

2426. Number of Pairs Satisfying Inequality

[My Submissions \(/contest/biweekly-contest-88/problems/number-of-pairs-satisfying-inequality/submissions/\)](#)
[Back to Contest \(/contest/biweekly-contest-88/\)](#)

You are given two **0-indexed** integer arrays `nums1` and `nums2`, each of size `n`, and an integer `diff`. Find the number of **pairs** (i, j) such that:

- $0 \leq i < j \leq n - 1$ **and**
- $nums1[i] - nums1[j] \leq nums2[i] - nums2[j] + diff$.

Return the **number of pairs** that satisfy the conditions.

Example 1:

Input: `nums1 = [3,2,5], nums2 = [2,2,1], diff = 1`

Output: 3

Explanation:

There are 3 pairs that satisfy the conditions:

- $i = 0, j = 1$: $3 - 2 \leq 2 - 2 + 1$. Since $i < j$ and $1 \leq 1$, this pair satisfies the conditions.
 - $i = 0, j = 2$: $3 - 5 \leq 2 - 1 + 1$. Since $i < j$ and $-2 \leq 2$, this pair satisfies the conditions.
 - $i = 1, j = 2$: $2 - 5 \leq 2 - 1 + 1$. Since $i < j$ and $-3 \leq 2$, this pair satisfies the conditions.
- Therefore, we return 3.

Example 2:

Input: `nums1 = [3,-1], nums2 = [-2,2], diff = -1`

Output: 0

Explanation:

Since there does not exist any pair that satisfies the conditions, we return 0.

Constraints:

- $n == nums1.length == nums2.length$
- $2 \leq n \leq 10^5$
- $-10^4 \leq nums1[i], nums2[i] \leq 10^4$
- $-10^4 \leq diff \leq 10^4$

[Discuss \(https://leetcode.com/problems/number-of-pairs-satisfying-inequality/discuss/\)](https://leetcode.com/problems/number-of-pairs-satisfying-inequality/discuss/)

JavaScript



```

1 class Node {
2   constructor(value) {
3     this.parent_ = null;
4     this.left_ = null;
5     this.right_ = null;
6     this.value_ = value;
7     this.size_ = 1;
8   }
9   Update() {
10    this.size_ = (this.left_ != null ? this.left_.size_ : 0) + (this.right_ != null ? this.right_.size_ : 0) + 1;
11  }
12  IsLeft() {
13    return this.parent_ != null && this.parent_.left_ == this;
14  }
15  IsRight() {
16    return this.parent_ != null && this.parent_.right_ == this;
17  }
18  IsRoot(guard = null) {
19    return this.parent_ == guard;
20  }
21 }
22 }
23
24 class SplayTree {

```

```

25 constructor() {
26     this.root_ = null;
27     this.cmp_ = (x, y) => x >= y ? 0 : 1;
28 }
29 Zig(x) {
30     let y = x.parent_;
31     if (x.right_ != null) x.right_.parent_ = y;
32     y.left_ = x.right_;
33     x.right_ = y;
34     if (y.IsLeft()) {
35         y.parent_.left_ = x;
36     } else if (y.IsRight()) { // Special attention here for Link-Cut Trees.
37         y.parent_.right_ = x;
38     }
39     x.parent_ = y.parent_;
40     y.parent_ = x;
41     y.Update();
42     x.Update();
43 }
44
45 // Zag:
46 //   y           x
47 //  / \         / \
48 // A  x  --.  y  C
49 //   / \       / \
50 //  B  C       A  B
51 Zag(x) {
52     let y = x.parent_;
53     if (x.left_ != null) x.left_.parent_ = y;
54     y.right_ = x.left_;
55     x.left_ = y;
56     if (y.IsLeft()) {
57         y.parent_.left_ = x;
58     } else if (y.IsRight()) { // Special attention here for Link-Cut Trees.
59         y.parent_.right_ = x;
60     }
61     x.parent_ = y.parent_;
62     y.parent_ = x;
63     y.Update();
64     x.Update();
65 }
66 ZigZig(x) {
67     this.Zig(x.parent_);
68     this.Zig(x);
69 }
70 ZigZag(x) {
71     this.Zig(x);
72     this.Zag(x);
73 }
74 ZagZag(x) {
75     this.Zag(x.parent_);
76     this.Zag(x);
77 }
78 ZagZig(x) {
79     this.Zag(x);
80     this.Zig(x);
81 }
82
83 // Splay a "node" just under a "guard", which is default to splay to the "root_".
84 Splay(node, guard = null) {
85     while (!node.IsRoot(guard)) {
86         if (node.parent_.IsRoot(guard)) {
87             if (node.IsLeft()) {
88                 this.Zig(node);
89             } else {
90                 this.Zag(node);
91             }
92         } else {
93             if (node.parent_.IsLeft()) {
94                 if (node.IsLeft()) {
95                     this.ZigZig(node);
96                 } else {
97                     this.ZagZig(node);
98                 }
99             } else {
100                 if (node.IsRight()) {

