Count Inversions in an array | Set 1 (Using Merge Sort) - GeeksforGeeks

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).

METHOD 1 (Simple)
For each element, count number of elements which are on right side of it and are smaller than it.

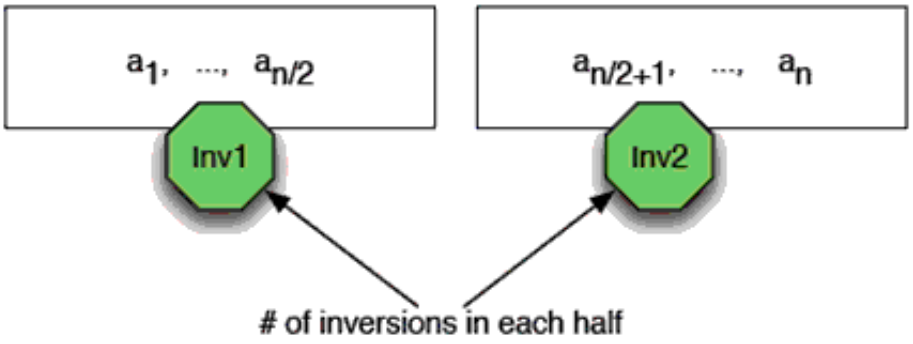
```
#include <bits/stdc++.h>
int getInvCount(int arr[], int n)
{
  int inv_count = 0;
  for (int i = 0; i < n - 1; i++)
    for (int j = i+1; j < n; j++)
      if (arr[i] > arr[j])
        inv_count++;
  return inv_count;
}/* Driver progra to test above functions */
int main(int argv, char** args)
{
  int arr[] = {1, 20, 6, 4, 5};
  int n = sizeof(arr)/sizeof(arr[0]);
  printf(" Number of inversions are %d \n", getInvCount(arr, n));
  return 0;
}
```

```
Number of inversions are 5
```

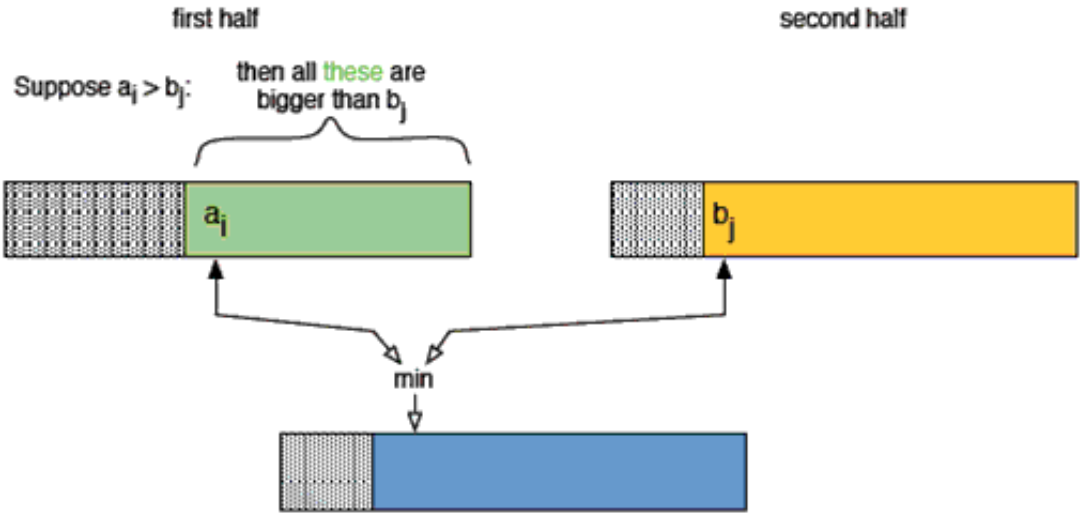
Time Complexity: O(n^2)

METHOD 2(Enhance Merge Sort)
Suppose we know the number of inversions in the left half and right half of the array (let be inv1 and inv2), what kinds of inversions are not accounted for in Inv1 + Inv2? The answer is - the inversions we have to count during the merge step. Therefore, to get number of inversions, we

