Assignment No. 01

- **TITLE –** Implement and compare Gauss's multiplication method and Karatsuba's algorithm for multiplying large integers. Track the number of recursive calls, multiplications, and additions in each method to analyse their performance. Evaluate and compare their efficiency using step counts and execution time.

- **THEORY –**

The goal of algorithms like Gauss's and Karatsuba's is to multiply large integers faster than the traditional "schoolbook" method, whose performance is $O(n^2)$. Both algorithms use a **Divide and Conquer** strategy, breaking the numbers down into smaller parts, solving them recursively, and then combining the results.

Gauss Multiplication: The Schoolbook Approach

This method splits two large numbers, x and y, into halves: a, b, c, and d. It calculates the final product by making **four** recursive calls to compute ac, ad, bc, and bd.

- **Formula:** result = ac * pow10(2*m) + (ad + bc) * pow10(m) + bd
- **Complexity:** Despite its recursive nature, this method still results in a time complexity of $O(n^2)$, offering no significant speed-up for large numbers.

Karatsuba Multiplication: The Optimized Approach

Karatsuba's algorithm is a more efficient take on the divide-and-conquer strategy. It cleverly reduces the number of required recursive multiplications from four to three.

- **The Trick:** Instead of calculating ad and bc separately, it computes just three products:
  1. ac = karatsuba(a, c)
  2. bd = karatsuba(b, d)
  3. abcd = karatsuba(a + b, c + d)

  The middle part is then found by subtracting the first two products from the third:
  (abcd - ac - bd).
- **Complexity:** This reduction to three recursive calls lowers the time complexity to approximately $O(n^{1.585})$, making it significantly faster for multiplying very large integers.

- **ALGORITHM –**

1. Gauss Multiplication Algorithm

- Step 1: If either number is a single digit, return their product.
  if x < 10 or y < 10 return x * y

- Step 2: Find number of digits n in larger number and take half m = n/2.
  n = max(length(x), length(y)); m = n/2

- Step 3: Split both numbers using 10^m.
  a = x / 10^m , b = x % 10^m
  c = y / 10^m , d = y % 10^m

- Step 4: Recursively compute four products.
  ac = gauss(a, c)
  ad = gauss(a, d)
  bc = gauss(b, c)
  bd = gauss(b, d)

- Step 5: Combine results using Gauss formula.
  result = ac * 10^(2*m) + (ad + bc) * 10^m + bd

- Step 6: Return result.

2. Karatsuba Multiplication Algorithm

- Step 1: If either number is a single digit, return their product.
  if x < 10 or y < 10 return x * y

- Step 2: Find number of digits n in the larger number and take half m = n/2.
  n = max(length(x), length(y)); m = n/2

- Step 3: Split both numbers using 10^m.
  a = x / 10^m , c = x % 10^m
  b = y / 10^m , d = y % 10^m

- Step 4: Recursively compute three products.
  ab = karatsuba(a, b)
  cd = karatsuba(c, d)
  abcd = karatsuba((a + c), (b + d))

- Step 5: Combine results using Karatsuba formula.
  result = ab * 10^(2*m) + (abcd - ab - cd) * 10^m + cd

- Step 6: Return result.

➢ **EXAMPLE –**

1. 1234 x 5678 by Gauss Multiplication

Step 1: Numbers: x = 1234, y = 5678
Number of digits = 4 → m = 2
So split at 10^2 = 100.

a = 1234 / 100 = 12
b = 1234 % 100 = 34
c = 5678 / 100 = 56
d = 5678 % 100 = 78

Step 2: Compute recursively

ac = gauss(12, 56) = 672

ad = gauss(12, 78) = 936
bc = gauss(34, 56) = 1904
bd = gauss(34, 78) = 2652

Step 3: Combine using Gauss formula
result = ac * 10^(2*m) + (ad + bc) * 10^m + bd
= 672 * 10000 + (936 + 1904) * 100 + 2652
= 6720000 + 284000 + 2652
= 7006652

Final Result = 7006652

2. 3456 x 9876 by Karatsuba Multiplication

Step 1: Same split
a (3456/100) = 34, c (3456%100) = 56
b (9876/100) = 98, d (9876%100) = 76
Step 2: Compute three products
ab = karatsuba(34, 98) = 3332
cd = karatsuba(56, 76) = 4256
abcd = karatsuba(34+56, 98+76) = karatsuba(90, 174) = 15660
Step 3: Combine
result = ab * 10^(2*m) + (abcd - ab - cd) * 10^m + cd
= 3332 * 10000 + (15660 - 3332 - 4256) * 100 + 4256
= 33320000 + (8072) * 100 + 4256
= 33320000 + 807200 + 4256
= 34131456

