PAGE No.
DATE / /

Experiment task - 1

// Brute force solution
// Input: Array of no.
// output: no. of inverted values

```
Func CountInv (arr):
    invCount = 0
    for (i=0 → len(arr)-1)
        for (j=i+1 → len(arr)-1)
            if arr[i] > arr[j]
                invCount++;
    return invCount;
```

Time Complexity: Since there are two nested loop the time complexity is $O(n^2)$

PAGE No.
DATE / /

Optimal Solution

```
// Input : Array of no.
// output : no. of inversions of value

Func Merge and Count (arr)
    if (len(arr) < 2) return 0

    Mid = len(arr) /2

    l, leftInv = MeargAndCount (arr[0, mid])
    r, rightInv = MergeAndCount (arr[0, len(arr)])

    Merge, SplitInv = Merge (l, r)

    totalInv = leftInv + rightInv + SplitInv
    return merged, totalInv.

Func merge (left, right)
// input : 2 arrays
// output : merge array and count of inv.
merge = []
i=0, j=0
SplitInv = 0
while i < len(left) && j < len(right)
    if left[i] <= right[j]
        merged.append (left[i])
        i++;
    else
        merged append (right[j])
        j++;
    append remaining element of l & r to
    merged

return merged, splitInv
```

Func CountStudentInv(arr)
ZeroInv = 0
OneInv = 0
TwoInv = 0
ThreeInv = 0

For each sourcecode in arr
    inversion = MergeAndCount(arr)
if inversion == 0;
    ZeroInv += 1
else if inversion == 1;
    OneInv += 1
else if inversion == 2;
    twoInv += 1
else if inversion == 3;
    ThreeeInv += 1

Time Complexity using piggyback on merge sort

Divide Step: array is recursively divided in half
$O(\log n)$

Conquer Step: During merge process → $O(n)$

∴ Total time ⇒ $T(n) = 2T(n/2) + O(n)$

Using Master Theorum: $T(n) \Rightarrow O(n \log n)$

PAGE NO.
DATE / /

Experiment task = 2

```
Func manualMultiply (num1, num2)
// input : 2 int with digits in 2 array
// output : product.

For (i = len(num1)-1 → 0)
    for (j = len(num2)-1 → 0)
        mul = num1[i] * num2[j]
        position1 = i+j
        position2 = i+j+1
        Sum = mul + result[position2]
        result[position2] = Sum % 10
        result[position1] += Sum / 10
```

Time Complexity $O(n_1 \times n_2)$ where n_1 and n_2 are the lengths of the input arrays.

⇒ Optimal Solution :

```
Func Karatsuba (x,y)
// inpt : 2 large integers as array of digits
// output : product

if (x<10 or y<10) return x*y;
m = max( len(x), len(y) )
half = m/2
highX, lowX = divmod(x, 10^half)
highY, lowY = divmod(y, 10^half)

z0 = Karatsuba (lowX, lowY)
z1 = Karatsuba (lowX+highX, lowY+highY)
z2 = Karatsuta (highX, highY)

return (z2 * 10^(2*half) + [z1-z2-z0]* 10^half
                                    + z0)
```

⇒ Time Complexity:

Divide and Conquer: Reduce multiplication of two n-digit number into three multiplication of n/2 digits no. This is crucial improvement over four multi. of traditional method.

$$T(n) = 3T(n/2) + O(n)$$

Using master theorum: $T = n^{\log_2 3} = n^{1.585}$

* Test Cases :-

(1) Arr 1 = {5, 4, 4, 3}
Arr 2 = {5, 4, 4, 3}
Output = 25626249

(2) Arr 2 = {0}
Arr 1 = {9, 9, 9, 1, 9, 9}
Output = 0

(3) Arr 1 = {9, 9, 2, 1, 0}
Arr 2 = {2, 3}
Output = 2281830

(4) Arr 1 = {8, 6, 5, 4, 1}
Arr 2 = {1, 1, 1}
Output = 960651

(5) Arr 1 = {-8, 6, 5, 4}
Arr 2 = {2, 1}
Output = -181734