EXPERIMENT NO. 4

NAME: ADITYA B. KULKARNI
REGISTRATION NO : 241070908
SUBJECT : DAA (LAB)
SY BTECH COMPUTER ENGINEERING

Aim:

1. To find inversion count of course choice of students.
2. To multiply two integers using brute force and divide and conquer method

Theory:

Counting Inversions in Student Course Code Choices

Counting inversions can be efficiently performed with a modified version of the Merge Sort algorithm. Merge Sort itself is based on the divide-and-conquer approach, where:

1. **Divide:** The array is split into two halves recursively until each half has a single element (a base case, as a single element is inherently sorted).
2. **Conquer:** Each sub-array is then sorted, and while merging the sorted arrays, we count the inversions.

Combine: The results from each recursive call are combined to get the sorted array, and the total inversion count is accumulated

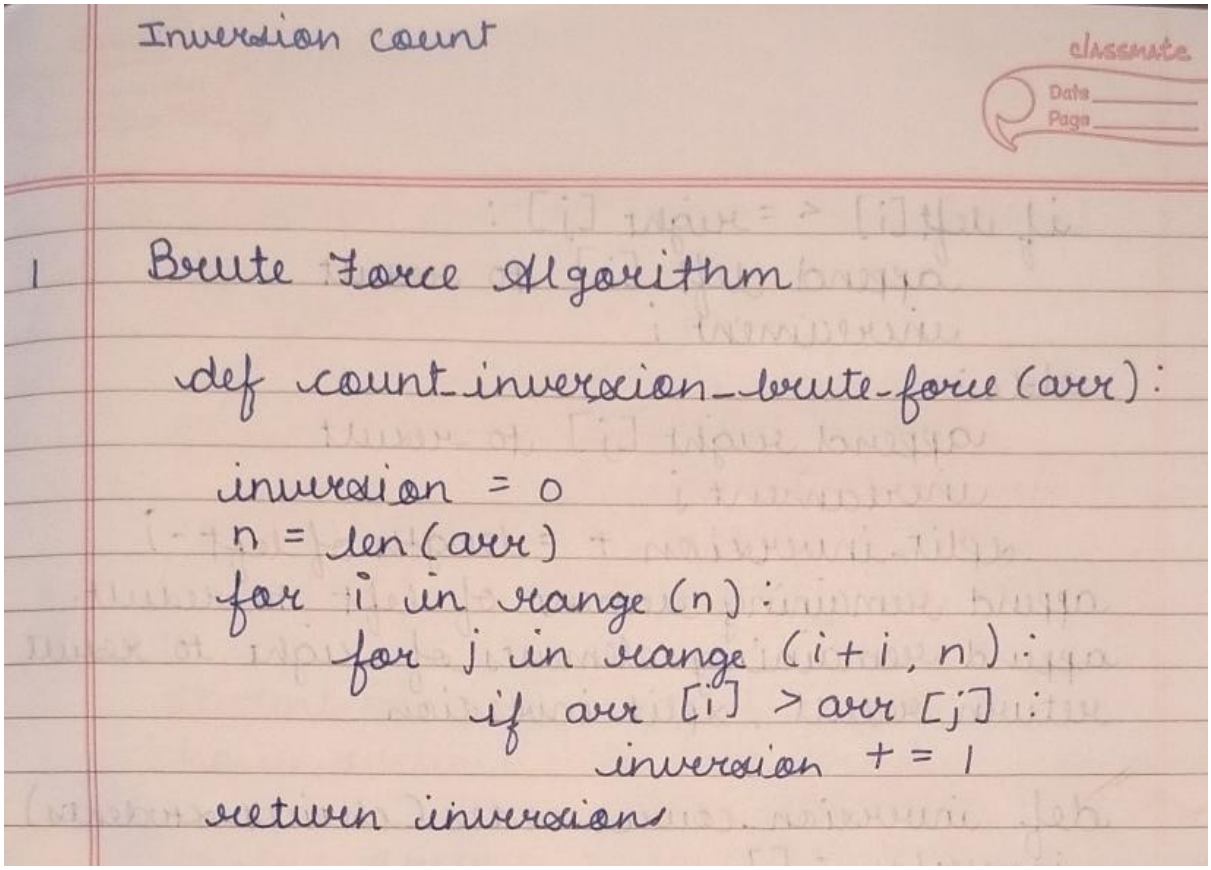
Steps in Counting Inversions:

1. **Divide the Array:** We divide the array of course codes of all students into two halves.
2. **Merge Sort and Count:** As we recursively merge each half, we use two pointers:
    - Pointer i scans from the start to the midpoint of the array.
    - Pointer j scans from the midpoint + 1 to the end.
    - If an element at i is greater than an element at j, it indicates an inversion.
    - The inversion count can be calculated as mid−i+1,as all elements from iii to the midpoint are greater than j.

