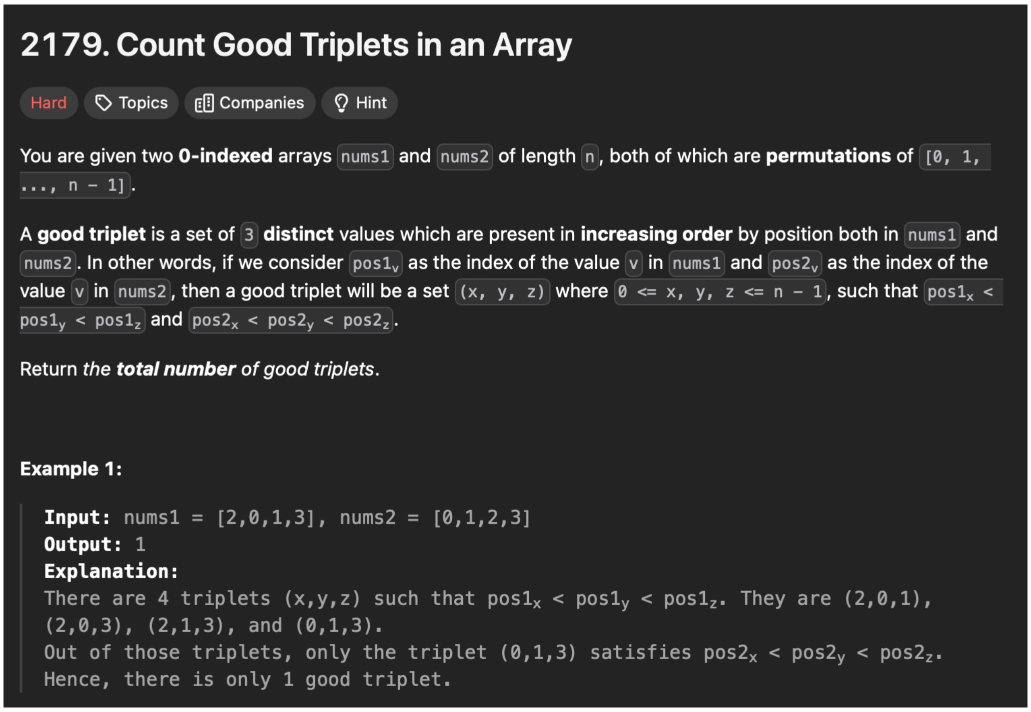


2179. Count Good Triplets in an Array

You are given two **0-indexed** arrays `nums1` and `nums2` of length `n`, both of which are **permutations** of `[0, 1, ..., n - 1]`.

A **good triplet** is a set of `3` **distinct** values which are present in **increasing order** by position both in `nums1` and `nums2`. In other words, if we consider $pos1_v$ as the index of the value `v` in `nums1` and $pos2_v$ as the index of the value `v` in `nums2`, then a good triplet will be a set `(x, y, z)` where `0 <= x, y, z <= n - 1`, such that $pos1_x < pos1_y < pos1_z$ and $pos2_x < pos2_y < pos2_z$.

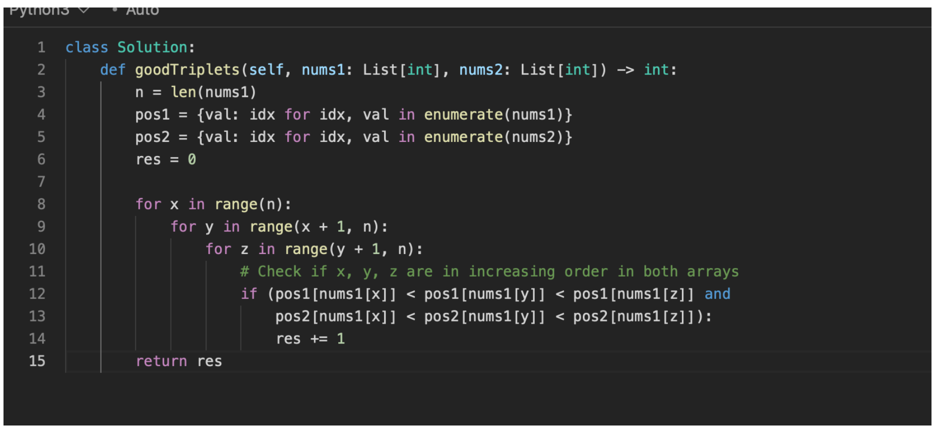
Return *the **total number** of good triplets*.