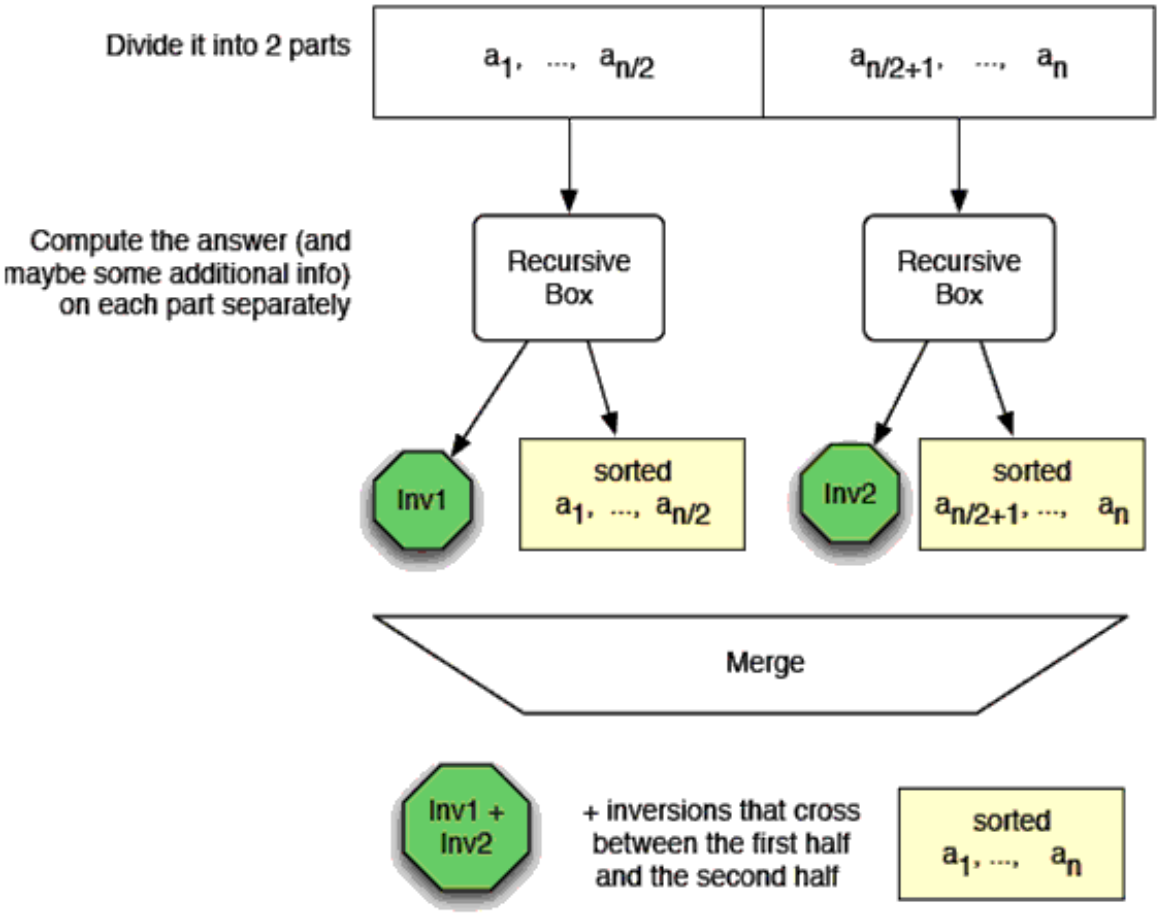
need to add number of inversions in left subarray, right subarray and merge().



How to get number of inversions in merge()?
In merge process, let i is used for indexing left sub-array and j for right sub-array. At any step in merge(), if a[i] is greater than a[j], then there are (mid – i) inversions. because left and right subarrays are sorted, so all the remaining elements in left-subarray (a[i+1], a[i+2] … a[mid]) will be greater than a[j]



The complete picture:



Implementation:

```
#include <bits/stdc++.h>
int  _mergeSort(int arr[], int temp[], int left, int right);
int merge(int arr[], int temp[], int left, int mid, int right);
/* This function sorts the input array and returns the
   number of inversions in the array */
int mergeSort(int arr[], int array_size)
{
    int *temp = (int *)malloc(sizeof(int)*array_size);
    return _mergeSort(arr, temp, 0, array_size - 1);
}/* 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;
  if (right > left)
  {
    /* Divide the array into two parts and call _mergeSortAndCountInv()
       for each of the parts */
    mid = (right + left)/2;
```

```
    /* Inversion count will be sum of inversions in left-part, right-
part
      and number of inversions in merging */
    inv_count  = _mergeSort(arr, temp, left, mid);
    inv_count += _mergeSort(arr, temp, mid+1, right);
    /*Merge the two parts*/
    inv_count += merge(arr, temp, left, mid+1, right);
  }
  return inv_count;
}/* 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;  /* i is index for right subarray*/
  k = left; /* i 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;
}/* Driver progra to test above functions */
```

```
int main(int argv, char** args)
{
  int arr[] = {1, 20, 6, 4, 5};
  printf(" Number of inversions are %d \n", mergeSort(arr, 5));
  getchar();
  return 0;
}
```