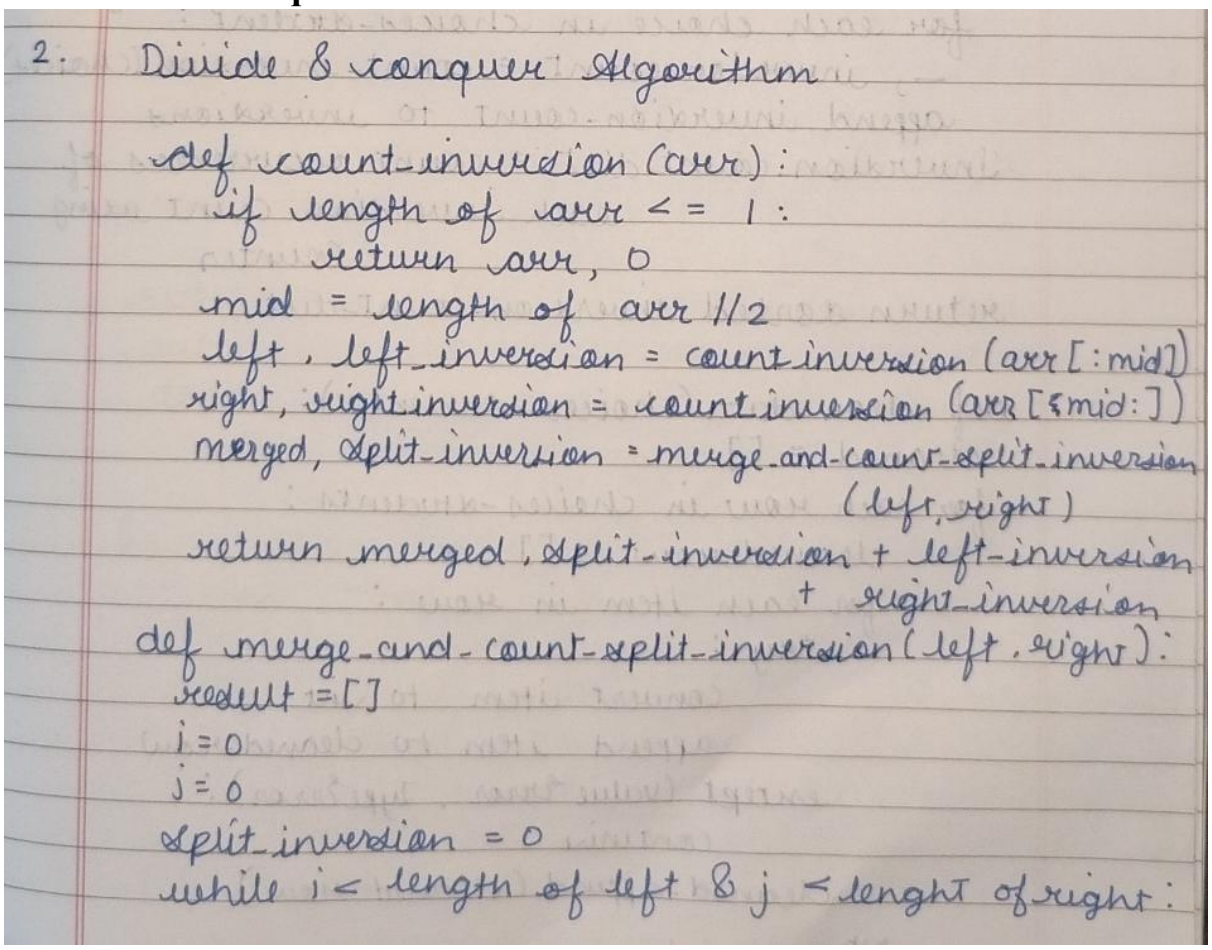
3. **Combine Counts:** As the recursive calls return, we accumulate the inversion count from each sub-array.

Algorithm (Inversion Count) :

1.Brute Force



2.Divide and Conquer



2. Divide & conquer Algorithm

```
def count-inversion (arr):
    if length of arr <= 1:
        return arr, 0
    mid = length of arr //2
    left, left_inversion = count_inversion (arr[:mid])
    right, right_inversion = count_inversion (arr[mid:])
    merged, split-inversion = merge_and-count-split_inversion
                                (left, right)
    return merged, split-inversion + left-inversion
                                + right-inversion
def merge-and-count-split-inversion (left, right):
    result = []
    i = 0
    j = 0
    split_inversion = 0
    while i < length of left & j < lenght of right:
```

```
if left[i] <= right[j]:
    append left[i] to result
    increament i
else:
    append right[j] to result
    increament j
    split_inversion += length of left - i
append remaining elemen of left to result.
append remaining elements of right to result
return result, split_inversion

def inversion_course_codes(choices_students):
 inversion = []
 for each choice in choices_strident:
  _, inversion_count = count_inversion(choices)
  append inversion_count to inverdions
 inverdion_count_dict = count occurrences of
                        each inverdion count using
                                Counter
 return sorted inverdion_count_dict

def clean_data(choices_students):
 cleaned = []
 for each row in choices_students:
     cleaned_row = []
     for each item in row:
         try:
             convert item to int
             append item to cleaned_row
         except (value Error, TypeError):
             continue
     cleaned.append(cleaned row)
 return cleaned
```

```
def process_csv(file_name, description):
    try:
        df = pd.read_csv(file_name)
    except FileNotFoundError:
        print(f"Error: The file '{file_name}' was
                not found.")
        return

student_ids = df['student'].tolist()
choices_students = df.drop(columns=['student'])
                   .values.tolist()
choices_students = cleandata(choices_students)

print no. of student for each inversion count
```

Time Complexity (Inversion Count)

1.Brute Force

1 Time complexity Brute Force

outer loop (for i in range(n)) - The outer loop iterates through each element of array, so it runs n times where n is the length of array.

Inner loop (for j in range(i+1, n))
- total no. of iterations in inner loop
  $(n-1) + (n-2) + (n-3) + \dots + 1 + 0$
  equivalent to $n-1$

$(n-1) * n/2$

$\Rightarrow O(n^2)$

2.Divide and Conquer

2. Time complexity Divide & conquer

→

count_inversion (arr) follows divide & conquer approach

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

$2T(n/2)$ → term correspond to 2 recursive calls to count_inversion on left & right halves of array.

n → corresponds to merging step & counting split inversions. Merging of 2 sorted arrays takes linear time $O(n)$.

