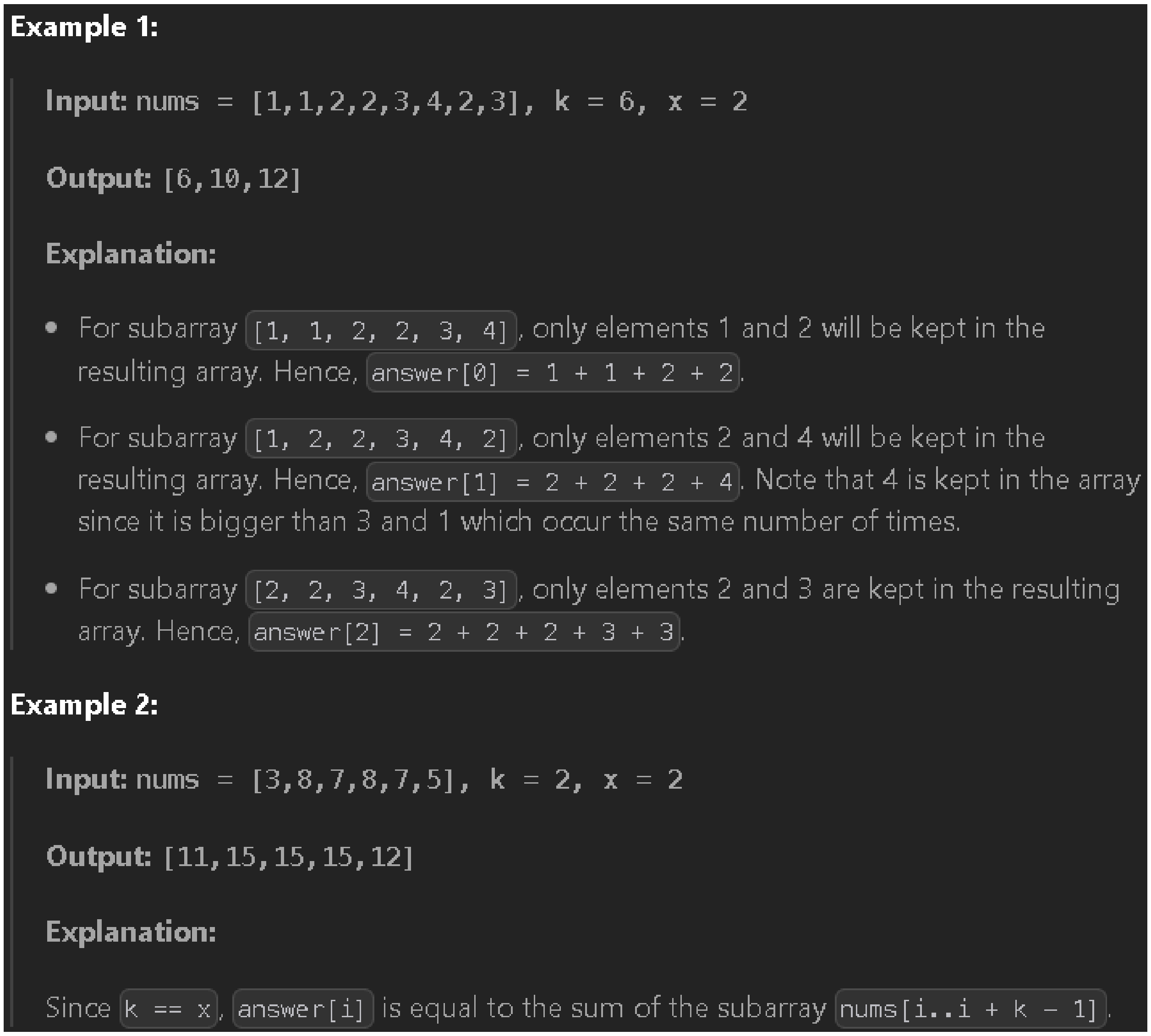
You are given an array `nums` of `n` integers and two integers `k` and `x`.

