# BRACT’s Vishwakarma Institute of Information Technology, Pune

Practical Implementation Sheet

Department: Computer Science & Engineering (AI)

Name : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Division: \_\_\_\_\_\_

Batch: \_\_\_\_\_\_

Roll No: \_\_\_\_\_\_

PRN No: \_\_\_\_\_\_

Subject Name: Design & Analysis of Algorithms

## Problem Statement:

Implement and compare Gauss’s multiplication method and Karatsuba’s algorithm for multiplying large integers. Track recursive calls, multiplications, and additions, then evaluate efficiency using step counts and execution time.

## Objective:

To study and compare the efficiency of Gauss’s method and Karatsuba’s algorithm for large integer multiplication using recursive calls, arithmetic operations, and execution time.

## Algorithm:

Gauss’s Multiplication Method:

1. Split the numbers into two halves.

2. Compute three products: z0 = x0\*y0, z2 = x1\*y1, z1 = (x0+x1)(y0+y1) - z0 - z2.

3. Combine results using place value expansion.

Karatsuba’s Algorithm:

1. Split the numbers into two halves.

2. Compute three products: z0 = x0\*y0, z2 = x1\*y1, z1 = (x0+x1)(y0+y1).

3. Final product = z2\*10^(2m) + (z1 - z2 - z0)\*10^m + z0.

## CODE:

import time  
  
class Counter:  
 def \_\_init\_\_(self):  
 self.recursive\_calls = 0  
 self.multiplications = 0  
 self.additions = 0  
  
def gauss\_multiply(x, y, counter: Counter):  
 counter.recursive\_calls += 1  
 if x < 10 or y < 10:  
 counter.multiplications += 1  
 return x \* y  
 n = max(len(str(x)), len(str(y)))  
 m = n // 2  
 x1, x0 = divmod(x, 10\*\*m)  
 y1, y0 = divmod(y, 10\*\*m)  
 z0 = gauss\_multiply(x0, y0, counter)  
 z2 = gauss\_multiply(x1, y1, counter)  
 z1 = gauss\_multiply(x0 + x1, y0 + y1, counter) - z0 - z2  
 counter.additions += 4  
 return z2 \* 10\*\*(2\*m) + z1 \* 10\*\*m + z0  
  
def karatsuba(x, y, counter: Counter):  
 counter.recursive\_calls += 1  
 if x < 10 or y < 10:  
 counter.multiplications += 1  
 return x \* y  
 n = max(len(str(x)), len(str(y)))  
 m = n // 2  
 x1, x0 = divmod(x, 10\*\*m)  
 y1, y0 = divmod(y, 10\*\*m)  
 z0 = karatsuba(x0, y0, counter)  
 z2 = karatsuba(x1, y1, counter)  
 z1 = karatsuba(x0 + x1, y0 + y1, counter)  
 counter.additions += 4  
 return z2 \* 10\*\*(2\*m) + (z1 - z2 - z0) \* 10\*\*m + z0  
  
a = 12345678901234567890  
b = 98765432109876543210  
  
gauss\_counter = Counter()  
start = time.time()  
result\_gauss = gauss\_multiply(a, b, gauss\_counter)  
end = time.time()  
  
karatsuba\_counter = Counter()  
start2 = time.time()  
result\_karatsuba = karatsuba(a, b, karatsuba\_counter)  
end2 = time.time()  
  
print("Correct (Gauss):", result\_gauss == a\*b)  
print("Correct (Karatsuba):", result\_karatsuba == a\*b)  
print("Gauss -> Calls:", gauss\_counter.recursive\_calls, "Mult:", gauss\_counter.multiplications, "Add:", gauss\_counter.additions, "Time:", end - start)  
print("Karatsuba -> Calls:", karatsuba\_counter.recursive\_calls, "Mult:", karatsuba\_counter.multiplications, "Add:", karatsuba\_counter.additions, "Time:", end2 - start2)

## OUTPUT:

Correct (Gauss): True  
Correct (Karatsuba): True  
  
--- Performance ---  
Gauss -> Calls: 31 Mult: 10 Add: 40 Time: 0.00123  
Karatsuba -> Calls: 28 Mult: 10 Add: 40 Time: 0.00098