Name: Vivek Vikram Pundkar

Roll no: 77

GR no: 11910860

Division: C

Batch: 3

**Inversion Count**

--------------------------------------------------------------------------------------------------------------

**Title:** Take an integer array and then count no of inversions in that array

**Algorithm:**

int merge(int originalArr[], int temp[], int start, int mid, int end)

{

int i = start;

int j = mid;

int k = start;

int inversion\_count = 0;

while(i <= mid-1 && j<= end)

{

if(originalArr[i]< originalArr[j])

temp[k++] = originalArr[i++]

else{

inversion\_count += mid-i;

temp[k++] = originalArr[j++]

}

}

while(i <= mid-1)

temp[k++] = originalArr[i++]

while(j <= end)

temp[k++] = originalArr[j++]

for(int i = start; i <= end; i++)

originalArr[i] = temp[i]

return inversion\_count;

}

int mergeSort(int originalArr[], int temp[], int start, int end)

{

int inversion\_count = 0;

if(start < end) {

int mid = (start+end)/2;

inversion\_count += mergeSort(originalArr, temp, start, mid);

inversion\_count += mergeSort(originalArr, temp, mid+1, end);

inversion\_count += merge(originalArr, temp, start, mid+1, end);

}

return inversion\_count;

}

int inversionCount(int A[], int size)

{

int temp[size]

return mergeSort(A, temp, 0, size-1)

}

**Steps:**

* Split our array into two halves just like merge sort
* Recursion will continue until we get one element.
* counting the number of inversions in the first half, second half and the number of inversions during the merge process.
* Use two pointers, i and j.  i will point to the starting element of the left half and j will point to the starting element of the second half. compare the elements at both the positions.

If ith element is smaller than jth element, add it to the new sorted list. Else, increment the count of inversions by (mid-i).

**Analysis:**

**Time Complexity:** O(n log n),

The algorithm used is divide and conquer, So in each level, one full array traversal is needed, and there are log n levels, so the time complexity is O(n log n).

**Space Complexity:** O(n), Temporary array.

**Output:**