Apply Masters theorem

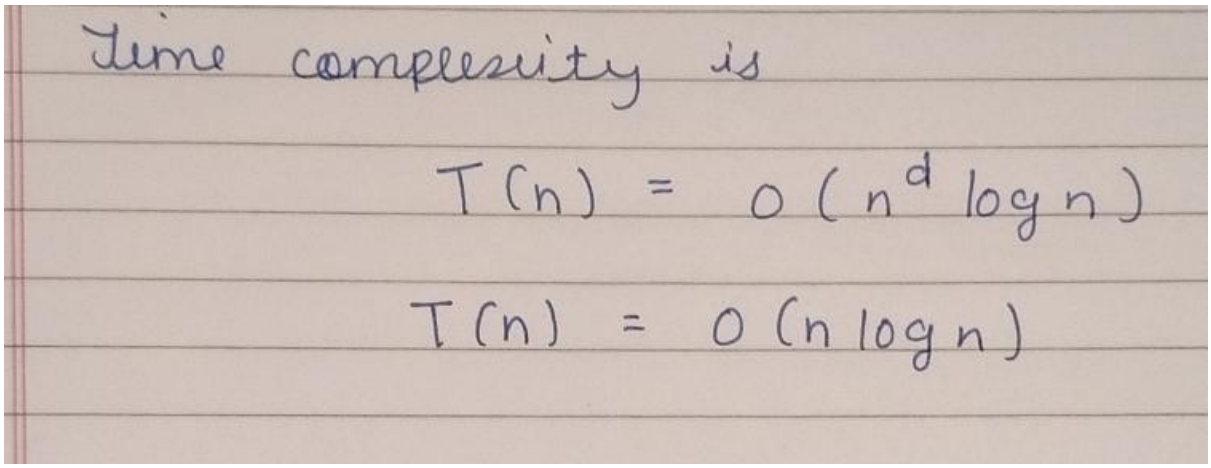
$$T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$$

$a = 2$ (array is divided in 2 subarrays)
$b = 2$ (size of each subarray is halved)
$d = 1$ (merging step takes linear time $O(n)$)

Now compare n^d with $n^{\log_b a}$

- $\log_b a = \log_2 2 = 1$
- $n^d = n^1$

$n^d = n^{\log_b a}$, this corresponds to case 2 of Master theorem which is $n^d = n \log_b a$



Code (Inversion count) :

1. Brute Force

```
def count_inversions_brute_force(arr):
    """Counts the number of inversions in an array using a brute-force
approach."""
    n = len(arr)
    inversions = 0
    for i in range(n):
        for j in range(i + 1, n):
            if arr[i] > arr[j]:
                inversions += 1
    return inversions

def inversions_course_codes_brute_force(choices_students):
    """Counts the number of inversions in a list of choices using brute force
    and classifies them according to the count of inversions."""
    inversions = []
    for choices in choices_students:
        t = count_inversions_brute_force(choices)
        inversions.append(t)
    count = dict(sorted(Counter(inversions).items()))
    # Creates a dictionary with key as the inversion count and value as the
number of students.
    return count

# Example data for testing
example_choices = [
    [101, 103, 102, 104],  # 1 inversion: (103, 102)
    [101, 102, 103, 104],  # 0 inversions
    [104, 103, 102, 101],  # 6 inversions: all pairs inverted
    [104, 102 ,101, 103],
```

```
    [101, 104, 102, 103],
]

for choices in example_choices:
    print(f"Array: {choices}, Inversions:
{count_inversions_brute_force(choices)}")
```

Output:

```
PS C:\Users\adity\OneDrive\Desktop\DAA lab> & "c:/Users/adity/OneD
_count_brute.py"
Array: [101, 103, 102, 104], Inversions: 1
Array: [101, 102, 103, 104], Inversions: 0
Array: [104, 103, 102, 101], Inversions: 6
Array: [104, 102, 101, 103], Inversions: 4
Array: [101, 104, 102, 103], Inversions: 2
PS C:\Users\adity\OneDrive\Desktop\DAA lab>
```

2. Divide and conquer:

```
from collections import Counter
import pandas as pd

def count_inversions(arr):
    """Counts the number of inversions in an array."""
    if len(arr) <= 1:
        return arr, 0
    mid = len(arr) // 2
    left, left_inversions = count_inversions(arr[:mid])
    right, right_inversions = count_inversions(arr[mid:])
    merged, split_inversions = merge_and_count_split_inversions(left, right)
    return merged, split_inversions + left_inversions + right_inversions

def merge_and_count_split_inversions(left, right):
    """Merges two arrays and counts the number of split inversions."""
    result = []
    i = j = split_inversions = 0
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
```

```
            split_inversions += len(left) - i
    result.extend(left[i:])
    result.extend(right[j:])
    return result, split_inversions

def inversions_course_codes(choices_students):
    """Counts the number of inversions in a list of choices and classifies
them
    according to the count of inversions."""
    inversions = []
    for choices in choices_students:
        _, t = count_inversions(choices)
        inversions.append(t)
    count = dict(sorted(Counter(inversions).items()))
    # Creates a dictionary with key as the inversion count and value as the
number of students.
    return count

def clean_data(choices_students):
    """Sanitizes the student choices by removing invalid data."""
    cleaned = []
    for row in choices_students:
        cleaned_row = []
        for item in row:
            try:
                # Attempt to convert to integer
                cleaned_row.append(int(item))
            except (ValueError, TypeError):
                continue
        cleaned.append(cleaned_row)
    return cleaned

def process_csv(file_name, description):
    """Processes a CSV file and prints the inversion counts."""
    try:
        # Load the CSV file
        df = pd.read_csv(file_name)
    except FileNotFoundError:
        print(f"Error: The file '{file_name}' was not found.")
        return

    # Extract student IDs and their course choices
    student_ids = df['Student'].tolist()
    choices_students = df.drop(columns=['Student']).values.tolist()

    # Clean and validate the data
    choices_students = clean_data(choices_students)
```

```
    # Remove empty rows
    # choices_students = [row for row in choices_students if row]

    # Print the number of students for each inversion count
    print(f"\n--- {description} ---")
    for k, v in inversions_course_codes(choices_students).items():
        print(f"{v:2d} students have {k:2d} inversion count.")

# Process the valid data file
process_csv('course_choice.csv', "Positive Test Cases (Valid Data)")

# Process the negative test cases file
process_csv('negative_course_choice.csv', "Negative Test Cases (Invalid/Edge
Data)")
```

