DAA-LAB-04:

PAGE NO.:
DATE / /

(a) INVERSION COUNT.

(1.) ALGORITHM – Regular:

```
// Input: array with choices of all the students
// Output: Total count inversions.
// This function calculates count inversions by brute force

def CI-brute (students[]):
    for s in students:
        n, count = len(s), 0
        for i in range(n):
            for j in range (i+1, n):
                if S[i] > S[j]:
                    count++;

        total += count
    return total
```

(2.) ALGORITHM – DAC with mergesort:

```
def mergesort (S[], l, r):
    count = 0
    if left < right:
        mid = (l+r)/2
        count += mergesort (S[], l, mid)
        Count += mergesort (S[], mid+1, r)
        count += merge (S[], l, m, r)
```

PAGE NO.:
DATE / /

```
def merge(s[], l, m, r):
    i = l
    j = m+1
    k = l

    inv_count = 0

    while i<=mid && j<=r:
        if s[i] <= s[j]:
            temp[k] = s[i]
            i++

        else{ count += (mid - i + 1)
            temp[k] = s[j]
            j++ }
        k++

    while i<=mid:
        temp[k++] = s[i++]

    while j<=r:
        temp[k++] = s[j++]

    for i in (l, r+1):
        s[i] = temp[i]
    return count
```

```
def CI_dAC():
    count = 0
    for s in students:
        for s in ~~do~~ students[]:
            count += mergesort(s[], 0, n-1);

    return count
```

(3.) Time Complexity.

LINEAR ⇒ It has 3 for loops that traverse across arrays. Let n be the length of individual array and m be the length of the dataset.

$\therefore T(n) = m \times n \times n$

$T(n) = O(m \cdot n^2)$

DAC ⇒ It uses mergesort algorithm which has 'nlogn' time complexity and this process is repeated 'm' times for m arrays in dataset.

$\therefore T(n) = m \times n \log n$

$T(n) = O(mn \log n)$

PAGE NO.:
DATE / /

4. TESTCASES :

1. Positive Testcases

All array values are integers and no values are missing. There are 100 such arrays in the dataset. Return and Display number of Total Inversions and students with 0, 1, 2, ... inversions.

2. Negative Testcases.

(1) If the array has non-int values
⟹ DISPLAY :- Incorrect input, choices ~~can't~~ have to be integers.

(2) If the array is empty
⟹ DISPLAY :- Dataset is empty

(3) Negative values of choices
⟹ DISPLAY :- Choices can't be negative.

(4) If array has 'zero' as a choice.
⟹ DISPLAY :- Source Codes cannot be zero

(b) KARATSUBA MULTIPLICATION

```
def multiplication(x, y)
    a, b = str(x), str(y)
    result = 0

    for i in range(len(b)):
        dB = int(b[-1-i])

        pp = 0
        carry = 0
        for j in range(len(a)):
            dA = int(a[-j-1])
            p = (dA * dB) + carry
            carry = p // 10
            p = p % 10

            pp += p * (10 ** j)

        if carry > 0:
            pp += carry * (10 ** len(a))

        pp *= (10 ** i)

        result += pp

    return result
```

(2.) ALGORITHM - Karatsuba (DAC)

// Input : Two n digit positive integer x and y

// Output : The product x * y .

// This function uses DAC (Karatsuba Algorithm) to calculate product.

```
karatsuba ( int a, int b):
    if a<10 or b<10 :
        return a*b
    else :
        m = n/2
        aH, aL = divmod ( a , 10**m)
        bH, bL = divmod ( b, 10**m)

        z0 = karatsuba (aL, bL)
        z1 = karatsuba (aL+aH, bL+bH)
        z2 = karatsuba(aH, bH)

        return (z2)*(10** n) + (z1-z2-z0)*(10**m)
                + z0
```

PAGE NO.:
DATE / /

(3) Time complexity:

In karatsuba algorithm we convert problem of size 'n' into problem of size $n/2$ and recursively calls the function thrice.

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

By master eqn:

$a = 3 \quad b = 2 \quad d = 1$

$b^d = 2, \quad a = 3$

$a > b^d$

$\therefore T(n) = n^{\log_b a} = n^{\log_2 3}$

$\therefore \boxed{T(n) = n^{1.585}}$

In regular multiplication, we use 2 loops to iterate through every digit of each number

multiplying each digit takes constant time.

$\therefore \quad TC = T(n) = O(n \times m)$ ∵ m, n are digits in the nos.

$\approx O(n^2)$ ... if $n \approx m$

PAGE NO.:
DATE / /

4. Testcases:

1) Positive Testcases:

(1) a = 1234567890123456
b = 9876543210987654
a*b = 12193263113702103443111363542576

(2) a = 9876543210123456
b = 1234567890987654
a*b = 12193263124676102548214078092576

(3) a = 1234567898765443
b = 9876543212345674
a*b = 12193263133333308663194987481

2) Negative Testcases

(1) a = 123456 (one of the number is zero)
b = 0
a*b = 0

(2) a = 123456 (number is not an integer)
b = ""
Both numbers should be integers.

(3) a = -1234 (number is negative)
b = 4567
a*b = -5635678.

(4) a = 12.3 b = 1234
DISPLAY: Both numbers should be integers.