```

```

102         this.ZagZag(node);
103     } else {
104         this.ZigZag(node);
105     }
106     }
107 }
108 }
109 if (guard == null) this.root_ = node; // reset "root_" to "node".
110 }
111 LastNode(x) {
112     this.Splay(x);
113     let node = x.left_;
114     if (node == null) return null;
115     while (node.right_ != null) node = node.right_;
116     this.Splay(node);
117     return node;
118 }
119 NextNode(x) {
120     this.Splay(x);
121     let node = x.right_;
122     if (node == null) return null;
123     while (node.left_ != null) node = node.left_;
124     this.Splay(node);
125     return node;
126 }
127 Find(value) {
128     return this.FindFirstOf(value);
129 }
130 FindFirstOf(value) {
131     let node = this.root_, res = null, last_visited = null;
132     while (node != null) {
133         last_visited = node;
134         if (this.cmp_(value, node.value_)) {
135             node = node.left_;
136         } else if (this.cmp_(node.value_, value)) {
137             node = node.right_;
138         } else {
139             res = node;
140             node = node.left_;
141         }
142     }
143     if (last_visited != null) this.Splay(last_visited);
144     return res;
145 }
146 FindLastOf(value) {
147     let node = this.root_, res = null, last_visited = null;
148     while (node != null) {
149         last_visited = node;
150         if (this.cmp_(value, node.value_)) {
151             node = node.left_;
152         } else if (this.cmp_(node.value_, value)) {
153             node = node.right_;
154         } else {
155             res = node;
156             node = node.right_;
157         }
158     }
159     if (last_visited != null) this.Splay(last_visited);
160     return res;
161 }
162 FindRankOf(node) {
163     this.Splay(node);
164     return node.left_ == null ? 0 : node.left_.size_;
165 }
166 FindSuccessorOf(value) {
167     let node = this.root_, res = null, last_visited = null;
168     while (node != null) {
169         last_visited = node;
170         if (this.cmp_(value, node.value_)) {
171             res = node;
172             node = node.left_;
173         } else {
174             node = node.right_;
175         }
176     }
177     if (last_visited != null) this.Splay(last_visited);
178     return res;

```

```

179     }
180     FindPrecursorOf(value) {
181         let node = this.root_, res = null, last_visited = null;
182         while (node != null) {
183             last_visited = node;
184             if (this.cmp_(node.value_, value)) {
185                 res = node;
186                 node = node.right_;
187             } else {
188                 node = node.left_;
189             }
190         }
191         if (last_visited != null) this.Splay(last_visited);
192         return res;
193     }
194     FindKth(rank) {
195         if (rank < 0 || rank >= this.Size()) return null;
196         let node = this.root_;
197         while (node != null) {
198             let left_size = node.left_ == null ? 0 : node.left_.size_;
199             if (left_size == rank) break;
200             if (left_size > rank) {
201                 node = node.left_;
202             } else {
203                 rank -= left_size + 1;
204                 node = node.right_;
205             }
206         }
207         this.Splay(node);
208         return node;
209     }
210     NewNode(value) {
211         return new Node(value);
212     }
213     DeleteNode(node) {
214         node = null;
215     }
216
217     // ----- Public Usage -----
218     Size() {
219         return this.root_ == null ? 0 : this.root_.size_;
220     }
221     IsEmpty() {
222         return this.root_ == null;
223     }
224
225     // Insert an element into the container O(log(n))
226     Insert(value) {
227         if (this.root_ == null) {
228             this.root_ = this.NewNode(value);
229             return this.root_;
230         }
231         let node = this.root_;
232         while (node != null) {
233             if (this.cmp_(value, node.value_)) {
234                 if (node.left_ == null) {
235                     node.left_ = this.NewNode(value);
236                     node.left_.parent_ = node;
237                     node = node.left_;
238                     break;
239                 }
240                 node = node.left_;
241             } else {
242                 if (node.right_ == null) {
243                     node.right_ = this.NewNode(value);
244                     node.right_.parent_ = node;
245                     node = node.right_;
246                     break;
247                 }
248                 node = node.right_;
249             }
250         }
251         this.Splay(node);
252         return node;
253     }
254
255     // Delete an element from the container if it exists O(log n)