Output :

Positive test cases :

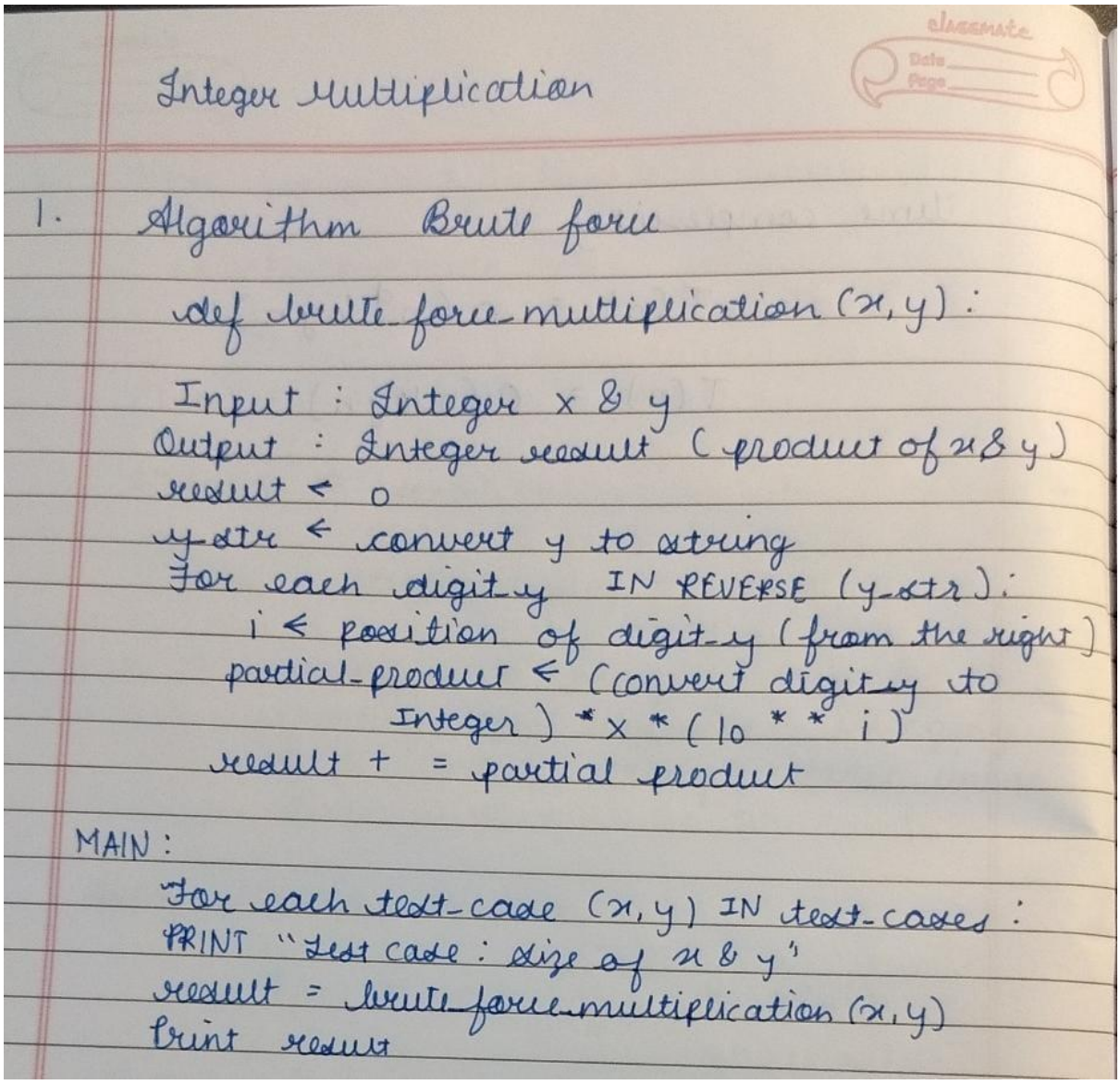
```
PS C:\Users\adity\OneDrive\Desktop\DAA lab> & "c:/Users/adity/OneDrive/Desktop/DAA lab
sktop/DAA lab/.venv/inversion_count.py"

--- Positive Test Cases (Valid Data) ---
 1 students have  0 inversion count.
 3 students have  1 inversion count.
 6 students have  2 inversion count.
 7 students have  3 inversion count.
 3 students have  4 inversion count.
11 students have  5 inversion count.
12 students have  6 inversion count.
 6 students have  2 inversion count.
 7 students have  3 inversion count.
 3 students have  4 inversion count.
11 students have  5 inversion count.
12 students have  6 inversion count.
10 students have  7 inversion count.
 6 students have  2 inversion count.
 7 students have  3 inversion count.
 3 students have  4 inversion count.
11 students have  5 inversion count.
12 students have  6 inversion count.
 3 students have  4 inversion count.
11 students have  5 inversion count.
12 students have  6 inversion count.
11 students have  5 inversion count.
12 students have  6 inversion count.
10 students have  7 inversion count.
20 students have  8 inversion count.
 9 students have  9 inversion count.
10 students have 10 inversion count.
 3 students have 11 inversion count.
 4 students have 12 inversion count.
 1 students have 13 inversion count.
```

Negative test cases:

```
--- Negative Test Cases (Invalid/Edge Data) ---
 2 students have  0 inversion count.
 6 students have  1 inversion count.
 2 students have  2 inversion count.
PS C:\Users\adity\OneDrive\Desktop\DAA lab>
```

Algorithm (Integer Multiplication):

