Assignment 4

SFWRENG 2CO3: Data Structures and Algorithms–Winter 2024

Deadline: February 11, 2024

Department of Computing and Software
McMaster University

Please read the *Course Outline* for the general policies related to assignments.

Plagiarism is a *serious academic offense* and will be handled accordingly.
All suspicions will be reported to the *Office of Academic Integrity*
(in accordance with the Academic Integrity Policy).

This assignment is an *individual* assignment: do not submit work of others. All parts of your submission *must* be your own work and be based on your own ideas and conclusions. Only *discuss or share* any parts of your submissions with your TA or instructor. You are *responsible for protecting* your work: you are strongly advised to password-protect and lock your electronic devices (e.g., laptop) and to not share your logins with partners or friends!

If you *submit* work, then you are certifying that you are aware of the *Plagiarism and Academic Dishonesty* policy of this course outlined in this section, that you are aware of the Academic Integrity Policy, and that you have completed the submitted work entirely yourself. Furthermore, by submitting work, you agree to automated and manual plagiarism checking of all submitted work.

Late submission policy. Late submissions will receive a late penalty of 20% on the score per day late (with a five hour grace period on the first day, e.g., to deal with technical issues) and submissions five days (or more) past the due date are not accepted. In case of technical issues while submitting, contact the instructor *before* the deadline.

Problem 1. Typically, we assume that basic operations on natural numbers (e.g., adding or multiplying two natural numbers together) are performed in *constant* time. In practice, this assumption is correct *whenever* we restrict ourselves to natural numbers with some maximum size (e.g., 64 bit natural numbers, for which basic operations are supported directly by modern processors). Applications such as cryptography often work with *huge* natural numbers, however (e.g., 4048 bit values, which can hold a maximum of $\approx 3.7 \cdot 10^{1218}$). Hence, for these applications we can no longer assume that operations on natural numbers are in *constant* time: these applications require the development of efficient algorithms even for basic operations on natural numbers.

Consider two n-digit natural numbers $A = a_1 \dots a_n$ and $B = b_1 \dots b_n$ written in base 10: the digits $a_1, \dots, a_n$ and $b_1, \dots, b_n$ each have a value in $0, \dots, 9$. For example, if $n = 4$, then we could have $A = 3456$ and $B = 9870$, in which case $a_1 = 3$, $a_2 = 4$, $a_3 = 5$, $a_4 = 6$ and $b_1 = 9$, $b_2 = 8$, $b_3 = 7$, $b_4 = 0$.

P1.1. Write an algorithm Add(A, B) that computes $A + B$ in $\Theta(n)$.

Argue why your algorithm is correct and why the runtime complexity is $\Theta(n)$.

P1.2. Consider the typical pen-and-paper multiplication algorithm outlined in the below figure to compute $A \times B$:

	3456	
×	9870	
	0	(compute $3456 \cdot 0 \cdot 10^0$)
	241920	(compute $3456 \cdot 7 \cdot 10^1$)
	2764800	(compute $3456 \cdot 8 \cdot 10^2$)
+	31104000	(compute $3456 \cdot 9 \cdot 10^3$)
	34110720	(compute $0 + 241920 + 2764800 + 31104000$).

What is the runtime complexity of this algorithm in terms of the number of digits in A and B?

P1.3. Let C be an n-digit number with $n = 2m$. Hence, $C = C_{\text{high}} \cdot 10^m + C_{\text{low}}$ with C_{high} the first m digits of C and C_{low} the remaining m digits of C. For example, if $n = 4$, $A = 3456$, and $B = 9870$ then $m = 2$ and

$$A = A_{\text{high}} \cdot 10^m + A_{\text{low}}, \qquad A_{\text{high}} = 34, \qquad A_{\text{low}} = 56;$$
$$B = B_{\text{high}} \cdot 10^m + B_{\text{low}}, \qquad B_{\text{high}} = 98, \qquad B_{\text{low}} = 70.$$

Using the breakdown of a number into their high-part and low-part, Professor Breaksdown notices the following:

$$\begin{aligned} A \times B &= (A_{\text{high}} \cdot 10^m + A_{\text{low}}) \times (B_{\text{high}} \cdot 10^m + B_{\text{low}}) \\ &= A_{\text{high}} \times B_{\text{high}} \cdot 10^{2m} + A_{\text{high}} \times B_{\text{low}} \cdot 10^m + A_{\text{low}} \times B_{\text{high}} \cdot 10^m + A_{\text{low}} \times B_{\text{low}} \\ &= A_{\text{high}} \times B_{\text{high}} \cdot 10^{2m} + \left(A_{\text{high}} \times B_{\text{low}} + A_{\text{low}} \times B_{\text{high}}\right) \cdot 10^m + A_{\text{low}} \times B_{\text{low}}. \end{aligned}$$

Hence, Professor Breaksdown produces the following recursive algorithm to compute $A \times B$:

Algorithm BreaksdownMultiply(A, B) **:**

Input: A and B have $n = 2m$ digits.

