Experiment 1 :

- Non Optimal Solution

```
// input : Array of number
// output : Number of inverted value & index pair.

func countbrute (arr)
{
    int count = 0;
    for (i:0 → len(arr)-1)
        for ( j: j+1 → len (arr) -1)
            if arr[i] > arr[j]
                inv count ++
        return invcount
}
```

- Time Complexity :

no of loops : 2

∴ time complexity = n^2

∴ Solution is non-optimal.

Optimal Solution

```
// Input : Array of number
// output : Number of inverted value

func mergeandcount (arr)
 if (len(arr) < 2)
  { return 0 }
 eles
  {
    mid = len(arr) /2
    l, leftInv = mergeandcount (arr[0:mid])
    r, rightInv = mergeandcount ( arr [0: len(arr)]
    merged, splitInv = merge ( l, r)
    totalInv = leftInv + rightInv + splitInv.
    return merged, TotalInv
  }

func merge( left, right)
 // input : two arrays.
 // output: merged arr & count of Inv.
    merge = []
    i:=0, j=0
    splitInv = 0
```

```
while i < len(left) && j < len(right)
  {
    if left[i] <= right[j]
     merge.append(left[i])
     i++
    else
      merged appenend(right[i])
       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
three Inv += 1

Time complexity using piggyback on mergesort

Divide step: array is recursively divide in half until it reached subarray of size 1;
∴ O(n) = O(log n)

Conquer step: During a merge process, two sorted halves are combined this requires time complexity of r(n)

∴ Total time complexity
T(n) = 2T(n/2) + O(n)
using masters theorme
a = 2, b = 2, d = 1
$a = b^d$ as $2 = 2^2$
$T(n) = a(b \log b) \log_a b + d$

= O(log n × n)
= O(n log n)

Experiment 2:

- // Input : 2 integer with digit in 2 arr
// output: product of two integer.

```
func manualMultiply (num1, num2)
{
  for (i: len (Num2)-1 → 0)
    for (j: len( Num2)-1 → 0)
      mul = num1[i] + num2[j]
      position 1 = i+j;
      position 2 = i+j+1;
      sum = mul + result [position 2]
    result [position 2] = sum .1. 10
    result [position 1] = sum .1. 10
}
```

- Time Complexity.

∴ no. of loops = 2

As there are two for loops.

∴ time complexity = $n \times n = n^2$.

∴ It is not an optimal solution.

__/__/__

- Optimal Solution

```
Func Karatsuba (x,y)
{
  if (x<10 or y<10)
    {
      return x*y
    }
    m = max (len(x), len(y));
    half = m/2;
    high x, low x = div mod (x, 10^half)
    high y, low y = div mod (y, 10^half)

    Z0 = Karatsuba (low x, low y)
    Z1 = Karatsuba (low x + high x, low y + low y)
    Z2 = Karatsuba (high x, high y)
}
return (Z2 * 10^(2*half) + (Z1 - Z2 - Z0) *
                    10^half + Z0)
```

- Time Complexity.

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

Recursive formula:

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

using master theorme.

$a = 3 \quad b = 2, \quad U(n) = 1$

$a > b^d \quad$ as $(3 > 2)$

$\therefore T(n) = O(n^{\log_b a})$

$= O(n^{\log_2 3})$

$= n^{1.585}$.

★ Test Cases.

1) Arr 1 = { 5, 3, 9, 2 }
Arr 2 = { 2, 7, 5, 6 }
output : 14860352

2) Arr 1 = { 5, 7, 3, 2 }
Arr 2 = { 0 }
output = 0

3) Arr 1 = { -8, 6, 5, 4 }
Arr 2 = { 6, 5, 4 }
output : -5659916

4) Arr 1 = {-1, 1, 1, 1, 1}
Arr 2 = {2, 2, -2, 2, 2}
Output = 2468 64 642.

5) Arr 1 = {5, 7, 5, -7}
Arr 2 = {0}
Output = 0