Example 1:

> **Input:** nums1 = [2,0,1,3], nums2 = [0,1,2,3]
> **Output:** 1
> **Explanation:**
> There are 4 triplets (x,y,z) such that $pos1_x < pos1_y < pos1_z$. They are (2,0,1), (2,0,3), (2,1,3), and (0,1,3).
> Out of those triplets, only the triplet (0,1,3) satisfies $pos2_x < pos2_y < pos2_z$.
> Hence, there is only 1 good triplet.

① nums1 → all triplets → find pos_v in nums2

nums1 (x. y. z) → check $pos2_x$. $pos2_y$.



```python
class Solution:
    def goodTriplets(self, nums1: List[int], nums2: List[int]) -> int:
        n = len(nums1)
        pos1 = {val: idx for idx, val in enumerate(nums1)}
        pos2 = {val: idx for idx, val in enumerate(nums2)}
        res = 0

        for x in range(n):
            for y in range(x + 1, n):
                for z in range(y + 1, n):
                    # Check if x, y, z are in increasing order in both arrays
                    if (pos1[nums1[x]] < pos1[nums1[y]] < pos1[nums1[z]] and
                        pos2[nums1[x]] < pos2[nums1[y]] < pos2[nums1[z]]):
                        res += 1
        return res
```

Time Limit Exceeded

② Binary Index Tree (BIT)

AKA Fenwick Tree

- good for :

O(logN) (1° updating value at an index

2° querying the sum (or count) of values in a prefix range

Fenwick Tree

- normally お sum :

n:

1	2	3	4	5	6	7

sum:

1	3	6	10	15	21	28

if n[5] update, takes O(n) to update sum.

- a little bit better

n:

1	2	3	4	5	6	7

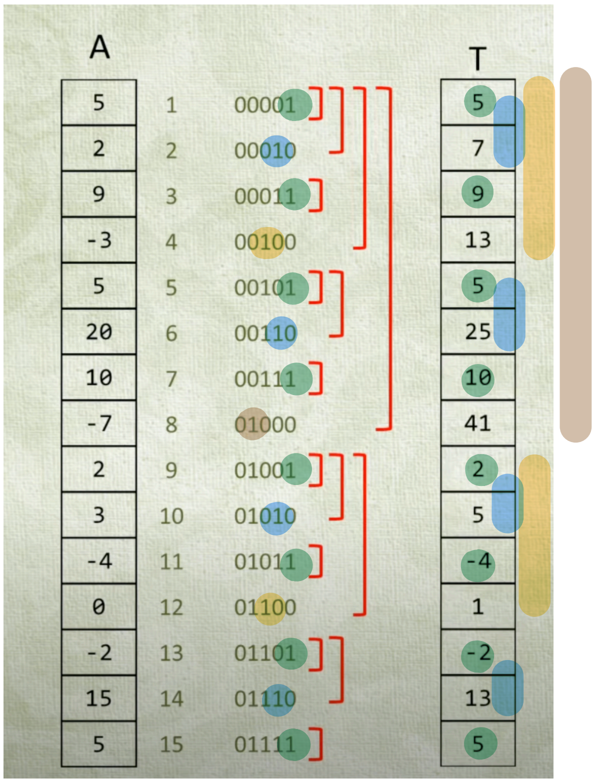
sum1:

1	3	6	10

sum2:

15	21	28

- ultimate : Fenwick Tree



A			T
5	1	00001	5
2	2	00010	7
9	3	00011	9
-3	4	00100	13
5	5	00101	5
20	6	00110	25
10	7	00111	10
-7	8	01000	41
2	9	01001	2
3	10	01010	5
-4	11	01011	-4
0	12	01100	1
-2	13	01101	-2
15	14	01110	13
5	15	01111	5

Eg. sum(7) = sum(00111)

