Karatsuba Algorithm (Fast Multiplication)

A Detailed Study and Implementation

Prepared By:
Muhammad Talal 55876

Course/Subject:
Analysis of Algorithms

Instructor:
Muhammad Usman Sharif

Github Link:

Abstract

This document presents a comprehensive study on the Karatsuba algorithm, a fast multiplication technique that reduces the time complexity of large number multiplication. By employing a divide-and-conquer approach, the algorithm breaks down large multiplication problems into smaller subproblems, making it significantly faster than the traditional multiplication method.

Table of Contents

1. Introduction

Efficient multiplication of large numbers is a fundamental problem in computer science, particularly in areas such as cryptography, numerical analysis, and arbitrary-precision arithmetic. The standard grade-school multiplication algorithm runs in quadratic time, which becomes a bottleneck for very large numbers. In 1960, Anatolii Karatsuba introduced an algorithm that significantly reduces the multiplication time using a divide-and-conquer approach.

---

2. Problem Statement

Given two large integers represented either as strings or native integers (where supported), the objective is to compute their product faster than the traditional O(n2)O(n^2)O(n2) time method.

---

3. Naive Multiplication Approach

The classical multiplication method multiplies each digit of the first number with every digit of the second number.

For two n-digit numbers, the time complexity is:

T(n)=O(n2)T(n) = O(n^2)T(n)=O(n2)

This method becomes inefficient when dealing with integers containing thousands or millions of digits.

---

4. Karatsuba Algorithm

Theoretical Foundation

The Karatsuba algorithm improves multiplication efficiency by reducing the number of necessary multiplications from four to three using a clever algebraic identity.

Given two numbers XXX and YYY, each with nnn digits, split each into two halves:

X=a·10m+bX = a \cdot 10^m + bX=a·10m+b Y=c·10m+dY = c \cdot 10^m + dY=c·10m+d

where m=⌊n/2⌋m = \lfloor n/2 \rfloorm=⌊n/2⌋, and:

XY=ac·102m+((a+b)(c+d)−ac−bd)·10m+bdXY = ac \cdot 10^{2m} + ((a + b)(c + d) - ac - bd) \cdot 10^m + bdXY=ac·102m+((a+b)(c+d)−ac−bd)·10m+bd

This reduces the problem to three multiplications:

- acacac
- bdbdbd
- (a+b)(c+d)(a + b)(c + d)(a+b)(c+d)

and several additions and subtractions, which are linear time operations.

Recursive Formula

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

This recurrence leads to a significant improvement over the classical approach.

---

5. Algorithm Design

The Karatsuba algorithm is structured as a recursive divide-and-conquer method:

1. **Base Case:** If either number has only one digit, multiply them directly.
2. **Divide:** Split each number into two halves: high and low.
3. **Recursive Computation:**
    - Compute z0=b·dz0 = b \cdot dz0=b·d
    - Compute z1=(a+b)(c+d)z1 = (a + b)(c + d)z1=(a+b)(c+d)
    - Compute z2=a·cz2 = a \cdot cz2=a·c
4. **Combine the Results:**

XY=z2·102m+(z1−z2−z0)·10m+z0XY = z2 \cdot 10^{2m} + (z1 - z2 - z0) \cdot 10^m + z0XY=z2·102m+(z1−z2−z0)·10m+z0

---

6. Pseudocode

```
function Karatsuba(x: integer, y: integer) -> integer:
    if x < 10 or y < 10:
        return x * y

    n = max(number_of_digits(x), number_of_digits(y))
    m = floor(n / 2)
    power = 10^m

    // Split the numbers into high and low parts
    high1 = x / power
    low1 = x % power
    high2 = y / power
    low2 = y % power

    // Recursively compute three products
    z0 = Karatsuba(low1, low2)
    z1 = Karatsuba(low1 + high1, low2 + high2)
    z2 = Karatsuba(high1, high2)

    // Combine the results using Karatsuba formula
    return z2 * 10^(2 * m) + (z1 - z2 - z0) * 10^m + z0
```

---

7. Code Implementation (C++)

```
#include <iostream>
#include <cmath>
using namespace std;

long long karatsuba(long long x, long long y) {
    // Base case
    if (x < 10 || y < 10)
        return x * y;

    // Calculate the number of digits
    int n = max((int)log10(x) + 1, (int)log10(y) + 1);
    int m = n / 2;
    long long power = pow(10, m);

    // Split the numbers
    long long high1 = x / power;
    long long low1 = x % power;
    long long high2 = y / power;
    long long low2 = y % power;

    // 3 recursive calls
    long long z0 = karatsuba(low1, low2);
    long long z1 = karatsuba(low1 + high1, low2 + high2);
    long long z2 = karatsuba(high1, high2);

    // Combine the results
    return z2 * power * power + (z1 - z2 - z0) * power + z0;
}

int main() {
    long long a = 1234;
    long long b = 5678;
```

```
    cout << "Product: " << karatsuba(a, b) << endl;
    return 0;
}
```

---

8. Time Complexity Analysis

The Karatsuba algorithm divides each input in half and performs three multiplications:

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

Using the Master Theorem, this solves to:

T(n)=O(nlog23)≈O(n1.585)T(n) = O(n^{\log_2 3}) \approx O(n^{1.585})T(n)=O(nlog23)≈O(n1.585)

This is a significant improvement over the classical O(n2)O(n^2)O(n2) algorithm.

Algorithm	Time Complexity
Classical Multiplication	O(n2)O(n^2)O(n2)
Karatsuba Algorithm	O(n1.585)O(n^{1.585})O(n1.585)

---

9. Space Complexity

- Each level of recursion uses constant space aside from the input and result.
- Maximum recursion depth is $\log n$, leading to:

$$\text{Space Complexity} = O(n)$$

This includes temporary space for intermediate results.

---

10. Advantages and Limitations

Advantages

- More efficient than naive multiplication for large inputs
- Simple and elegant recursive structure
- Used in many high-precision libraries

Limitations

- Overhead from recursion and splitting makes it slower for small inputs
- Not the most efficient for extremely large inputs (e.g., FFT or Schönhage–Strassen algorithms are faster)
- Recursive depth can lead to stack overflow in constrained environments

---

11. Applications

- Cryptographic computations involving large integers (e.g., RSA)
- Scientific and engineering software needing arbitrary precision
- Mathematical computation libraries (e.g., GMP, Python's long integers)
- Algorithmic number theory and computer algebra systems

---

12. Conclusion

The Karatsuba algorithm represents a critical milestone in fast arithmetic algorithms. It demonstrates the power of divide-and-conquer in reducing time complexity. Although it has been surpassed in efficiency by more advanced methods for very large integers, it remains a practical and widely used algorithm in computer science for medium to large integer multiplication.