Count Inversions in an array (Using Merge sort)

```
#include <stdio.h>
#include <stdlib.h>

int merge(int *arr, int *temp, int left, int middle, int right)
{
	int leftIndex, rightIndex, result, inversionCount = 0;

	leftIndex = left;		// leftIndex is left sub array index
	rightIndex = middle;  // rightIndex is right sub array index
	result = left;			// result index starts from left sub array

	while( (leftIndex <= middle -1) && (rightIndex <= right))
	{
		if(arr[leftIndex] <= arr[rightIndex])
			temp[result++] = arr[leftIndex++];
		else
		{
			temp[result++] = arr[rightIndex++];
			inversionCount = inversionCount + (middle - leftIndex);
		}
	}

	//copy remaining elements of left subarray

	while(leftIndex <= middle -1)
		temp[result++] = arr[leftIndex++];

	//copy remaining elements of right sub array

	while(rightIndex <= right)
		temp[result++] = arr[rightIndex++];

	for(leftIndex = left; leftIndex <= right; leftIndex++)
		arr[leftIndex] = temp[leftIndex];

	return inversionCount;
}

int mergeSort(int *arr, int *temp, int left, int right)
{
	int middle, inversionCount = 0;

	if(left < right)
	{
		//divide the array into two parts
		middle = (left + right) / 2;

		//inversion count

		inversionCount = mergeSort(arr, temp, left, middle);
```

```
                inversionCount += mergeSort(arr, temp, middle + 1, right);

                //merge left array and right array

                inversionCount += merge(arr, temp, left, middle+1, right);
        }
        return inversionCount;
}


int countInversions(int *arr, int size)
{
        int *temp = (int *)malloc(size * sizeof(int));
        return mergeSort(arr, temp, 0, size-1);
}

int main()
{
        int *arr, size;
        printf("Enter size of an array\n");
        scanf("%d", &size);
        //allocate memory for array
        arr = (int *)malloc(size * sizeof(int));

        printf("Enter Array elements");
        for(int index = 0; index < size; index++)
                scanf("%d", &arr[index]);
        printf("Total Number of Inversions = %d", countInversions(arr, size));
        return 0;
}
```
Time Complexity: O(nlogn)