```cpp
//https://www.geeksforgeeks.org/counting-inversions/

/*
 * Inversion Count for an array indicates – how far (or close) the array
 * is from being sorted. If array is already sorted then inversion count
 * is 0. If array is sorted in reverse order that inversion count is the
 * maximum. Formally speaking, two elements a[i] and a[j] form an
 * inversion if a[i] > a[j] and i < j
 *
 * Example:
 * The sequence 2, 4, 1, 3, 5 has three inversions
 * (2, 1), (4, 1), (4, 3).
 */

#include <bits/stdc++.h>

/* This funt merges two sorted arrays and returns inversion count in
   the arrays.*/
int merge (int arr[], int temp[], int left, int mid, int right) {
    int i, j, k;
    int inv_count = 0;
    i = left; /* i is index for left subarray*/
    j = mid;  /* j is index for right subarray*/
    k = left; /* k is index for resultant merged subarray*/

    while ( (i <= mid - 1) && (j <= right) ) {
        if (arr[i] <= arr[j]) {
            temp[k++] = arr[i++];
        } else {
            temp[k++] = arr[j++];
            /*this is tricky -- see above explanation/diagram for merge()*/
            inv_count = inv_count + (mid - i);
        }
    }

    /* Copy the remaining elements of left subarray
     (if there are any) to temp*/
    while (i <= mid - 1)
        temp[k++] = arr[i++];

    /* Copy the remaining elements of right subarray
     (if there are any) to temp*/
    while (j <= right)
        temp[k++] = arr[j++];

    /*Copy back the merged elements to original array*/
    for (i = left; i <= right; i++)
        arr[i] = temp[i];

    return inv_count;
}

/* An auxiliary recursive function that sorts the input array and
  returns the number of inversions in the array. */
int _mergeSort (int arr[], int temp[], int left, int right) {
    int mid, inv_count = 0;

```

```
58      if (right > left) {
59          /* Divide the array into two parts and call _mergeSortAndCountInv()
60             for each of the parts */
61          mid = (right + left) / 2;
62          /* Inversion count will be sum of inversions in left-part, right-part
63            and number of inversions in merging */
64          inv_count  = _mergeSort (arr, temp, left, mid);
65          inv_count += _mergeSort (arr, temp, mid + 1, right);
66          /*Merge the two parts*/
67          inv_count += merge (arr, temp, left, mid + 1, right);
68      }
69
70      return inv_count;
71  }
72
73  /* This function sorts the input array and returns the
74     number of inversions in the array */
75  int mergeSort (int arr[], int array_size) {
76      int *temp = (int *) malloc (sizeof (int) * array_size);
77      return _mergeSort (arr, temp, 0, array_size - 1);
78  }
79
80  /* Driver program to test above functions */
81  int main () {
82      int arr[] = {1, 20, 6, 4, 5};
83      printf ("Number of inversions are %d \n", mergeSort (arr, 5) );
84      getchar();
85      return 0;
86  }
87
88
89  // Output: Number of inversions are 5
90
```