1. Brute force



Integer Multiplication

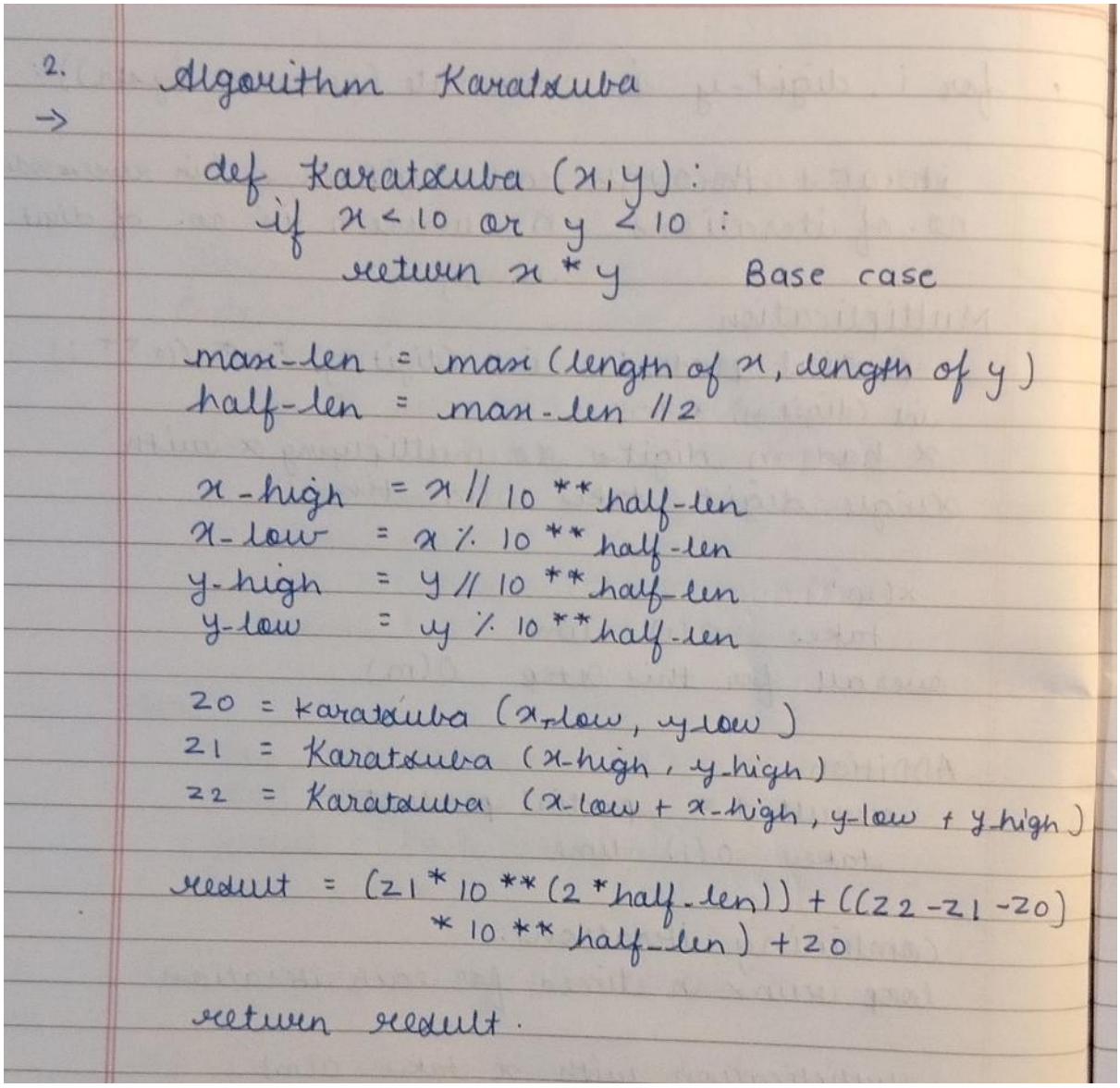
1. Algorithm Brute force

def brute_force_multiplication(x, y):

Input : Integer x & y
Output : Integer result (product of x & y)
result ← 0
y_str ← convert y to string
For each digit_y IN REVERSE (y_str):
i ← position of digit_y (from the right)
partial_product ← (convert digit_y to Integer) * x * (10 ** i)
result += partial product

MAIN:
For each test_case (x, y) IN test_cases:
PRINT "Test case : size of x & y"
result = brute_force_multiplication (x, y)
Print result