```

```

256 Delete(value) {
257     let node = this.Find(value);
258     if (node == null) return false;
259     this.Splay(node);
260     if (node.left_ == null) {
261         this.root_ = node.right_;
262         if (node.right_ != null) node.right_.parent_ = null;
263         this.DeleteNode(node);
264         return true;
265     }
266     if (node.right_ == null) {
267         this.root_ = node.left_;
268         if (node.left_ != null) node.left_.parent_ = null;
269         this.DeleteNode(node);
270         return true;
271     }
272     let last_node = this.LastNode(node);
273     let next_node = this.NextNode(node);
274     this.Splay(last_node);
275     this.Splay(next_node, last_node);
276     // After the above operations, the tree becomes:
277     //      last_node
278     //      /      \
279     //   A        next_node
280     //      /      \
281     //   node      B
282     // Then "next_node.left_" is "node".
283     this.DeleteNode(next_node.left_);
284     next_node.left_ = null;
285     next_node.Update();
286     last_node.Update();
287     return true;
288 }
289
290 // Whether the splay tree contains value O(log n).
291 Contains(value) {
292     return this.CountOf(value) > 0;
293 }
294
295 // The number of occurrences of value O(log n)
296 CountOf(value) {
297     let x = this.FindFirstOf(value);
298     if (x == null) return 0;
299     let rank_x = this.FindRankOf(x);
300     let y = this.FindLastOf(value);
301     let rank_y = this.FindRankOf(y);
302     return rank_y - rank_x + 1;
303 }
304
305 // The number of elements strictly less than value O(log n)
306 RankOf(value) {
307     let x = this.FindPrecursorOf(value);
308     return x == null ? 0 : this.FindRankOf(x) + 1;
309 }
310
311 // Get the k-th element (0-indexed) O(log n).
312 Kth(rank) {
313     let x = this.FindKth(rank);
314     return x == null ? null : (x.value_);
315 }
316
317 // Find the smallest element that is strictly greater than value > , if it exists O(log n).
318 SuccessorOf(value) {
319     let node = this.FindSuccessorOf(value);
320     return node == null ? null : (node.value_);
321 }
322
323 // Find the largest element that is strictly less than value < , if it exists O(log n).
324 PrecursorOf(value) {
325     let node = this.FindPrecursorOf(value);
326     return node == null ? null : (node.value_);
327 }
328
329 // Get sorted values in the splay tree O(n).
330 GetValues() {
331     let res = [];
332     const dfs = (x) => {

```


```
333         if (x == null) return;
334         dfs(x.left_);
335         res.push(x.value_);
336         dfs(x.right_);
337     };
338     dfs(this.root_);
339     return res;
340 }
341 }
342
343 const numberOfPairs = (a, b, diff) => {
344     let tree = new SplayTree(), n = a.length, res = 0;
345     let d = [];
346     for(let i = 0; i < n; i++) d.push(a[i]-b[i]);
347     for (let i = 0; i < n; i++) {
348         let x = d[i], y = x + diff;
349         res += tree.RankOf(y + 1);
350         // pr(tree.RankOf(y+1));
351         tree.Insert(x);
352     }
353     return res;
354 };
```

☐ Custom Testcase[Use Example Testcases](#)[Run](#)[Submit](#)**Submission Result: Accepted** (/submissions/detail/877142049/) ⓘ[More Details >](#) (/submissions/detail/877142049/)

Share your acceptance!

◀ 1

Copyright © 2023 LeetCode

[Help Center \(/support\)](#) | [Jobs \(/jobs\)](#) | [Bug Bounty \(/bugbounty\)](#) | [Online Interview \(/interview/\)](#) | [Students \(/student\)](#) | [Terms \(/terms\)](#) | [Privacy Policy \(/privacy\)](#) [United States \(/region\)](#)