Algorithmique de base

Master 1, Université de Rennes

12/09/2024 – TD 1
13/09/2024 – TD 2

Exercise 1. Recall that to find a number x in a sorted array, the dichotomy principle consists of dividing the array in two parts, looking for x in one part and if it is not in this part, to search in the other part and so on.

1. Compute the computational cost of the binary search algorithm.
2. Write the pseudocode for a *linear search*, which scans through the array, looking for x. Using a loop invariant, prove that your algorithm is correct. Make sure that your loop invariant fulfills the three necessary properties.

Exercise 2.

1. Sort the array $L = [5, 2, 4, 6, 1, 3]$ using the *insertion sort* algorithm. How many comparisons and assignments have you made?
2. Propose a variation of insertion sort using binary search and apply it to array L.

   (You can make use of the function BinarySearch seen in class).
3. Compute the complexity of the new *binary insertion sort* algorithm.
4. Rewrite the insertion sort algorithm to sort into nonincreasing instead of nondecreasing order.

Exercise 3. Consider the problem of implementing a k-bit binary counter that counts upward from 0. We use an array $A[0, ..., k-1]$ of bits, where $\#A = k$, as the counter. A binary number x that is stored in the counter has its lowest-order bit in $A[0]$ and its highest-order bit in $A[k-1]$, so that $x = \sum_{i=0}^{k-1} A[i] \cdot 2^i$. Initially, $x = 0$, and thus $A[i] = 0$ for all $i = 1, \ldots, k-1$.

1. Write an example of an 8-bit binary counter as its value goes from 0 to 16 by a sequence of 16 increment operations.
2. Write a procedure *Increment(A)* that add 1 (modulo 2^k) to the value in the counter.

   Hint: Write a loop such that at the start of each iteration we wish to add a 1 into position i. If $A[i] = 1$, then adding 1 flips the bit to 0 in position i and yields a carry of 1, to be added into position $i + 1$ on the next iteration of the loop.
3. Compute the complexity of the algorithm. Make an analysis of the worst and average case scenario.

Exercise 4. Show that:

1. $\sum_{i=1}^{n} i^k = O(n^{k+1})$.
2. $\log(n!) = O(n \log n)$.
3. $\sum_{i=1}^{n} \frac{1}{i} = O(\log n)$.

Exercise 5. Consider the Fibonacci sequence:

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}.$$

Let $\phi = \frac{1+\sqrt{5}}{2}$ be the golden ratio and $\hat{\phi} = \frac{1-\sqrt{5}}{2}$ its conjugate.

1. Show that ϕ and $\hat{\phi}$ satisfy $x^2 - x - 1 = 0$.
2. Show that $F_i = \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}}$. In particular, observe that $F_{n+1} > \phi^{n-1}$ for $n \geq 2$.

Consider the Euclidean division algorithm for calculating the gcd of two positive integers.

3. Show that F_{n+1} and F_{n+2} are relatively prime and the Euclidean algorithm takes exactly n divisions to verify that $\gcd(F_{n+1}, F_{n+2}) = 1$.
4. Consider $\gcd(a, b)$, for two integers a, b, with $a > b > 0$. If there are n division in the Euclidean division algorithm, then $b \geq F_{n+1}$.

Exercise 6. Use the master theorem to give tight asymptotic bounds for the following recurrences.

1. $C(n) = 2C(n/4) + 1$.
2. $C(n) = 2C(n/4) + \sqrt{n}$.
3. $C(n) = 2C(n/4) + n$.
4. $C(n) = 2C(n/4) + n^2$.
5. $C(n) = 4C(n/2) + n^2 \log n$.

Exercise 7. Let u, v be integers such that $0 \leq u, v < 2^{2n}$. We want to compute the product of u and v.

1. Show that u, v can be written in binary notation using $2n$ bit.

Let $u = 2^n U_1 + U_0$, $v = 2^n V_1 + V_0$, with $V_0, U_0, V_1, U_1 < 2^n$.

2. Show that
$$uv = (2^{2n} + 2^n)U_1V_1 + 2^n(U_1 - U_0)(V_0 - V_1) + (2^n + 1)U_0V_0.$$

This is the basic principle of Karatsuba's algorithm which allows to compute the product of two large numbers u, v using 3 multiplications of smaller numbers, each with about half as many digits as u, v, plus some additions. (This basic step is, in fact, a generalization of a similar complex multiplication algorithm, where the imaginary unit i is replaced by a power of the base.) It is clear that these latter operations require a linear time in n. Let $C(n)$ be the time required to multiply two n-bit numbers by this method.