= T[00111] +

T[00110] +

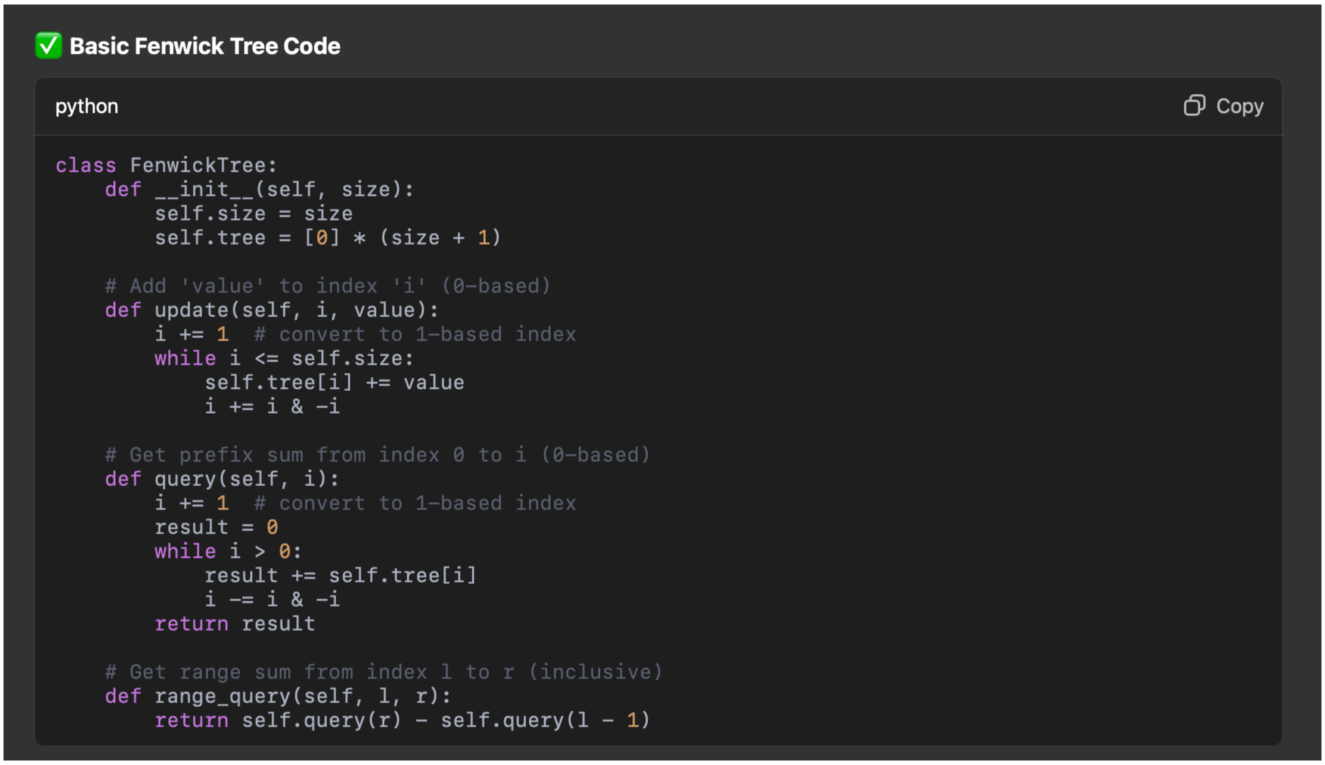
T[00100]

= T[7] + T[6] + T[4]

= range(7,7) + range(5,6)

+ range(1,4)