The **x-sum** of an array is calculated by the following procedure:

- Count the occurrences of all elements in the array.
- Keep only the occurrences of the top `x` most frequent elements. If two elements have the same number of occurrences, the element with the **bigger** value is considered more frequent.
- Calculate the sum of the resulting array.

Note that if an array has less than `x` distinct elements, its **x-sum** is the sum of the array.

Return an integer array `answer` of length `n - k + 1` where `answer[i]` is the **x-sum** of the subarray `nums[i..i + k - 1]`.

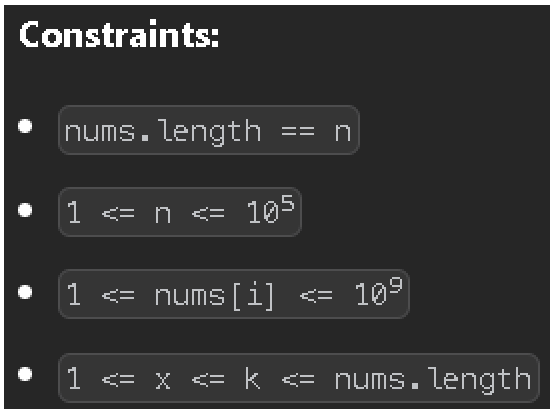


Example 1:

> **Input:** nums = [1,1,2,2,3,4,2,3], k = 6, x = 2
>
> **Output:** [6,10,12]
>
> **Explanation:**
>
> - For subarray `[1, 1, 2, 2, 3, 4]`, only elements 1 and 2 will be kept in the resulting array. Hence, `answer[0] = 1 + 1 + 2 + 2`.
> - For subarray `[1, 2, 2, 3, 4, 2]`, only elements 2 and 4 will be kept in the resulting array. Hence, `answer[1] = 2 + 2 + 2 + 4`. Note that 4 is kept in the array since it is bigger than 3 and 1 which occur the same number of times.
> - For subarray `[2, 2, 3, 4, 2, 3]`, only elements 2 and 3 are kept in the resulting array. Hence, `answer[2] = 2 + 2 + 2 + 3 + 3`.

Example 2:

> **Input:** nums = [3,8,7,8,7,5], k = 2, x = 2
>
> **Output:** [11,15,15,15,12]
>
> **Explanation:**
>
> Since `k == x`, `answer[i]` is equal to the sum of the subarray `nums[i..i + k - 1]`.



Constraints:

- `nums.length == n`
- $1 <= n <= 10^5$
- $1 <= nums[i] <= 10^9$
- `1 <= x <= k <= nums.length`

Python:

from sortedcontainers import SortedList

```
class Solution:
    def findXSum(self, A: List[int], K: int, X: int) -> List[int]:
        bot = SortedList()
        top = SortedList()
        count = Counter()
        cur_sum = 0

        def update(x, qty):
            nonlocal cur_sum

            if count[x]:
                try:
                    bot.remove([count[x], x])
                except:
                    top.remove([count[x], x])
                    cur_sum -= count[x] * x

            count[x] += qty
            if count[x]:
                bot.add([count[x], x])

        ans = []
        for i in range(len(A)):
            update(A[i], 1)
            if i >= K:
                update(A[i - K], -1)

            # rebalance
            while bot and len(top) < X:
                cx, x = bot.pop()
                cur_sum += cx * x
                top.add([cx, x])

            while bot and bot[-1] > top[0]:
                cx, x = bot.pop()
                cy, y = top.pop(0)
                cur_sum += cx * x - cy * y
                bot.add([cy, y])
                top.add([cx, x])

            if i >= K - 1:
                ans.append(cur_sum)

        return ans
```

