Day-17 Python DSA

Count Inversions

https://www.geeksforgeeks.org/problems/inversion-of-array-1587115620/1

Bruteforce

```
def inversionCount(arr):
    n = len(arr)
    count=0
    for i in range(0,n):
        for j in range(i+1, n):
            if arr[i] > arr[j]:
                count+=1
    return count
arr= [2, 4, 1, 3, 5]
inversionCount(arr)
```

TC – O(N^2)

SC – O(1)

Optimal

```
from typing import List, Tuple
```

```
class Solution:
    def mergeList(self, arr1: List[int], arr2: List[int]) -> Tuple[List[int], int]:
        n = len(arr1)
        m = len(arr2)
        i, j = 0, 0
        count = 0
        result = []

        while i < n and j < m:
            if arr1[i] <= arr2[j]:
                result.append(arr1[i])
                i += 1
            else:
                count += n - i  # Count inversions
                result.append(arr2[j])
                j += 1

        while i < n:
            result.append(arr1[i])
            i += 1
        while j < m:
            result.append(arr2[j])
            j += 1

        return result, count
    def mergeSort(self, lst: List[int]) -> Tuple[List[int], int]:
        if len(lst) <= 1:
            return lst, 0  # Return count as 0

        mid = len(lst) // 2
        first_half = lst[:mid]
        second_half = lst[mid:]

        fh, cnt1 = self.mergeSort(first_half)
        sh, cnt2 = self.mergeSort(second_half)

        merged, count = self.mergeList(fh, sh)
        return merged, cnt1 + cnt2 + count

    def inversionCount(self, arr: List[int]) -> int:
        _, count = self.mergeSort(arr)
        return count
```

Dry Run

arr = [2, 4, 1, 3, 5]

We'll follow **merge sort** flow and keep track of:

- The current split
- Merge results
- Inversion count additions

Step 1 – Call inversionCount

inversionCount([2, 4, 1, 3, 5])

- Calls mergeSort([2, 4, 1, 3, 5])

Step 2 – First split

mergeSort([2, 4, 1, 3, 5])

- mid = 2
- first_half = [2, 4]
- second_half = [1, 3, 5]

Step 3 – Sort [2, 4]

mergeSort([2, 4])

- mid = 1
- first_half = [2] → returns ([2], 0)
- second_half = [4] → returns ([4], 0)

Merge step:

mergeList([2], [4])
→ Compare 2 ≤ 4 → append 2
→ Append 4
→ result = [2, 4], count = 0

Return: ([2, 4], 0)

Step 4 – Sort [1, 3, 5]

mergeSort([1, 3, 5])

- mid = 1
- first_half = [1] → returns ([1], 0)
- second_half = [3, 5]

Sort [3, 5]

mergeSort([3, 5])

- mid = 1
- first_half = [3] → ([3], 0)
- second_half = [5] → ([5], 0)

Merge step:

mergeList([3], [5])
→ Compare 3 ≤ 5 → append 3
→ Append 5
→ result = [3, 5], count = 0

Return: ([3, 5], 0)

Merge [1] and [3, 5]

mergeList([1], [3, 5])
→ Compare 1 ≤ 3 → append 1
→ Append 3, 5
→ result = [1, 3, 5], count = 0

Return: ([1, 3, 5], 0)

Step 5 – Merge [2, 4] and [1, 3, 5]

mergeList([2, 4], [1, 3, 5])

- Compare **2 > 1** → inversion count += (n - i) = (2 - 0) = 2

Append 1

result = [1], count = 2

- Compare 2 ≤ 3 → append 2

result = [1, 2]

- Compare 4 > 3 → inversion count += (2 - 1) = 1

Append 3

result = [1, 2, 3], total count so far = 3

- Append 4

Append 5

result = [1, 2, 3, 4, 5]

Return: ([1, 2, 3, 4, 5], count = 3)

Step 6 – Final count

- cnt1 = 0 (from left part [2,4])
- cnt2 = 0 (from right part [1,3,5])
- count from merge = 3

Total inversions = 0 + 0 + 3 = 3

Output:

inversionCount([2, 4, 1, 3, 5]) → 3

TC – O(N log N)

SC- O(N)