2. Karatsuba



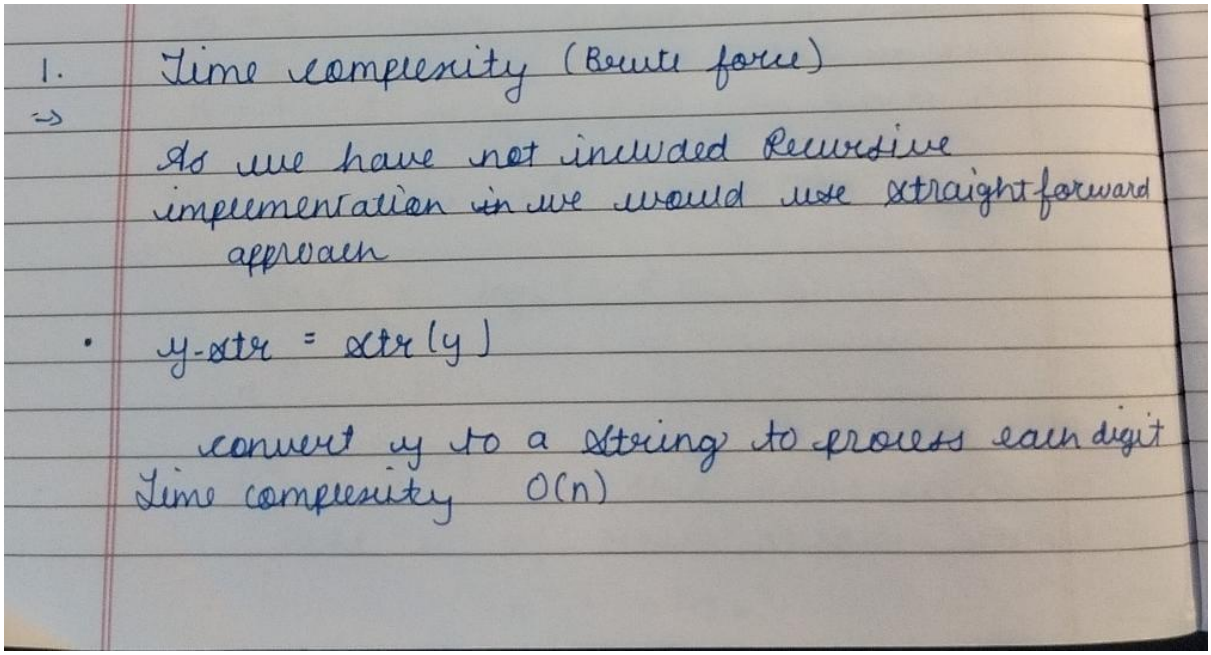
2.
→

Algorithm Karatsuba

```
def Karatsuba (x, y):
    if x < 10 or y < 10 :
        return x * y        Base case

max_len = max (length of x, length of y)
half_len = max_len // 2

x_high = x // 10 ** half_len
x_low = x % 10 ** half_len
y_high = y // 10 ** half_len
y_low = y % 10 ** half_len

z0 = Karatsuba (x_low, y_low)
z1 = Karatsuba (x_high, y_high)
z2 = Karatsuba (x_low + x_high, y_low + y_high)

result = (z1 * 10 ** (2 * half_len)) + ((z2 - z1 - z0)
         * 10 ** half_len) + z0

return result.
```

Time Complexity(Integer multiplication)

1.Brute force



- for i, digit_y in enumerate(reversed(y_str)):

iterates through each digit in y (in reverse order)
no. of iterations : n, where n is no. of digit in y.

Multiplication

partial product = int(digit_y) * x * (10 ** i)
int(digit_y) x x.
x had m digits, so multiplying x with single digit takes O(m) time.

x(10^i)
takes O(1) time
overall for this step O(m)

ADDition

result += partial product
takes O(1) time

Combining iterations.
loop runs n times for each iteration

Multiplication with x takes O(m)
Shifting & addition are O(1)
Time complexity per iteration O(m)

Total time complexity of loop O(n x m)

T(n,m) = O(n x m)

2.Karatsuba

2. Time complexity (Karatsuba)

divides problem in 3 recursive subproblem

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n)$$

$T(n)$ → time complexity for multiplying 2 no with n digit.

$3T(n/2)$ → cost of solving subproblem of size $n/2$

$O(n)$ → cost of spitting & combining result.

Master theorem

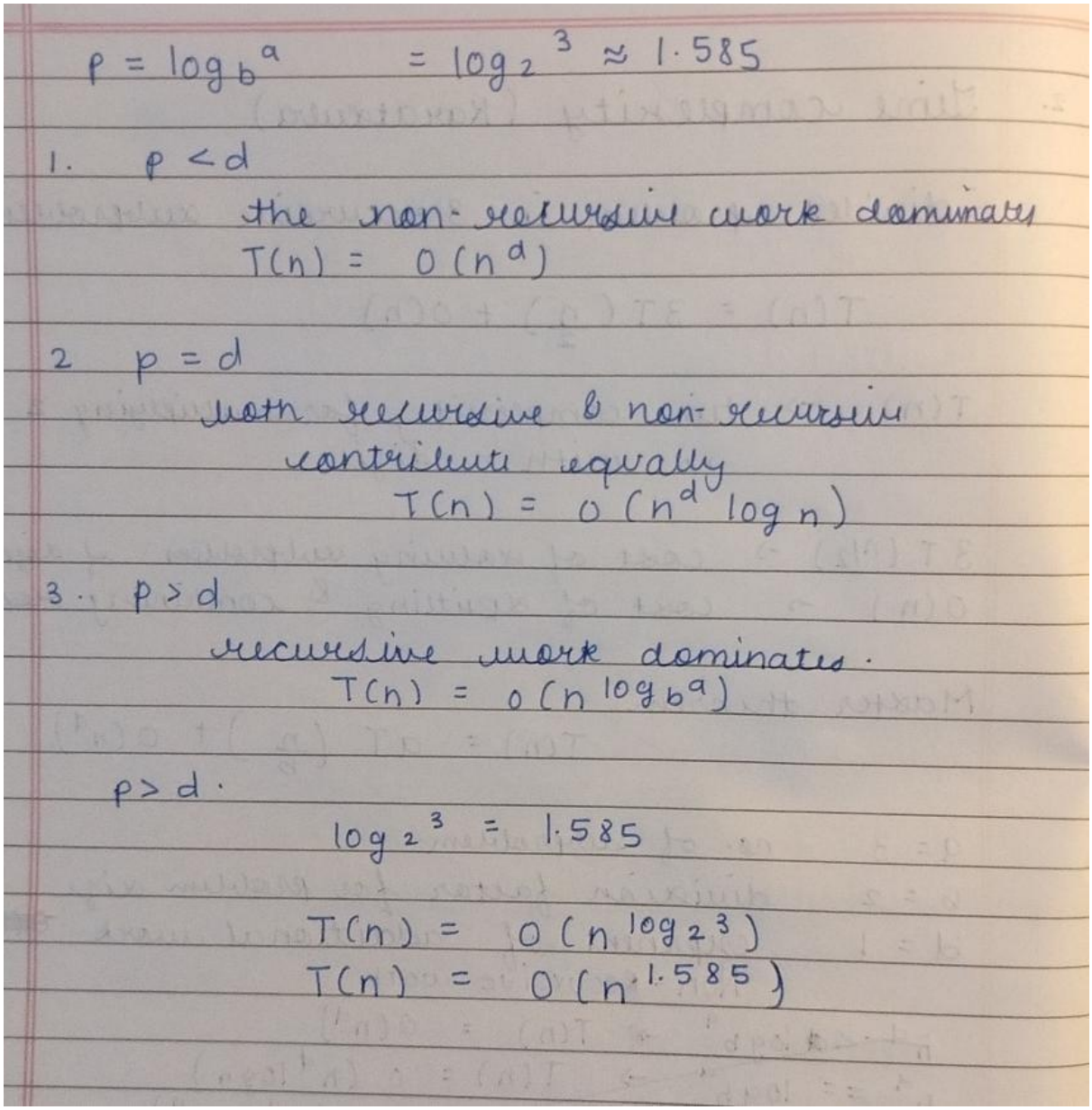
$$T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$$