JavaScript:

```
var findXSum = function (nums, k, x) {
        const n = nums.length;
        let res = [],
                sum = 0;
        let freqMap = new Map(),
                top = new OrderedSet(),
                rest = new OrderedSet();

        for (let i = 0; i < n; i++) {
                let count = freqMap.get(nums[i]) || 0;
                if (count > 0) {
                        if (rest.find(nums[i], count)) {
                                rest.delete(nums[i], count);
                        } else {
                                top.delete(nums[i], count);
                                sum -= nums[i] * count;
                        }
                }

                freqMap.set(nums[i], count + 1);
                top.insert(nums[i], count + 1);
                sum += nums[i] * (count + 1);

                if (top.size > x) {
                        const [minNum, minCount] = top.getMin();
                        sum -= minNum * minCount;
                        rest.insert(minNum, minCount);
                        top.delete(minNum, minCount);
                }

                if (i >= k) {
                        const leftCount = freqMap.get(nums[i - k]);
                        if (rest.find(nums[i - k], leftCount)) {
                                rest.delete(nums[i - k], leftCount);
                        } else {
                                top.delete(nums[i - k], leftCount);
                                sum -= leftCount * nums[i - k];
                        }

                        freqMap.set(nums[i - k], leftCount - 1);
                        if (leftCount - 1 > 0) {
                                rest.insert(nums[i - k], leftCount - 1);
                }
                if (top.size < x && rest.size > 0) {
                    const [maxNum, maxCount] = rest.getMax();
                    sum += maxNum * maxCount;
                    top.insert(maxNum, maxCount);
                    rest.delete(maxNum, maxCount);
                }
            }
            if (i >= k - 1) {
                res.push(sum);
            }
        }
        return res;
};
class RBTreeNode {
        constructor(key, value, nilNode) {
            this.key = key;
            this.value = value;
            this.color = 'red';
            this.left = nilNode;
            this.right = nilNode;
            this.parent = nilNode;
        }

        isRed() {
            return this.color === 'red';
        }
}

class RBTree {
        constructor() {
            this.nil = new RBTreeNode(null, null, null); // nil 节点初始化
            this.nil.color = 'black'; // nil 节点是黑色的
            this.root = this.nil;
        }

        // 自定义的比较函数，先按value比较，value相同再按key比较
        compare(node1, node2) {
            if (node1.value !== node2.value) {
                return node1.value - node2.value; // 按value升序排序
            }
            return node1.key - node2.key; // value相同则按key升序排序
        }
```

```
insert(key, value) {
        let z = new RBTreeNode(key, value, this.nil);
        let y = this.nil;
        let x = this.root;
        // 插入节点时根据compare函数来比较
        while (x !== this.nil) {
                y = x;
                if (this.compare(z, x) < 0) {
                        x = x.left;
                } else {
                        x = x.right;
                }
        }

        z.parent = y;
        if (y === this.nil) {
                this.root = z;
        } else if (this.compare(z, y) < 0) {
                y.left = z;
        } else {
                y.right = z;
        }

        z.left = this.nil;
        z.right = this.nil;
        z.color = 'red';

        this.insertFixup(z);
}

// 修改 delete 方法，基于 key 和 value 查找
delete(key, value) {
        let node = this.root;
        let targetNode = null;
        // 查找符合 key 和 value 的节点
        while (node !== this.nil) {
                let tempNode = new RBTreeNode(key, value, this.nil);
                if (this.compare(tempNode, node) === 0) {
                        targetNode = node; // 找到目标节点
                        break;
                } else if (this.compare(tempNode, node) < 0) {
                        node = node.left;
                } else {
                        node = node.right;
```

```
            }
        }

        if (targetNode) {
            this._deleteNode(targetNode);
        }
    }

    _deleteNode(node) {
        let y = node;
        let yOriginalColor = y.color;
        let x;

        if (node.left === this.nil) {
            x = node.right;
            this.transplant(node, node.right);
        } else if (node.right === this.nil) {
            x = node.left;
            this.transplant(node, node.left);
        } else {
            y = this.minimum(node.right);
            yOriginalColor = y.color;
            x = y.right;
            if (y.parent === node) {
                x.parent = y;
            } else {
                this.transplant(y, y.right);
                y.right = node.right;
                y.right.parent = y;
            }
            this.transplant(node, y);
            y.left = node.left;
            y.left.parent = y;
            y.color = node.color;
        }

        if (yOriginalColor === 'black') {
            this.deleteFixup(x);
        }
    }

    transplant(u, v) {
        if (u.parent === this.nil) {
            this.root = v;
        } else if (u === u.parent.left) {
            u.parent.left = v;
        } else {
            u.parent.right = v;
        }
        v.parent = u.parent;
    }

    minimum(node) {
        while (node.left !== this.nil) {
            node = node.left;
        }
        return node;
    }

    maximum(node) {