= 10 + 25 + 13 = 48



✅ Basic Fenwick Tree Code

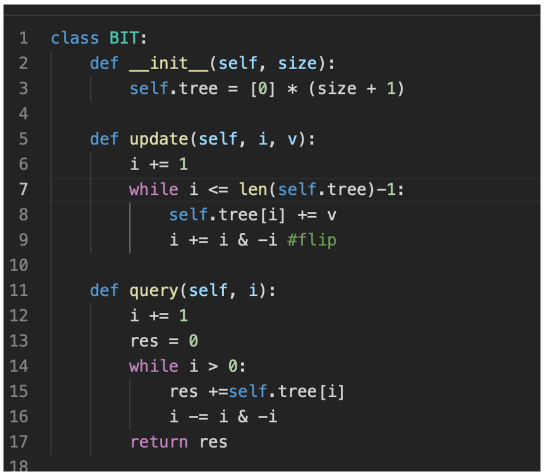
```python
class FenwickTree:
    def __init__(self, size):
        self.size = size
        self.tree = [0] * (size + 1)

    # Add 'value' to index 'i' (0-based)
    def update(self, i, value):
        i += 1  # convert to 1-based index
        while i <= self.size:
            self.tree[i] += value
            i += i & -i

    # Get prefix sum from index 0 to i (0-based)
    def query(self, i):
        i += 1  # convert to 1-based index
        result = 0
        while i > 0:
            result += self.tree[i]
            i -= i & -i
        return result

    # Get range sum from index l to r (inclusive)
    def range_query(self, l, r):
        return self.query(r) - self.query(l - 1)
```

⇒

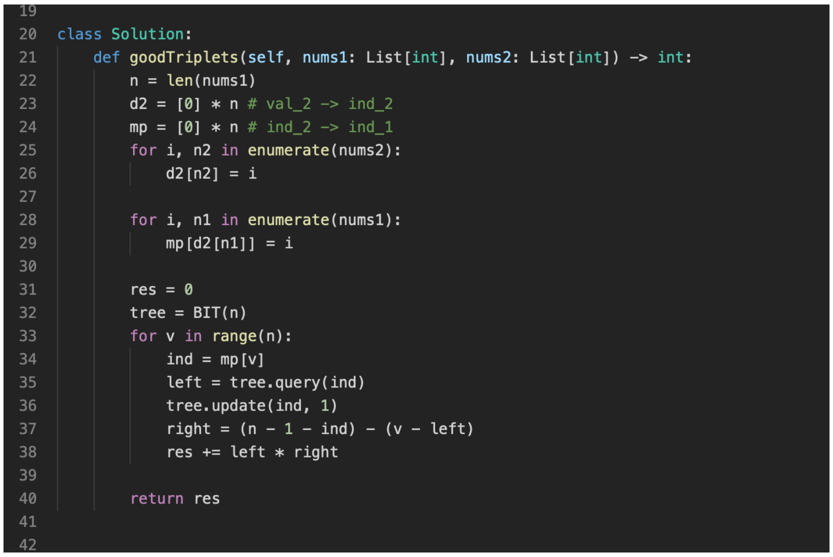


```python
class BIT:
    def __init__(self, size):
        self.tree = [0] * (size + 1)

    def update(self, i, v):
        i += 1
        while i <= len(self.tree)-1:
            self.tree[i] += v
            i += i & -i #flip

    def query(self, i):
        i += 1
        res = 0
        while i > 0:
            res +=self.tree[i]
            i -= i & -i
        return res
```

Common BIT initialization

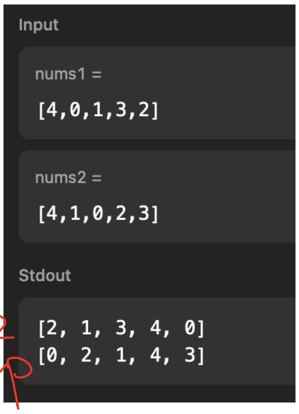


```python
class Solution:
    def goodTriplets(self, nums1: List[int], nums2: List[int]) -> int:
        n = len(nums1)
        d2 = [0] * n # val_2 -> ind_2
        mp = [0] * n # ind_2 -> ind_1
        for i, n2 in enumerate(nums2):
            d2[n2] = i

        for i, n1 in enumerate(nums1):
            mp[d2[n1]] = i

        res = 0
        tree = BIT(n)
        for v in range(n):
            ind = mp[v]
            left = tree.query(ind)
            tree.update(ind, 1)
            right = (n - 1 - ind) - (v - left)
            res += left * right

        return res
```



Input

nums1 =

```
[4,0,1,3,2]
```

nums2 =

```
[4,1,0,2,3]
```

Stdout

```
d2 [2, 1, 3, 4, 0]
mp [0, 2, 1, 4, 3]
```