Université Gustave Eiffel
M2 – Semester 1 – 2025/2026

Algorithms and Data Structures - Homework

CHAU Dang Minh

We answer here the questions 5, 6, 9, 10 and 13 from the exercise sheet. Answer to other questions are found in the source code of `homework.py`.

Question 5. Assume that the coefficients of A and B takes $O(1)$ bits in memory. Prove that, in the worst case, the coefficients of the result of the sequence of products use $\Theta(n)$ bits.

Solution. Let $C^{(n)}, n \geq 1$ be the result of the product of n matrices. By the hypothesis, the coefficients of $C^{(1)}$ use $O(1)$ bits. Denote by the constant b the largest number of bits used by the coefficients of $C^{(1)}$. We consider the worst case where all coefficients A and B are equal to $2^b - 1$, or $C^{(1)} = (2^b - 1)\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}$. Then $C^{(n)} = (2^b-1)^n \begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}^n$. Using induction, we can prove that $\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}^n = 2^{n-1}\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}$. Then $C^{(n)} = (2^b-1)^n 2^{n-1}\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}$. It remains to estimate the number of bits used by $(2^b-1)^n 2^{n-1}$. We have

$$2^{n-1} \leq (2^b-1)^n 2^{n-1} \leq 2^{bn+n-1} \leq 2^{(b+1)n} - 1.$$

Therefore, the number of bits used by $(2^b-1)^n 2^{n-1}$ is between n and $(b+1)n$, which is $\Theta(n)$. Thus, in the worst case, the coefficients of the result of the sequence of products use $\Theta(n)$ bits.

Question 6. The large integer addition of two numbers encoded with $O(n)$ bits takes $O(n)$ time and their multiplication takes $O(n \log n)$ time. What is the complexity of `naive`.

Solution. Multiplication of two 2×2 matrices involves 4 additions and 8 multiplications. From Question 5, the number of bits used by the coefficients of $C^{(n)}$ is $\Theta(n)$. Hence the time complexity of multiplying $C^{(n)}$ and either A or B is in

$$8O(n \log n) + 4O(n) = O(n \log n).$$

For u of length n, we need to perform $n-1$ multiplications. We have,

$$\sum_{i=1}^{n-1} i \log i \in O((n-1)^2 \log(n-1)) = O(n^2 \log n).$$

Thus, the time complexity of `naive` is $O(n^2 \log n)$.

Question 9. Under the same complexity hypothesis as in Question 6, what is the complexity of `divide_and_conquer`.

Solution. Let $T(n)$ be the time complexity of `divide_and_conquer` for a word of length n. We have the recurrence

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

where $f(n) \in O(n \log n)$ as in Question 6. Using the Master Theorem with $a = 2$, $b = 2$ and $f(n) \in O(n^{\log_b a} \log n)$, we have

$$T(n) \in O(n^{\log_b a} \log^2 n) = O(n \log^2 n).$$

Question 10. Draw the curves of an experimental comparison between `naive` and `divide_and_conquer`.

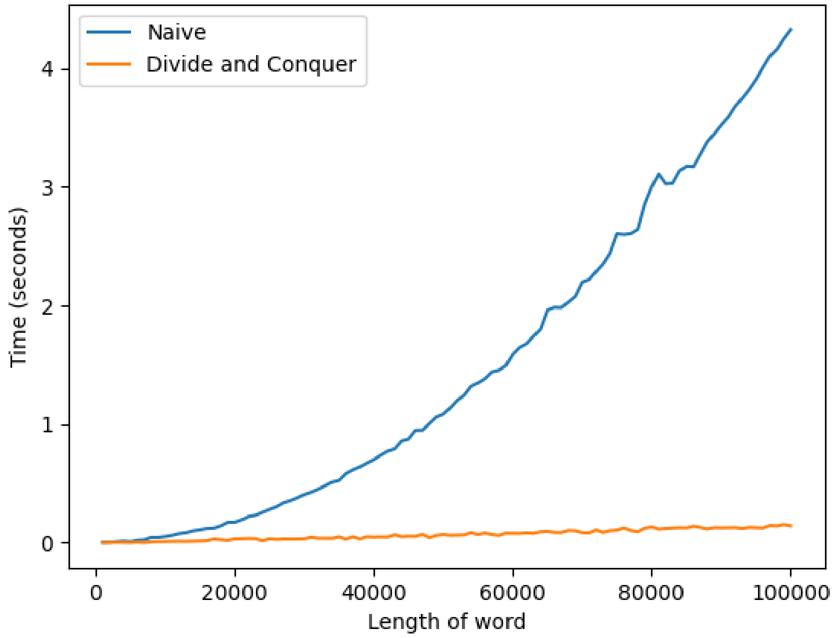

Figure 1: Comparison between `naive` and `divide_and_conquer`.

Question 12. Draw the curves of an experimental comparison between `divide_and_conquer` and `opt_divide_and_conquer`.

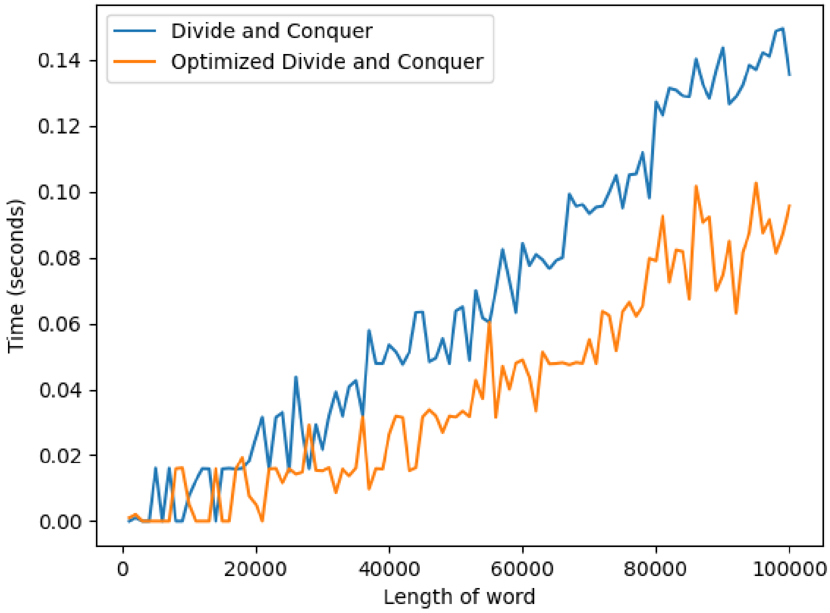

Figure 2: Comparison between `divide_and_conquer` and `opt_divide_and_conquer` (the cache is cleared before each run).

Question 13. Explain the result in Question 12?

Solution. It is seen from Figure 2 that `opt_divide_and_conquer` consistently outperforms `divide_and_conquer` across various lengths of u, especially with larger ones. This is because there can be the same subproblems occurring many times, producing a cache hit. The larger the length of u is, the greater chance an expensive subproblem is hit, which is then more likely to surpass the cost of cache management.