        while (node.right !== this.nil) {
            node = node.right;
        }
        return node;
    }

    deleteFixup(x) {
        while (x !== this.root && !x.isRed()) {
            if (x === x.parent.left) {
                let w = x.parent.right;
                if (w.isRed()) {
                    w.color = 'black';
                    x.parent.color = 'red';
                    this.leftRotate(x.parent);
                    w = x.parent.right;
                }
                if (!w.left.isRed() && !w.right.isRed()) {
                    w.color = 'red';
                    x = x.parent;
                } else {
                    if (!w.right.isRed()) {
                        w.left.color = 'black';
                        w.color = 'red';
                        this.rightRotate(w);
                        w = x.parent.right;
                    }
                    w.color = x.parent.color;
                    x.parent.color = 'black';
                    w.right.color = 'black';
                    this.leftRotate(x.parent);
                    x = this.root;
                }
            } else {
                let w = x.parent.left;
                if (w.isRed()) {
                    w.color = 'black';
                    x.parent.color = 'red';
                    this.rightRotate(x.parent);
                    w = x.parent.left;
                }
                if (!w.right.isRed() && !w.left.isRed()) {
                    w.color = 'red';
                    x = x.parent;
                } else {
                    if (!w.left.isRed()) {
                        w.right.color = 'black';
                        w.color = 'red';
                        this.leftRotate(w);
                        w = x.parent.left;
                    }
                    w.color = x.parent.color;
                    x.parent.color = 'black';
                    w.left.color = 'black';
                    this.rightRotate(x.parent);
                    x = this.root;
                }
            }
        }
        x.color = 'black';
    }

    insertFixup(z) {
        while (z.parent.isRed()) {
            if (z.parent === z.parent.parent.left) {
                let y = z.parent.parent.right;
                if (y.isRed()) {
                    z.parent.color = 'black';
                    y.color = 'black';
                    z.parent.parent.color = 'red';
                    z = z.parent.parent;
                } else {
                    if (z === z.parent.right) {
                        z = z.parent;
                        this.leftRotate(z);
                    }
                    z.parent.color = 'black';
                    z.parent.parent.color = 'red';
                    this.rightRotate(z.parent.parent);
                }
            } else {
                let y = z.parent.parent.left;
                if (y.isRed()) {
                    z.parent.color = 'black';
                    y.color = 'black';
                    z.parent.parent.color = 'red';
                    z = z.parent.parent;
                } else {
                    if (z === z.parent.left) {
                        z = z.parent;
                        this.rightRotate(z);
                    }
                    z.parent.color = 'black';
                    z.parent.parent.color = 'red';
                    this.leftRotate(z.parent.parent);
                }
            }
        }
        this.root.color = 'black';
    }

    leftRotate(x) {
        let y = x.right;
        x.right = y.left;
        if (y.left !== this.nil) {
            y.left.parent = x;
        }
        y.parent = x.parent;
        if (x.parent === this.nil) {
            this.root = y;
        } else if (x === x.parent.left) {
            x.parent.left = y;
        } else {
            x.parent.right = y;
        }
        y.left = x;
        x.parent = y;
```

```
	}

	rightRotate(x) {
		let y = x.left;
		x.left = y.right;
		if (y.right !== this.nil) {
			y.right.parent = x;
		}
		y.parent = x.parent;
		if (x.parent === this.nil) {
			this.root = y;
		} else if (x === x.parent.left) {
			x.parent.left = y;
		} else {
			x.parent.right = y;
		}
		y.right = x;
		x.parent = y;
	}
}

class OrderedSet {
	constructor() {
		this.rbtree = new RBTree();
		this.size = 0;
	}

	insert(key, value) {
		this.rbtree.insert(key, value);
		this.size++;
	}

	find(key, value) {
		let node = this.rbtree.root;
		let targetNode = null;
		let tempNode = new RBTreeNode(key, value, this.rbtree.nil);

		while (node !== this.rbtree.nil) {
			if (this.rbtree.compare(tempNode, node) === 0) {
				targetNode = node;
				break;
			} else if (this.rbtree.compare(tempNode, node) < 0) {
				node = node.left;
			} else {
```

```
                    node = node.right;
                }
            }

            return targetNode ? targetNode.value : null;
        }

        delete(key, value) {
            this.rbtree.delete(key, value);
            this.size--;
        }

        getMin() {
            let node = this.rbtree.minimum(this.rbtree.root);
            return [node.key, node.value];
        }

        getMax() {
            let node = this.rbtree.maximum(this.rbtree.root);
            return [node.key, node.value];
        }

        // 中序遍历
        inorderTraversal() {
            const result = [];
            const inorder = (node) => {
                if (node !== this.rbtree.nil) {
                    inorder(node.left); // 递归左子树
                    result.push([node.key, node.value]); // 访问当前节点
                    inorder(node.right); // 递归右子树
                }
            };
            inorder(this.rbtree.root);
            return result;
        }
}
```

