**Brief Report of Count Inversion Code (Merge Sort)**

Divide-and-conquer strategy (using inv\_count to count inversions.)

**1. Divide: mergeSort() function is used to separate array[] into two equal size subarrays.**

**Divide: O(1).**

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 mergeSort and merge\_count()

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\_count(arr, temp, left, mid + 1, right);

}

return inv\_count;

}

2. Conquer: recursively count inversions in each half. **Conquer: 2T (n / 2)**

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 mergeSort and merge\_count()

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\_count(arr, temp, left, mid + 1, right);

}

return inv\_count;

}

3. Combine: count inversions where a[i] and a[j] are as the initially index in different halves and return sum of three quantities.

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 mergeSort and merge\_count()

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\_count(arr, temp, left, mid + 1, right);

}

return inv\_count;

}

**3.Combine: count different half inversions**

■ Assume each half is sorted.

■ Count inversions where a[i] and a[j] are in different halves. In merge process, let i is used for indexing left sub-array and j for right sub-array. At any step in merge\_count(), 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]

**Count: O(n)**

Refer to code: inv\_count = inv\_count + (mid - i);

■ Merge two sorted halves into sorted whole. **Merge: O(n)**

int merge\_count(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++];

//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]

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;

}

**Time Complexity of the worst case**

In the worst-case scenario, the numbers would be **totally inverted**, so instead of having a list of numbers as: [ 1 , 2 , 3 , 4 , 5 , 6 ], we would have it as [ 6 , 5 , 4 , 3 , 2 , 1 ]. The mathematical formula for calculating the number of inversions given a list of length n and assuming that the numbers are totally inverted is n(n-1) / 2.

T(n) <=T(n/2) + T(n/2) + O(n) =2T(n/2) +O(n)

a=2, b=2, d=1

logba=1=d

Thus => O (nlog n)