public class InversionCount {

public static void main(String[] args) {

int[] arr = {2, 5, 3, 1, 10};

int n = arr.length;

int inversionCount = countInversions(arr);

System.out.println("Number of inversions: " + inversionCount);

}

// Wrapper method

public static int countInversions(int[] arr) {

int[] temp = new int[arr.length];

return mergeSortAndCount(arr, temp, 0, arr.length - 1);

}

// Modified merge sort to count inversions

private static int mergeSortAndCount(int[] arr, int[] temp, int left, int right) {

int mid, count = 0;

if (right > left) {

mid = (left + right) / 2;

count += mergeSortAndCount(arr, temp, left, mid);

count += mergeSortAndCount(arr, temp, mid + 1, right);

count += mergeAndCount(arr, temp, left, mid + 1, right);

}

return count;

}

// Merge step that counts the inversions

private static int mergeAndCount(int[] arr, int[] temp, int left, int mid, int right) {

int i = left; // Left subarray pointer

int j = mid; // Right subarray pointer

int k = left; // Temp array pointer

int count = 0;

while (i <= mid - 1 && j <= right) {

if (arr[i] <= arr[j]) {

temp[k++] = arr[i++];

} else {

temp[k++] = arr[j++];

count += (mid - i); // Elements remaining in left array are greater

}

}

// Copy remaining elements

while (i <= mid - 1) {

temp[k++] = arr[i++];

}

while (j <= right) {

temp[k++] = arr[j++];

}

// Copy back to original array

for (i = left; i <= right; i++) {

arr[i] = temp[i];

}

return count;

}

}

import java.util.stream.IntStream;

public class StreamInversionCount {

public static void main(String[] args) {

int[] arr = {2, 5, 3, 1, 10};

long count = IntStream.range(0, arr.length)

.boxed()

.flatMap(i ->

IntStream.range(i + 1, arr.length)

.filter(j -> arr[i] > arr[j])

.mapToObj(j -> {

System.out.println("Inversion: (" + arr[i] + ", " + arr[j] + ")");

return new int[]{arr[i], arr[j]};

})

)

.count();

System.out.println("Total number of inversions: " + count);

}

}