Java:

```
class Pair implements Comparable<Pair>{
   int val;
   int freq;
   public Pair(int v,int f){
      this.val = v;
      this.freq = f;
```

```
    }
    public int compareTo(Pair p){
        if(this.freq == p.freq){
            return this.val - p.val;
        }else{

            return this.freq - p.freq;
        }
    }
}

class Solution {
    long sum = 0;
    HashMap<Integer,Integer> map = new HashMap<>();
    TreeSet<Pair> large = new TreeSet<>();
    TreeSet<Pair> small = new TreeSet<>();
    public void update(int x,int v){
        int freq = map.getOrDefault(x,0);
        if(large.contains(new Pair(x,freq))){
            large.remove(new Pair(x,freq));
            sum -= 1l * freq * x;
            map.put(x,freq+v);
            sum += 1l* map.get(x) * x;
            large.add(new Pair(x,map.get(x)));

        }else if(small.contains(new Pair(x,freq))){
            small.remove(new Pair(x,freq));
            map.put(x,freq+v);
            small.add(new Pair(x,map.get(x)));
        }
    }

    public void equilibrium(int x){
        while(large.size()< x && !small.isEmpty()){
            Pair second = small.last();
            large.add(second);
            sum += 1l * second.val * second.freq;
            small.remove(second);
        }

        if(small.isEmpty()){
            return;
        }
        while(true){
            Pair first = large.first();
            Pair second = small.last();

            if(first.freq<second.freq || (first.freq==second.freq && first.val < second.val)){
                large.remove(first);
                small.remove(second);
                large.add(second);
                small.add(first);
                sum -= 1l * first.val * first.freq;
                sum += 1l * second.val * second.freq;
            }else{
                break;
            }
        }
    }

    public long[] findXSum(int[] nums, int k, int x) {
        int n = nums.length;
        long ans[] = new long[n-k+1];
        // initialize set small -> (nums[i],freq)
        for(int i=0; i<n; i++){
            small.add(new Pair(nums[i],0));
        }

        for(int i=0; i<n; i++){
            update(nums[i],1); // insert ith element inside window
            if(i>=k){
                update(nums[i-k],-1); // remove starting element of window
            }
            if(i>=k-1){
                equilibrium(x);  //generate ans
                ans[i-k+1] = sum;
            }
        }
        return ans;
    }
}
```