Weekly Challenge 06 (Group 9): Divide and Conquer Multiplication

CS/CE 412/471 Algorithms: Design and Analysis

Breeha Qasim 08283, Taha Jawed Munshi 08486

Spring 2025

Total points: 25

Objective

In this WC, you will work in groups and:

- understand and modify a divide-and-conquer algorithm to multiply 2 n-digit numbers efficiently
- demonstrate step-by-step working with examples and hand-drawn illustrations.
- derive recurrence relations from the given algorithm
- solve recurrence relations using methods like substitution, recursion trees, and/or the Master Theorem,

Motivation

Traditional methods of multiplication like the one we learned in grade 3, become inefficient for large numbers, making divide-and-conquer approaches more effective. This challenge enhances understanding of such techniques by understanding, modifying, and analyzing the algorithm Multiply-Binary, helping students revisit recursion, recurrence relations, and time complexity calculations.

1 Multiply-Binary

Consider the following divide-and-conquer algorithm which multiplies 2 n-bit **binary** numbers using divide-and-conquer:

Algorithm: MULTIPLY-BINARY(x, y)
Input: 2 positive binary integers x and y
Output: Their product $x * y$

1. $n = \max(\text{size of } x, \text{size of } y)$
2. **if** $n = 1$ **then return** $x * y$
3. $m = \lfloor n/2 \rfloor$
4. $x_L = \lfloor x/2^m \rfloor, \quad x_R = x \mod 2^m$
5. $y_L = \lfloor y/2^m \rfloor, \quad y_R = y \mod 2^m$
6. $P_1 =$ MULTIPLY-BINARY(x_L, y_L)
7. $P_2 =$ MULTIPLY-BINARY(x_R, y_R)
8. $P_3 =$ MULTIPLY-BINARY$(x_L + x_R, y_L + y_R)$
9. **return** $P_1 \cdot 2^n + (P_3 - P_1 - P_2) \cdot 2^m + P_2$

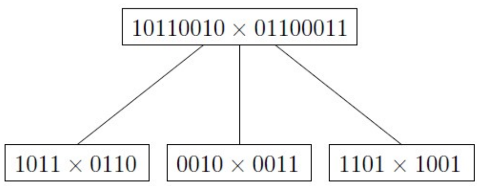


Figure 1: Instance of a Recursion Tree showing how a size-n multiplication problem is divided into size-n/2 sub-problems. How many sub-problems do you see?

Solution:

There are three sub-problems in the given Recursion Tree. Since the root node 10110010 × 01100011 divides into three sub-problems P1 (1011 × 0110), P2 (0010 × 0011) and P3 (1101 × 1001).

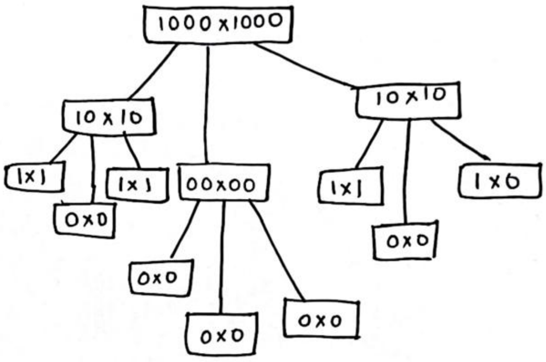
1.1 Working Example (5 points)

Demonstrate a working example of Procedure Multiply-Binary using any 2 4-bit numbers. Include figure of your hand-written working.

Solution:

we take $x = 1000$ and $y = 1000$

1) $n = 4$
2) $m = \lfloor 4/2 \rfloor = 2$
3) $x_L = 10_2$, $x_R = 00_2$
4) $y_L = 10_2$, $y_L = 00_2$
5) $P_1 =$ Multiply-Binary $(10, 10)$
    1) $n = 2$
    2) $m = 1$
    3) $x_L = 1_2$, $x_R = 0_2$
    4) $y_L = 1_2$, $y_R = 0_2$
    5) $P_1 = 1_2$
    6) $P_2 = 0_2$
    7) $P_3 = 1_2$
    8) $1 \cdot 2^2 + (1_2 - 1_2 - 0_2) \cdot 2 + 0_2$
       $= 100_2$
6) $P_2 =$ MULTIPLY-BINARY$(00, 00)$
    1) $n = 2$
    2) $m = 1$
    3) $x_L = 0$, $x_R = 0$
    4) $y_L = 0$, $y_R = 0$
    5) $P_1 = 0_2$
    6) $P_2 = 0_2$
    7) $P_3 = 0_2$
    8) $0 \cdot 2^2 + (0 - 0 - 0) . 2 + 0_2$
       $= 0_2$
7) $P_3 =$ MULTIPLY-BINARY$(10, 10)$
    1) $n = 2$
    2) $m = 1$
    3) $x_L = 1$, $x_R = 0$
    4) $y_L = 1$, $y_R = 0$
    5) $P_1 = 1_2$
    6) $P_2 = 0_2$
    7) $P_3 = 1_2$
    8) $1 \cdot 2^2 + (1 - 1 - 0) \cdot 2 + 0_2$
       $= 100_2$
8) $100_2 \cdot 2^4 + (100_2 - 100_2 - 0_2) \cdot 2^2 + 0_2$
   $= 1000000_2$.



2 Multiply-Decimal

We need to modify the algorithm Multiply-Binary considering if the inputs and outputs are now decimal numbers.

