Léonard Brice, Mathieu Sassolas
leonard.brice@ulb.be, mathieu.sassolas@ulb.be

Session 4:
Complexity

Solutions

1 Complexity

1. *Prove that* $3n^3 - 2n^2 + n - 10 \in O(n^3)$.

   ☛ For every $n > 0$, we have :

   $$3n^3 - 2n^2 + n - 10 = \left(3 - \frac{2}{n} + \frac{1}{n^2} - \frac{10}{n^3}\right) n^3,$$

   and the mapping $n \mapsto \left(3 - \frac{2}{n} + \frac{1}{n^2} - \frac{10}{n^3}\right)_{n>0}$ is bounded (sequence with a finite limit). Let then $c > 0$ be such that $3 - \frac{2}{n} + \frac{1}{n^2} - \frac{10}{n^3} \leq c$ for every $n \geq 1$; then, for every $n \geq 1$, we have $3n^3 - 2n^2 + n - 10 \leq cn^3$ as desired.

2. *Sort by increasing asymptotic order the following functions over* $\mathbb{N}$*:*

   (a) $n \mapsto n^2 + 10$
   (b) $n \mapsto \log(n)$
   (c) $n \mapsto 2^n$
   (d) $n \mapsto \frac{n^2}{4}$
   (e) $n \mapsto n - 1$
   (f) $n \mapsto n\log(n)$
   (g) $n \mapsto n^n$
   (h) $n \mapsto n^3 - \frac{n^2}{4}$
   (i) $n \mapsto n^{\log(n)}$

   ☛ From the slowest to the fastest: 2b, 2e, 2f, 2d, 2a, 2h, 2i, 2c, 2g.

2 Algorithms

For each of the following problems, describe an algorithm that solves it and bound its execution time:

1. *Given a list ℓ of integers and a number k, is every element of ℓ smaller than k?*

   ☛ Initialize `result` = `True`. For each element e of ℓ, compare e and k: if $e \geq k$, then update `result` = `False`. Finally, return `result`.

   This algorithm requires one comparison to n for each element of ℓ. Therefore, it requires a number of operations that is linear in the size of ℓ and n.

2. *Given an undirected graph (as an adjacency matrix) and an initial vertex v, can every vertex be reached from v?*

   ☛ Mark v in blue, and every other vertex in red.

   For each blue vertex w, mark all of its red neighbours in blue. Then, mark w in green. Repeat that operation until there is no more blue vertex. There is an unreachable vertex if and only if in the end of this process, there is still a red vertex.

   This algorithm requires, for each vertex, to read the the list of its neighbours. It requires therefore a number of operations that is linear in the number of edges, or quadratic in the number of vertices.

3. *Given a Boolean formula φ, and a valuation ν, does ν satisfy φ?*

   ☛ Recursively:

   - if $\varphi = \top$, return `True`;
   - if $\varphi = \bot$, return `False`;
   - if $\varphi = x$ and $\nu(x) = 1$, return `True`;
   - if $\varphi = x$ and $\nu(x) = 0$, return `False`;
   - if $\varphi = \neg\psi$, then apply the same algorithm to ψ, and if the result is `False`, return `True`, and if it is `True`, return `False`;
   - $\varphi = \psi \wedge \chi$, then apply the same algorithm to both ψ and χ, and if both answers are `True`, then return `True`, otherwise return `False`.

   This algorithm is applied once to each subformula. It requires, therefore, a number of operations that is linear in the size of φ.

4. SAT*: Given a Boolean formula φ in CNF, does there exists a valuation ν that satisfies it?*

   ☛ Enumerate all the possible valuations ν, and for each of them, apply the previous algorithm. If φ contains m subformulae and n variables, then this algorithm requires $O(m2^n)$ operations.

5. *Given a number $p \in \{0, \ldots, 2^{64} - 1\}$, written in binary, is p a prime number?*

   ☛ Case by case:

   - If $p = 0$, return `False`.
   - If $p = 1$, return `False`.
   - If $p = 2$, return `True`.
   - ...
   - If $p = 2^{64} - 1$, return `False` (yes, you can check).

   This algorithm requires at most 2^{64} comparisons, i.e. a bounded time. It runs, therefore, in time $O(1)$.

6. *Given $p \in \mathbb{N}$, written in binary, is p a prime number?*

   ☛ Enumerate all the integers d from 2 to $\sqrt{p}$, and for each of them, test whether d divides p. If no such d does, then p is prime.

   This algorithm requires $O(\sqrt{p})$ divisibility tests, and each of them requires $O(\log_2(p))$ operations. It requires therefore $O(\sqrt{p}\log_2(p))$ operations.

3 Complexity classes

1. *Among the classes* P *and* EXP*, which one is included in the other?*

   ☛ P is included in EXP, because every mapping that is polynomially bounded is exponentially bounded.

2. *Among the algorithms you give as answers to Exercise 2, which ones are polynomial? Which of the problems presented in that exercise belong to* P*?*

   ☛ The algorithms given as answers to Questions 2.1, 2.2, 2.3, and 2.5 are polynomial, hence the corresponding problems belong to P. The problem given in Question 2.4 is not known to belong to P, but it is not proved that it does not (although it is very unlikely).

   The algorithm given for Question 2.6 is polynomial only if the number p is given in unary, but it is exponential if p is given in binary: indeed, in that case, the size of p (the number of bits required to write it) is not p, but $n = \log_2(p)$, and the algorithm runs therefore in time

$O(2^{\frac{n}{2}}n)$. Since the binary encoding is the canonical encoding for integers, this algorithm is not polynomial, but exponential (we sometimes call such an algorithm *pseudo-polynomial*).

However, there exist actual polynomial algorithms to solve the same problem: see Manindra Agrawal, Neeraj Kayal, and Nitin Saxena, *PRIMES is in* P, 2002, for the first that was published.

3. *Computer scientists usually consider, as a first approximation, that a problem is "easy" if it belongs to* P*, and "hard" if it does not. In what extent would you say it is true?*

   ☛ It is an approximation that makes sense: a problem that requires 2^n operations to be solved will in practice be impossible to solve on big instances, no matter how powerful the machine is. Indeed, the fastest computers in the world can currently make around 2^{60} operations per second. Therefore, on an instance of size $n = 118$, such a computer would solve that problem in 2^{58}s, which is the current approximation for the age of the universe. It is thus easy to see that even a computer that would be a thousand, a million or even a billion times more powerful could still solve that problem only on small instances.

   However, it is important to remember that if an algorithm is not polynomial, it only means that it will need a non-polynomial time *in the worst case*. In practice, some exponential algorithms are frequently used (SAT-solvers, for example) and are usually rather efficient on most of the instances they face; but, on *some* instances, they will require a very long time. Moreover, some problems for which no polynomial algorithms is known can be solved efficiently by a quantum computer, which does not execute operations sequentially. Quantum complexity is nowadays a very active research field, but it is beyond the scope of this course.

   On the other hand, it can be objected that an algorithm requiring n^{78} operations to be solved on an instance of size n is not very efficient. Such cases are rare in practice, but they justify the need of finer analyses: being able to say that a problem belongs to P is only a starting point to study its precise complexity.