Cheat Sheet: Merge Sort-based Problems (e.g., Inversion Count) - Java DSA

1. Introduction

Merge Sort is a **Divide and Conquer** algorithm that can be adapted to solve various problems efficiently. One such application is **counting inversions** in an array, which measures how far the array is from being sorted.

2. Problem: Inversion Count

Given an array `arr[]`, find the number of **inversions** in it.

Inversion: For `i < j`, an inversion is a pair `(arr[i], arr[j])` such that `arr[i] > arr[j]`.

3. Brute Force Approach

Description:

Check all pairs `(i, j)` and count those satisfying `arr[i] > arr[j]`.

Code:

```
public int countInversionsBruteForce(int[] arr) {
    int count = 0;
    for (int i = 0; i < arr.length - 1; i++) {
        for (int j = i + 1; j < arr.length; j++) {
            if (arr[i] > arr[j]) count++;
        }
    }
    return count;
}
```

Time and Space:

- Time: $O(n^2)$
- Space: $O(1)$

4. Optimal Approach: Modified Merge Sort

Description:

Modify Merge Sort to count inversions during the merge step. When an element from the right subarray is placed before an element from the left subarray, it's an inversion.

Code:

```
public class InversionCount {
    public int countInversions(int[] arr) {
        return mergeSort(arr, 0, arr.length - 1);
    }

    private int mergeSort(int[] arr, int left, int right) {
        int count = 0;
        if (left < right) {
            int mid = (left + right) / 2;
            count += mergeSort(arr, left, mid);
            count += mergeSort(arr, mid + 1, right);
            count += merge(arr, left, mid, right);
        }
        return count;
    }

    private int merge(int[] arr, int left, int mid, int right) {
        int[] temp = new int[right - left + 1];
        int i = left, j = mid + 1, k = 0, count = 0;

        while (i <= mid && j <= right) {
            if (arr[i] <= arr[j]) {
                temp[k++] = arr[i++];
            } else {
                temp[k++] = arr[j++];
                count += (mid - i + 1); // Count inversions
            }
        }

        while (i <= mid) temp[k++] = arr[i++];
        while (j <= right) temp[k++] = arr[j++];

        System.arraycopy(temp, 0, arr, left, temp.length);
        return count;
    }
}
```

Time and Space:

- Time: O(n log n)
- Space: O(n)

5. Example

```
Input: arr[] = [2, 4, 1, 3, 5]
Output: 3
Explanation: Inversions are (2,1), (4,1), (4,3)
```

6. Complexity Comparison

Approach	Time Complexity	Space Complexity
Brute Force	$O(n^2)$	O(1)
Merge Sort-based	O(n log n)	O(n)

7. Applications

- Measure how far array is from sorted
- Used in inversion-sensitive sorting
- Important in ranking algorithms, genomics

8. Practice Problems

1. Count Inversions - GFG
2. Leetcode Hard - Reverse Pairs
3. Smallest Element on Right Side that is Greater