- **CODE –**

```
#include <iostream>
#include <string>
#include <ctime>
using namespace std;

int recursive_calls = 0;
int multiplications = 0;
int additions = 0;
int digit_count = 0;

void reset_counters() {
    recursive_calls = 0;
    multiplications = 0;
    additions = 0;
    digit_count = 0;
}

long long pow10(int m) {
    long long p = 1;
    for (int i = 0; i < m; i++){
        p *= 10;
    }
```

```
    return p;
}

long long gauss(long long x, long long y) {
    recursive_calls++;
    if (x < 10 || y < 10) {
        multiplications++;
        return x * y;
    }
    int n = max(to_string(x).length(), to_string(y).length());
    int m = n / 2;
    digit_count += n;
    long long power = pow10(m);
    long long a = x / power;
    long long b = x % power;
    long long c = y / power;
    long long d = y % power;
    long long bd = gauss(b, d);
    long long bc = gauss(b, c);
    long long ad = gauss(a, d);
    long long ac = gauss(a, c);
    additions += 2;
    return ac * pow10(2 * m) + (bc + ad) * pow10(m) + bd;
}

long long karatsuba(long long x, long long y) {
    recursive_calls++;
    if (x < 10 || y < 10) {
        multiplications++;
        return x * y;
    }
    int n = max(to_string(x).length(), to_string(y).length());
    int m = n / 2;
    digit_count += n;
    long long power = pow10(m);
    long long a = x / power;
    long long b = x % power;
    long long c = y / power;
    long long d = y % power;
    long long bd = karatsuba(b, d);
    long long z1 = karatsuba(a + b, c + d);
    long long ac = karatsuba(a, c);
    additions += 2;
    return ac * pow10(2 * m) + (z1 - ac - bd) * pow10(m) + bd;
}

int main() {
    long long n1, n2;
    int choice;
    cout << "Enter first number : ";
    cin >> n1;
    cout << "Enter second number : ";
    cin >> n2;
    cout <<"Choose a method:"<<endl;
    cout << "1. Gauss Multiplication"<<endl;
```

```
    cout << "2. Karatsuba Multiplication"<<endl;
    cout << "Enter choice: ";
    cin >> choice;
    reset_counters();
    clock_t start = clock();
    long long result;

    switch (choice) {
        case 1:
            result = gauss(n1, n2);
            cout << "Gauss Multiplication : "<<result << endl;
            break;
        case 2:
            result = karatsuba(n1, n2);
            cout << "Karatsuba Multiplication : "<<result << endl;
            break;
        default:
            cout <<"Invalid option"<< endl;
            return 0;
    }

    clock_t stop = clock();
    double duration = double(stop-start) /CLOCKS_PER_SEC * 1000.0;
    cout<< "Recursive calls: "<<recursive_calls<< endl;
    cout<< "Multiplications: " <<multiplications<<endl;
    cout<< "Additions: " <<additions<<endl;
    cout<< "Digit count: " <<digit_count<< endl;
    cout<< "Execution time: " <<duration<< " ms" << endl;
    return 0;
}
```

- **OUTPUT –**

1. Gauss Multiplication –

```
Enter first number : 12345
Enter second number : 56789
Choose a method:
1. Gauss Multiplication
2. Karatsuba Multiplication
Enter choice: 1
Gauss Multiplication : 701060205
Recursive calls: 25
Multiplications: 19
Additions: 12
Digit count: 18
Execution time: 3 ms
```

2. Karatsuba Multiplication –

```
Enter first number : 12345
Enter second number : 56789
Choose a method:
1. Gauss Multiplication
2. Karatsuba Multiplication
Enter choice: 2
Karatsuba Multiplication : 701060205
Recursive calls: 25
Multiplications: 17
Additions: 16
Digit count: 21
Execution time: 2 ms
```

- **COMPARISON –**

Feature / Metric	Gauss Multiplication	Karatsuba Multiplication
Recursive Calls	Higher (4 recursive calls per split → ac, ad, bc, bd)	Lower (3 recursive calls per split → ac, bd, z1)
Multiplications per Step	4 multiplications (per recursion level)	3 multiplications (per recursion level)
Additions per Step	2 additions (for (bc + ad))	2 additions (for (a+b)(c+d) - ac - bd)
Time Complexity	~ $O(n^{\log_2 4}) = O(n^2)$	~ $O(n^{\log_2 3}) \approx O(n^{1.585})$ (faster asymptotically)

- **CONCLUSION –**

In this practical, we have observed that Gauss multiplication requires more recursive calls and multiplications, making it slower for large inputs. Karatsuba multiplication reduces the number of multiplications, achieving better efficiency for large numbers.