ICP Complexity Exercises – Master SAFE

Vanessa Vitse

2013-2014

Exercise 1. Give the computational complexity of the following piece of code:

```
for (int i=n; i>0; i/=2) {
      for (int j=1; j<n; j*=2) {
            for (int k=0; k<n; k+=2) {
                  ...  // constant number of operations
            }
      }
}
```

Exercise 2. Give the computational complexity of the following piece of code:

```
for (int i=1; i<n; i++) {
      for (int j=i; j<n; j++) {
            for (int k=0; k<j; k++) {
                  ...  // constant number of operations
            }
      }
}
```

Exercise 3. Give the computational complexity of the following piece of code:

```
for (int i=n; i>0; i--) {
      for (int j=1; j<n; j*=2) {
            for (int k=0; k<j; k++) {
                  ...  // constant number of operations
            }
      }
}
```

Exercise 4. Determine the expected running time of the following recursive procedure:

```
int F(int n) {
      if (n>0) {
            int i = random(n-1);
            printf("%d", i);
            F(i); F(n-1-i);
            }
      return 0;
}
```

where `random(int n)` returns a random integer value uniformly distributed in $[0, n]$, and the only instruction that takes a non-negligible time is the `printf` statement.

Exercise 5. You stand alone in front of an extremely long wall, similar to the Great Wall of China in one of its deserted areas. You want to cross, but you are not able to climb over the wall. You know that there exists a door somewhere, however you do not know if it is on your left or on your right, and you do not know how far it is from your position. Show that it is possible to find the door while walking $\Theta(n)$ kilometers in the worst case, where n is the (unknown) distance to the door in km.

Exercise 6. In computer programming, there are different ways to store data in a fixed order. *Arrays* have the advantage that retrieving or modifying the i-th entry takes constant time $O(1)$, but a given array can only store a fixed quantity of data corresponding to the amount of memory that it has been allocated. We want to be able to append elements at the end of an array (this is called a *dynamic* array).

- If the current number of elements is smaller than the allocated size, this can be done in constant time $O(1)$.
- Otherwise, we need to reallocate a larger amount of memory and copy the entries at this new position; this takes a time proportional to the number of entries.

Explain how to implement dynamic arrays so that the *amortized* cost of appending elements (i.e. the mean cost, when the number of operations goes to infinity) is in $O(1)$.

Exercise 7. We recall that if C is a class of decisional problems, then $co\text{-}C = \{L \subset A^* : \bar{L} \in C\}$.

1. Show that $co\text{-}P = P$.
2. Show that if $co\text{-}NP \neq NP$, then $P \neq NP$.

Exercise 8. In this exercise we consider a field K such that the addition and the multiplication of two elements of K take a constant time (e.g. K is a finite field).

1. Give the complexity of the usual algorithm for the addition and the multiplication of two matrices $A, B \in M_n(K)$.
2. Assume that $n = 2n'$ is even. We divide A, B and the product $C = AB$ in $n' \times n'$ sub matrices:
$$A = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix}, \quad B = \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix}, \quad C = \begin{pmatrix} C_1 & C_2 \\ C_3 & C_4 \end{pmatrix}$$
Now we define the following $n' \times n'$ matrices:
$$\begin{aligned} M_1 &= (A_1 + A_4)(B_1 + B_4) \\ M_2 &= (A_3 + A_4)B_1 \\ M_3 &= A_1(B_2 - B_4) \\ M_4 &= A_4(B_3 - B_1) \\ M_5 &= (A_1 + A_2)B_4 \\ M_6 &= (A_3 - A_1)(B_1 + B_2) \\ M_7 &= (A_2 - A_4)(B_3 + B_4) \end{aligned}$$

Show that

$$C_1 = M_1 + M_4 - M_5 + M_7, \quad C_2 = M_3 + M_5, \quad C_3 = M_2 + M_4, \quad C_4 = M_1 - M_2 + M_3 + M_6.$$

3. Deduce that the complexity of matrix multiplication is in $O(n^{\log_2(7)}) = O(n^{2.807})$ and in $\Omega(n^2)$.

Exercise 9. Let F be a computational problem such that the size of its output is in $O(\ln(n))$ (where n is the size of the input), and let D be the corresponding decision problem. Show that D is in P if and only if there exists an algorithm that solves F in polynomial time.

Exercise 10. Let $L \subset A^*$ be a language. We recall that the Kleene star of L is

$$L^* = \bigcup_{n \in \mathbb{N}} L^n = \{w_1 w_2 \dots w_k : k \in \mathbb{N}, w_i \in L \ \forall 1 \leqslant i \leqslant n\}.$$

We consider the following algorithm that uses an oracle for L:

```
Input  : A word w = a_1 ... a_n of length n
E ← {0};
for i = 1 to n do
    for j = 0 to i − 1 do
        if j ∈ E and a_{j+1} ... a_i ∈ L then
            E ← E ∪ {i}
if n ∈ E then
    return true
else
    return false
```

1. Show that this algorithm returns true if and only if w belongs to L^*.
2. Show that P is stable under the Kleene star, i.e. if $L \in P$ then $L^* \in P$.
3. Let L be a language such that L^* is in P. What can be said of L?