$a = 3$ no. of subproblem

$b = 2$ division factor for problem size

$d = 1$ exponent of additional work ~~$O(n^1)$~~

Non-Recursive work



Code

1.Brute Force

```python
def brute_force_multiplication(x, y):
    """Multiplies two integers using brute force."""
    result = 0
    y_str = str(y)  # Convert y to string for processing
    for i, digit_y in enumerate(reversed(y_str)):
        # Multiply x with each digit of y and shift by the power of 10
        partial_product = int(digit_y) * x * (10 ** i)
        result += partial_product
    return result
```

```python
# Generate test cases with integers of different sizes
test_cases = [
    (int("9" * 10), int("8" * 10)),  # 10-digit numbers
    (int("9" * 50), int("8" * 50)),  # 50-digit numbers
    (int("9" * 100), int("8" * 100)),  # 100-digit numbers
    (int("9" * 500), int("8" * 500)),  # 500-digit numbers
    (int("9" * 1000), int("8" * 1000)),  # 1000-digit numbers
]

# Perform brute force multiplication for each test case and print results
print("--- Brute Force Multiplication Results ---")
for i, (x, y) in enumerate(test_cases, start=1):
    print(f"\nTest Case {i}: {len(str(x))}-digit numbers")
    result = brute_force_multiplication(x, y)
    print(f"Result: {result}")
```

Output:

```
PS C:\Users\adity\OneDrive\Desktop\DAA lab> & "c:/Users/adity/OneDrive/Desktop/DAA lab/.venv/Scripts/python.exe" "c:/Users/adity/OneDrive/Desktop/DAA lab/.venv/integer_mul_brute.py"
--- Brute Force Multiplication Results ---

Test Case 1: 10-digit numbers
Result: 88888888871111111112

Test Case 2: 50-digit numbers
Result: 8888888888888888888888888888888888888888888888888711111111111111111111111111111111111111111111111112

Test Case 3: 100-digit numbers
Result: 88888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888871111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111112

Test Case 4: 500-digit numbers
Result: 8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888711111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111112

Test Case 5: 1000-digit numbers
Result: 888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888871111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111112
PS C:\Users\adity\OneDrive\Desktop\DAA lab> 
```

Code

2.Karatsuba

```
def karatsuba(x, y):
    if x < 10 or y < 10:
        return x * y

    max_len = max(len(str(x)), len(str(y)))
    half_len = max_len // 2

    x_high = x // 10**half_len
    x_low = x % 10**half_len
    y_high = y // 10**half_len
    y_low = y % 10**half_len

    z0 = karatsuba(x_low, y_low)
    z1 = karatsuba(x_high, y_high)
    z2 = karatsuba(x_low + x_high, y_low + y_high)


    return (z1 * 10**(2 * half_len)) + ((z2 - z1 - z0) * 10**half_len) + z0


if __name__ == "__main__":
    x = 1111100000
    y = 1111111110
    result = karatsuba(x, y)
    print(f"The product of \n{x}\n and \n{y}\n is \n{result}.")
```

Output

10 digit:

```
20
21    if __name__ == "__main__":
22        x = 1111100000
23        y = 1111111110
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS 1 PORTS

```
teger_mul_karatsuba.py"
The product of
1111100000
 and
1111111110
 is
1234555554321000000.
PS C:\Users\adity\OneDrive\Desktop\DAA lab>
```

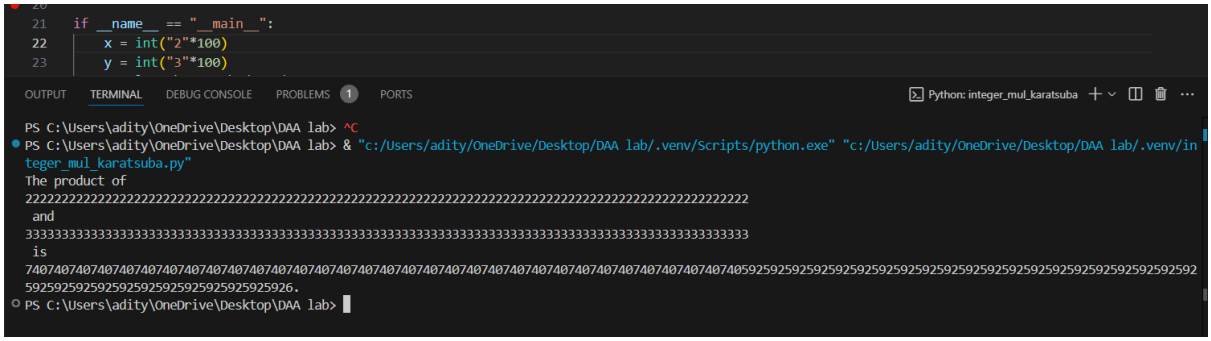
50 digit

