Lab 2: Karatsuba Algorithm for Multiplication

Algorithms and Computation

Name: Robert Rice
Date: December 4, 2025
Student ID: 657588340
Language: Python

---

1. Problem Statement

The goal of this lab is to multiply two non-negative integers that are given as strings. The inputs `num1` and `num2` can each be up to 200 digits long. I must return the product as a string.

Important constraints:

- I cannot use any built-in big integer library.
- I cannot convert the whole string into an `int` and just multiply.
- I must treat the strings like real big integers and do the arithmetic by hand.
- For large inputs, I must use the Karatsuba multiplication algorithm.

So, the main task is to build a small "big integer" multiplication using only strings, plus addition and subtraction that also work on strings.

2. Algorithm Overview

2.1 Big Integer as Strings

Because I am not allowed to use built-in big integer types, I treat each number as:

$$\text{num} = d_0 d_1 d_2 \dots d_{n-1}$$

where each d_i is a digit character from '0' to '9'. All calculations are done using:

- Character to digit conversion: `ord(c) - ord('0')`.
- Manual carrying and borrowing.
- Padding with zeros using string functions.

I wrote helper functions to:

- Remove the leading zeros from a string.
- Add two decimal strings.
- Subtract one decimal string from another (when $a \geq b$).

2.2 Grade-School Multiplication

For smaller inputs, I use the normal grade-school method:

- Multiply each digit of the first number by each digit of the second number.
- Keep an array of digits for the result.
- The handle carries us as we go from right to left.

This runs in $O(n^2)$ time for n-digit numbers, but is simple and has low overhead.

2.3 Karatsuba Multiplication

For larger inputs, I use the Karatsuba algorithm. The main idea is to divide each n-digit number into two halves:

$$x = a \cdot 10^m + b, \quad y = c \cdot 10^m + d$$

where a and c are the high parts and b and d are the low parts.

The product is:

$$xy = ac \cdot 10^{2m} + (ad + bc) \cdot 10^m + bd$$

Karatsuba reduces the work by computing:

$$\begin{aligned} z_0 &= ac, \\ z_1 &= bd, \\ z_2 &= (a + b)(c + d), \end{aligned}$$

and then using:

$$ad + bc = z_2 - z_0 - z_1.$$

This only needs three large multiplications instead of four. Each multiplication is performed recursively.

In my implementation:

- I pad both strings with leading zeros so that they have the same length.
- If the length is odd, I add one more leading zero to make it even.
- I divide each string into halves to get a, b, c, d.
- I call Karatsuba recursively in the parts.
- I shift by powers of 10 by appending zeros to the right.

3. Implementation Details

3.1 Choice of Language: Python

I went with Python for this lab because it is easier to focus on the algorithm instead of on extra syntax. Python has simple tools for working with strings (such as slicing, padding, and joining), which is important because all of the big integer work in this lab is done using strings. It also has low overhead for writing helper functions and recursive functions, which is helpful for the Karatsuba algorithm.

3.2 Main Methods Used

To build big integer multiplication using strings, I organized the code into small, focused methods:

- `strip_leading_zeros(s)`
- `add_strings(a, b)`
- `compare_strings(a, b)`
- `subtract_strings(a, b)`
- `grade_school_multiply(x, y)`
- `multiply(num1, num2)`

4. Time Complexity

Let n be the number of digits in the larger input.

- The grade-school method does $O(n^2)$ single-digit multiplications.
- Karatsuba satisfies the recurrence

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

  which solves to

$$T(n) = O\left(n^{\log_2 3}\right) \approx O\left(n^{1.585}\right).$$

For the LeetCode constraint ($n \leq 200$), the inputs are not large enough to really show a big speedup, and the string splitting and padding add overhead. But this lab is mainly about understanding the algorithm and how large integer libraries can work.

5. Testing and Examples

I tested the code with several cases:

- Small examples:
  - `"2"` × `"3"` → `"6"`
  - `"123"` × `"456"` → `"56088"`
- Zeros:
  - `"0"` × `"9999"` → `"0"`
- Larger random numbers:
  - I compared the result of `multiplying` with Python's built-in `int` for many random test cases (only for checking).

The answers matched in all tests.

6. What Is New Compared to Earlier Labs

In earlier labs, I often worked with normal `int` or `long` types. The new ideas in this lab are:

- Treating big integers as strings and doing all arithmetic by hand.
- Implement string-based addition and subtraction, including padding and removing leading zeros.
- Using a divide-and-conquer algorithm (Karatsuba) for multiplication.
- Combining two algorithms: using grade-school for small sizes and Karatsuba for larger sizes.

This is closer to a real big integer library than just calling a built-in type.

7. Reflections

This lab helped me understand large integers as more than just a type in the language. By forcing all calculations to happen on strings, I had to think carefully about:

- How do carrying and borrowing really work.
- How to split the numbers into high and low parts.
- How algorithm design can reduce time complexity.