3. Then show that $C(2n) = 3C(n) + T(n)$, with $t(1) = 1$ and deduce an estimate complexity.

Exercise 8.

1. Write the pseudocode for Strassen's algorithm.
2. Use Strassen's algorithm to compute the matrix product
$$\begin{pmatrix} 1 & 3 \\ 7 & 5 \end{pmatrix} \cdot \begin{pmatrix} 6 & 8 \\ 4 & 2 \end{pmatrix}.$$
3. Modify Strassen's algorithm to multiply $n \times n$ matrices when n is not an exact power of 2. Show that the resulting algorithm runs in time $O(n^{\log_2 7})$.
4. Assume that we want to develop a matrix-multiplication algorithm that is asymptotically faster than Strassen's algorithm. This new algorithm will use the divide and conquer method, dividing each matrix into pieces of size $n/4 \times n/4$, and the divide and combine steps together will take $\Theta(n^2)$. If the algorithm creates m subproblems, then the recurrence for the running time $C(n) = mC(n/4) + \Theta(n^2)$. What is the largest integer value of m for which the new algorithm would be asymptotically faster than Strassen's algorithm?

Exercise 9. *Bubblesort* is a sorting algorithm that works by repeatedly swapping adjacent elements that are out of order.

Algorithm 1 Bubble sort

```
1: function BubbleSort(x_1, ..., x_n):
2:     for i ∈ {1, ..., n − 1} do:
3:         for j ∈ {n, ..., i + 1} do:
4:             if x_j < x_{j−1} then:
5:                 x_j ↔ x_{j−1}
```

1. Let A' denote the output of Bubblesort. To prove that Bubblesort is correct, we need to prove that it terminates and that
$$A'[1] \leq A'[2] \leq \cdots \leq A'[n], \tag{1}$$
where n is the length of A'. In order to show that Bubblesort actually sorts, what else do we need to prove?
2. State precisely a loop invariant for the for loop in lines 3–5, and prove that this loop invariant holds.
3. State a loop invariant for the for loop in lines 2–5 that will allow you to prove inequality (1).
4. What is the worst-case running time of Bubblesort? How does it compare to the running time of insertion sort?
5. Describe the algorithm to sort the list $[4, 5, 3, 1, 2]$.

Exercise 10. Consider the ring $R = \mathbb{Q}[x]/(x^2 - 1)$ and the ring homomorphism

$$\phi : R \to \mathbb{Q}[x]/(x-1) \times \mathbb{Q}[x]/(x+1), \quad f + (x^2 - 1) \mapsto (f + (x-1), f + (x+1)).$$

1. Show that ϕ is bijective and write its inverse.
2. Let $u = ax + b$, $v = cx + d \in R$. By using ϕ, show that we can multiply u and v in R using 2 multiplications in $\mathbb{Q}$.

Exercise 11. Consider the QuickSort algorithm.

Algorithm 2 Quick sort

---

function QuickSort$((x_1, ..., x_n), lo, hi)$: // Sorts the list between the indexes lo and hi included.
    i_{pivot} = Partition$\big((x_1, ..., x_n), lo, hi\big)$ // Guarantees that all elements from $(x_{lo}, ..., x_{i_{pivot}})$ are less or equal than $(x_{i_{pivot}+1}, ..., x_{hi})$.
    QuickSort$\big((x_1, ..., x_n), lo, i_{pivot}\big)$
    QuickSort$\big((x_1, ..., x_n), i_{pivot} + 1, hi\big)$

function Partition$((x_1, ..., x_n), lo, hi)$:
    $i \leftarrow lo - 1$
    $j \leftarrow hi + 1$
    $pivot \leftarrow x_{\lfloor \frac{lo+hi}{2} \rfloor}$ // Pivot is the middle element of the list to be sorted.
    while True **do** // Endless loop.
        repeat
            $i \leftarrow i + 1$
        until $x_i \geq pivot$
        repeat
            $j \leftarrow j - 1$
        until $x_j \leq pivot$
        if $i \geq j$ **then**
            return j // Returns the pivot index.
        $x_i \leftrightarrow x_j$

---

1. Describe the execution on the array $[4, 5, 3, 1, 2]$, with indexes $lo = 1$ and $hi = 5$.
2. Show that the quick sort algorithm above has complexity $O(n^2)$ in the worst case, and complexity $O(n \log n)$ in the best case.