```
20
21    if __name__ == "__main__":
22        x = 12345678909876543210123456790987654321012345678 90
23        y = 12121212121212121212121212121212121212121212121212121
```

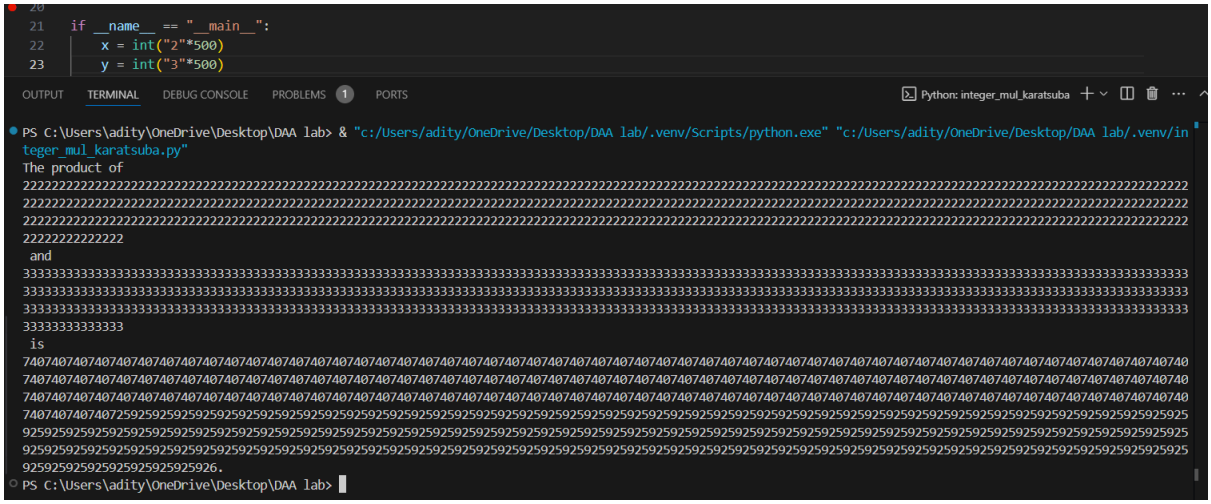
OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS 1 PORTS

```
PS C:\Users\adity\OneDrive\Desktop\DAA lab> ^C
PS C:\Users\adity\OneDrive\Desktop\DAA lab> & "c:/Users/adity/OneDrive/Desktop/DAA lab/.venv/Script
teger_mul_karatsuba.py"
The product of
12345678909876543210123456790987654321012345678 90
 and
12121212121212121212121212121212121212121212121212121
 is
14964459284698840254695099140591096146681631125936775159019117096894822297044335952113724653946 90.
PS C:\Users\adity\OneDrive\Desktop\DAA lab>
```

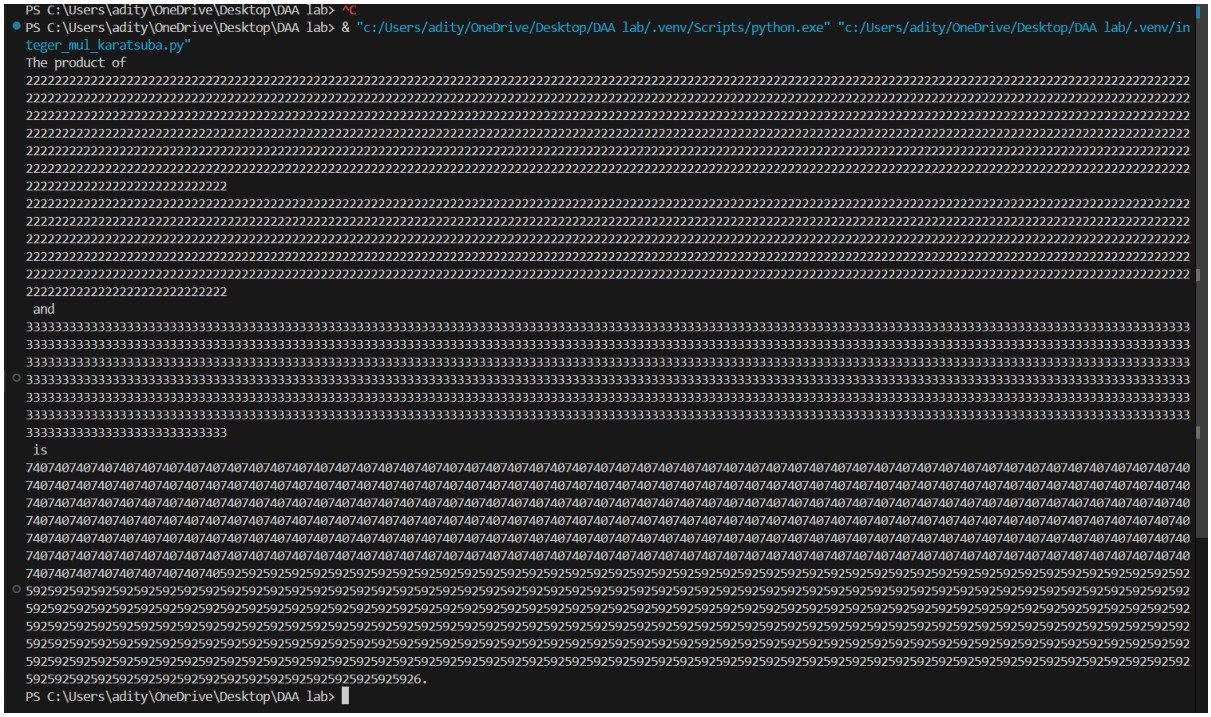
100 digit



500 digit



1000 digit



Conclusion: Hence in this practical we learnt about how to find **Inversion count using Brute force and Divide and conquer method** also found its time complexity , we also learnt how to do **integer multiplication using Karatsuba method** and found its time complexity