1: **if** $n = 1$ **then**
2: **return** $a_1 \times b_1$.
3: **else**
4: $hh :=$ BreaksdownMultiply$(A_{\text{high}}, B_{\text{high}})$.
5: $hl :=$ BreaksdownMultiply$(A_{\text{high}}, B_{\text{low}})$.
6: $lh :=$ BreaksdownMultiply$(A_{\text{low}}, B_{\text{high}})$.
7: $ll :=$ BreaksdownMultiply$(A_{\text{low}}, B_{\text{low}})$.
8: **return** $hh \cdot 10^{2m} + (hl + lh) \cdot 10^m + ll$.
9: **end if**

Result: return $A \times B$.

Prove that the algorithm BreaksdownMultiply is correct.

P1.4. Give a recurrence $T(n)$ for the runtime complexity of BreaksdownMultiply. Explain each term in your recurrence $T(n)$.

Draw a recurrence tree for $T(n)$ and use this recurrence tree to solve the recurrence $T(n)$ by proving that $T(n) = \Theta(f(n))$ for some function f of n.

What is the runtime complexity of BreaksdownMultiply? Do you expect BreaksdownMultiply to be faster than the earlier pen-and-paper multiplication algorithm?

Hint: Feel free to assume that n is a power-of-two. Feel free to assume that we can add two v-digit numbers in $\Theta(v)$ (e.g., using Add) and that we can multiply a v-digit number with 10^w in $\Theta(v+w)$.

P1.5. Professor Smartmaths does not like that BreaksdownMultiply performs *four* multiplications via recursion. With some puzzling, Professor Smartmaths makes the following observation:

$$(A_{\text{high}} + A_{\text{low}}) \times (B_{\text{high}} + B_{\text{low}}) = A_{\text{high}} \times B_{\text{high}} + A_{\text{high}} \times B_{\text{low}} + A_{\text{low}} \times B_{\text{high}} + A_{\text{low}} \times B_{\text{low}}.$$

Hence, by rearranging terms, Professor Smartmaths concludes

$$A_{\text{high}} \times B_{\text{low}} + A_{\text{low}} \times B_{\text{high}} = (A_{\text{high}} + A_{\text{low}}) \times (B_{\text{high}} + B_{\text{low}}) - A_{\text{high}} \times B_{\text{high}} - A_{\text{low}} \times B_{\text{low}}.$$

Based on the above, Professor Smartmaths concludes:

$$\begin{aligned} A \times B &= (A_{\text{high}} \cdot 10^m + A_{\text{low}}) \times (B_{\text{high}} \cdot 10^m + B_{\text{low}}) \\ &= A_{\text{high}} \times B_{\text{high}} \cdot 10^{2m} + A_{\text{high}} \times B_{\text{low}} \cdot 10^m + A_{\text{low}} \times B_{\text{high}} \cdot 10^m + A_{\text{low}} \times B_{\text{low}} \\ &= A_{\text{high}} \times B_{\text{high}} \cdot 10^{2m} + \left(A_{\text{high}} \times B_{\text{low}} + A_{\text{low}} \times B_{\text{high}}\right) \cdot 10^m + A_{\text{low}} \times B_{\text{low}} \\ &= A_{\text{high}} \times B_{\text{high}} \cdot 10^{2m} + \Big(\big((A_{\text{high}} + A_{\text{low}}) \times (B_{\text{high}} + B_{\text{low}})\big) - \\ &\qquad (A_{\text{high}} \times B_{\text{high}}) - (A_{\text{low}} \times B_{\text{low}})\Big) \cdot 10^m + A_{\text{low}} \times B_{\text{low}}. \end{aligned}$$

Note that in final rewritten equation for $A \times B$ above, we only require the computation of *three multiplication terms*: namely $A_{\text{high}} \times B_{\text{high}}$, $A_{\text{low}} \times B_{\text{low}}$, and $(A_{\text{high}} + A_{\text{low}}) \times (B_{\text{high}} + B_{\text{low}})$.

Use the observation of Professor Smartmaths to construct a recursive multiplication algorithm SmartmathsMultiply that *only* performs three recursive multiplications. Argue why your SmartmathsMultiply algorithm is correct.

P1.6. Give a recurrence $T(n)$ for the runtime complexity of your SmartmathsMultiply algorithm. Explain each term in your recurrence $T(n)$.

Solve the recurrence $T(n)$ by proving that $T(n) = \Theta(f(n))$ for some function f of n. You can use any method you feel comfortable with.

What is the runtime complexity of SmartmathsMultiply? Do you expect SmartmathsMultiply to be faster than the earlier multiplication algorithms?

Hint: Feel free to assume that we can add two v-digit numbers in $\Theta(v)$ (e.g., using Add), that we can subtract two v-digit numbers in $\Theta(v)$, and that we can multiply a v-digit number with 10^w in $\Theta(v + w)$.

Assignment Details

Write a report in which you solve each of the above problems. Your submission:

1. must start with your name, student number, and MacID;
2. must be a PDF file;
3. must have clearly labeled solutions to each of the stated problems;
4. must be clearly presented;
5. must *not* be hand-written: prepare your report in LaTeX or in a word processor such as Microsoft Word (that can print or export to PDF).

Submissions that do not follow the above requirements will get a grade of zero.

Grading

Each problem counts equally toward the final grade of this assignment.