The code follow PEP 8 guidelines.

Output:

```
C:\Users\Krish\AppData\Local\Programs\Python\Python312\python.exe "C:\Users\Krish\Python\Test\DAA LAB 4.py"
0 iversions occur for 1 times
1 iversions occur for 3 times
2 iversions occur for 10 times
3 iversions occur for 12 times
4 iversions occur for 19 times
5 iversions occur for 13 times
6 iversions occur for 15 times
7 iversions occur for 14 times
8 iversions occur for 10 times
9 iversions occur for 3 times
10 iversions occur for 0 times
```

```
C:\Users\Krish\Python\Test\.venv\Scripts\python.exe "C:\Users\Krish\Python\Test\DAA LAB 4.py"
Number of inversions: 10
Number of inversions: 6
Number of inversions: 1
Number of inversions: 0
Number of inversions: 8
Number of inversions: The array is not compatible!!
Number of inversions: The array is not compatible!!
Number of inversions: The array is not compatible!!
Number of inversions: The array is not compatible!!
Number of inversions: The array is not compatible!!
```

```
C:\Users\Krish\Python\Test\.venv\Scripts\python.exe "C:\Users\Krish\Python\Test\DAA LAB 4.py"
35937749302050
706776567696
69192562487288
959702547700
1753874300073288
Invalid Datatype
Invalid Datatype
Invalid Datatype
Invalid Datatype
Invalid Datatype

Process finished with exit code 0
```

```
C:\Users\Krish\Python\Test\.venv\Scripts\python.exe "C:\Users\Krish\Python\Test\DAA LAB 4.py"
35937749302050
706776567696
69192562487288
959702547700
1753874300073288
Invalid Datatype
Invalid Datatype
Invalid Datatype
Invalid Datatype
Invalid Datatype

Process finished with exit code 0
```

Conclusion:

This experiment shows the power of using divide and conquer approach over the brute force approach. The count inversion algorithm has time complexity of O(nlogn) same as merge sort because it leverages merge sort with just counting the number of inversion as additional feature. The brute force integer multiplication has time complexity of $O(n^2)$ and divide and conquer lowers it to $O(n^{\log 3})$. The number of inversions follows a normal distribution or a bell curve.

Counting Inversions
Course choice arrays of students
Output: Inversion count

```
Count(L,R):

    i,j <- 0,0
    n1,n2 <- len(L), len(R)
    count <- 0
    ans <- []

    while i != n1 or j != n2 do:

        if L[i] > R[j] then

            add R[j] to ans
            j++
            count <- n1 - i + count

        else

            add L[i] to ans
            i++

    if i == n1 then
        while j != n2 do

            add R[j] to ans
            j++
    if j == n2 then
        while i != n1 do
            add L[i] to ans
            i++
    return count
```

```
mergeandcount(arr):

    if len(arr) == 1:
        return 0

    mid ← len(arr) // 2

    inversions ← 0

    left = arr[ : mid]
    right = arr[mid : ]

    inversions = mergeandcount(left)
    inversions += mergeandcount(right)

    inversions += count(left, right)

    return inversions.
```

# Example:
Base = [1,2,3,4,5]

① [5,4,3,2,1] ⟶ count = 10

② [4,3,2,1,5] ⟶ count = 6

③ [1,2,3,5,4] ⟶ count = 1

④ [1,2,3,4,5] ⟶ count = 0

⑤ [5,3,4,1,2] ⟶ count = 8

⑥ [1, 2, 3, 4] → not compatible

⑦ [6, 2, 1, 4] → not compatible

⑧ [1, 2, 3, 4, 5, 6] → not compatible

⑨ [7, 8, 9, 10] → not compatible

⑩ [ ] → not compatible

Time complexity :

Count Inversions :

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

$2T(n/2)$ → 2 subproblem of length $n/2$

n → merge of sorted array and counting

$\therefore T(n) = aT(n/b) + f(n)$

$\therefore a = 2, b = 2, f(n) = n^d = n^1 \Rightarrow d = 1$

by master theorem,

$b^d = 2^1 = 2 = a$

$\therefore$ Time complexity $= O(n^d \log n)$

$= O(n \log n)$

Multiplication of 2 integers
Input: 2 integers
Output: Multiplicative product

Mul(a,b):

```
x ← min(a,b)
y ← max(a,b)
ans ← 0
i ← 0
for each bit of y do
    partial ← bit x x
    partial ← partial x 10^i
    i ← i+1
    ans ← ans + partial
return ans
```

Integer Multiplication (divide and conquer)
Input : 2 integers
Output : multiplication answer of 2 integers

Mul(a, b) :

if $a < 10$ or $b < 10$:
    return $a \times b$

smaller multiplication can be done easily

$n1 \leftarrow$ no of digits in a
$n2 \leftarrow$ no of digits in b

$n \leftarrow \max(n1, n2) // 2$

$x1 \leftarrow a // 10^n$
$x0 \leftarrow a \% 10^n$
$y1 \leftarrow b // 10^n$
$y0 \leftarrow b \% 10^n$

$p = mul(x1, y1)$
$q = mul(x0, y0)$
$r = mul(x1+x0, y1+y0) - p - q$

return $p \times 10^{2n} + r \times 10^n + q$

Test case:

① a = 4554210
b = 7891105 → 35937749302050

② a = 234578
b = 975432 → 228814887696

③ a = 9821204
b = 7045222 → 69192562487288

④ a = 966425
b = 993044 → 959702547700

⑤ a = 70854699
b = 24753112 → 1753874300073288

⑥ a = 5678.00
b = 60799.17 → ~~not~~ invalid datatype

⑦ a = 333.333
b = 666.666 → invalid datatype

⑧ a = 7974.17
b = 93210.22 → invalid datatype

⑨ a = 12345.17
b = 54312.61 → invalid datatype

⑩ a = 786132.14
b = 215473.66 → invalid datatype.

Time Complexity:

basic operation: addition of 2 bits

each integer is 'n' bit long.

Therefore, the ~~outer~~ loop runs 'n' times and in the loop the line where the partial products are added also costs O(n) time as there are n ~~bits~~ ~~to~~ addition to be performed.

∴ Time complexity is $O(n^2)$

Time complexity:

Karatsuba Algorithm

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

↓ 3 subproblems of length $n/2$

↓ addition of p, q, r

$T(n) = aT(n/b) + f(n)$

$\therefore a = 3, b = 2, f(n) = n^1 = n^d \quad \therefore d = 1$

By master theorem,

∴ Time complexity $= O(n^{\log_2 3})$

$[\because b^d = 2^1 = 2 < a]$