2.1 Procedure Multiply-Decimal (5 points)

Write the pseudo-code for Procedure Multiply-Decimal in CLRS notation. No credit if you use any programming language constructs or methods e.g. list.append(), etc.

Solution:

Algorithm: $\textsc{Multiply-Decimal}(x, y)$
Input: 2 positive decimal integers x and y
Output: Their product $x * y$

$n = \max(\text{size of } x, \text{size of } y)$
if $n == 1$ **then**
    return $x * y$
$m = \lfloor n/2 \rfloor$
$x_L = \lfloor x/10^m \rfloor$
$x_R = x \mod 10^m$
$y_L = \lfloor y/10^m \rfloor$
$y_R = y \mod 10^m$
$P_1 = \text{Multiply-Decimal}(x_L, y_L)$
$P_2 = \text{Multiply-Decimal}(x_R, y_R)$
$P_3 = \text{Multiply-Decimal}(x_L + x_R, y_L + y_R)$
return $P_1 \cdot 10^n + (P_3 - P_1 - P_2) \cdot 10^m + P_2$

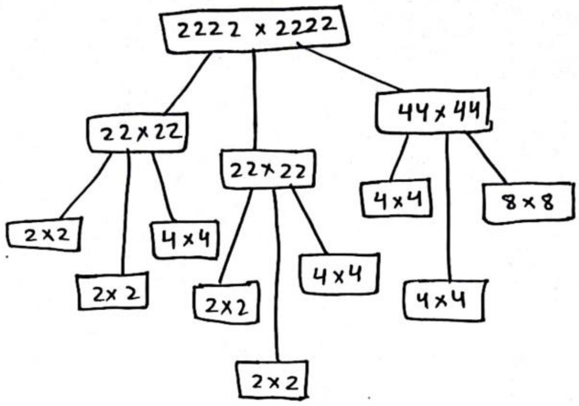
2.2 Working Example (5 points)

Demonstrate a working example of Procedure Multiply-Decimal using any 2 4-digit numbers. Include figure of your hand-written working.

Solution:

We take $x = 2222$ and $y = 2222$

1) $n = 4$
2) $m = \lfloor 4/2 \rfloor = 2$
3) $x_L = 22$, $x_R = 22$
4) $y_L = 22$, $y_R = 22$
5) $P_1 =$ MULTIPLY-DECIMAL(22,22)
    1) $n = 2$
    2) $m = 1$
    3) $x_L = 2$, $x_R = 2$
    4) $y_L = 2$, $y_R = 2$
    5) $P_1 = 2 \times 2 = 4$
    6) $P_2 = 2 \times 2 = 4$
    7) $P_3 = 4 \times 4 = 16$
    8) $4 \cdot 10^2 + (16-4-4) \cdot 10 + 4$
       $\Rightarrow 484.$
6) $P_2 =$ MULTIPLY-DECIMAL(~~44~~ 22, 22)
    1) $n = 2$
    2) $m = 1$
    3) $x_L = 2$, $x_R = 2$
    4) $y_L = 2$, $y_R = 2$
    5) $P_1 = 4$
    6) $P_2 = 4$
    7) $P_3 = 16$
    8) 484
7) $P_3 =$ MULTIPLY-DECIMAL(44,44)
    1) $n = 2$
    2) $m = 1$
    3) $x_L = 4$, $x_R = 4$
    4) $y_L = 4$, $y_R = 4$
    5) $P_1 = 16$
    6) $P_2 = 16$
    7) $P_3 = 64$
    8) $16 \cdot 10^2 + (64-16-16) \cdot 10 + 16$
       $= 1936$
8) $484 \cdot 10^4 + (1936-484-484) \cdot 10^2 + 484$
   $\Rightarrow 4937{,}284$



2.3 Recurrence (5 points)

Devise a recurrence for the designed algorithm Procedure Multiply-Decimal.

Solution:

The running time of the procedure can be expressed as,

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1, \\ 3T\left(\frac{n}{2}\right) + \Theta(n) & \text{otherwise.} \end{cases}$$

2.4 Time Complexity (5 points)

Solve the recurrence using the master method to find the Time Complexity of the algorithm.

Solution:

The given recurrence is in the form:

$$T(n) = aT(n/b) + f(n)$$

We'll use Master Theorem,

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

Here, $a = 3$, $b = 2$, and $f(n) = \Theta(n)$. We'll plug our values in watershed function $n^{\log_b a}$,

$$n^{\log_b a} = n^{\log_2 3} = n^{1.585}$$

Now we'll compare $f(n) = \Theta(n)$ with $n^{1.585}$. Since $f(n) = \Theta(n) = O(n^{log_2 3-\epsilon})$ for some $\epsilon = 0.585$, it follows that $f(n)$ grows asymptotically slower than $n^{\log_2 3}$, which aligns with property of **Master Theorem Case 1**.

Master Theorem Case 1

If there exists a constant $\epsilon > 0$ such that:

$$f(n) = O(n^{\log_b a-\epsilon}),$$

then:

$$T(n) = \Theta(n^{\log_b a}).$$

We then conclude that,

$$T(n) = \Theta(n^{\log_2(3)}) = \Theta(n^{1.585})$$

References

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Fourth Edition.*

Submission

1. Submit one pdf file including your solutions.
2. Clearly write your group number, member names and ids.
3. Where a working examples are required, you should include a hand-written working snapshot that demonstrates the step-wise working of the procedure.