Algorithms
Worksheet n° 1 : Complexity, Arithmetic operations

Exercice 1 : limits of progress...

Twenty-five years ago, a computer performed ten million operations per second and executed a sorting algorithm requiring $50 \times n \times \log_{10} n$ operations for an array of lenghts n. Let's compare it with a computer 100 times faster, but executing a (weaker) sorting algorithm requiring n^2 operations. What is the computing time for each on an input of size $n = 10^6$? And $n = 10^7$?
On the other hand, what is the maximum size of a table that can be processed in an hour for each of the two configurations ?

Exercice 2 : magnitudes

1. Show that $\frac{1}{2}n^4 + 2n + 3 \in \Theta(n^4)$.

2. Show that $5n^3 \notin \Theta(n^4)$.

3. Compare $n!$ to 2^n and to n^n.

Exercice 3 :

In the following, f and g designate functions with values in $\mathbb{N}^*$.

1. Show that $\forall f, g,\ \max(f, g) \in \Theta(f + g)$.

2. What about $\min(f, g)$?

Exercice 4 : complexity of loops

Express in the simplest possible way the number of instructions carried out by the following loops :

1.
```
for i in range(n) :
  for j in range(n) :
    for k in range(n) :
      # instruction in constant time
```

2.
```
for i in range(n) :
  for j in range(i) :
    # instruction in constant time
```

Now, let's consider algorithms taking an integer parameter `n`, and two functions of `n` noted f et g.

3. Let's suppose that the algorithm `A(n)` does $\Theta(f(n))$ loop turns, each one having a complexity $\Theta(g(n))$. What is the order of its complexity ?

Exercice 5 : ranking

Classify the following functions depending on their magnitude in the classes Θ_1 to Θ_7 respecting the following conditions :

- two functions appear in the same class Θ_i if and only if they have the same order :

$$\forall f, g, \quad f \in \Theta(g) \iff \exists i,\ f, g \in \Theta_i$$

- the classes Θ_i are arranged in increasing order of magnitude :

$$\forall i,\ \forall f, g, \quad f \in \Theta_i \text{ et } g \in \Theta_{i+1} \implies f \in O(g)$$

List of functions to be processed :

$$2n^2 + 3n, \quad n^2 + \frac{1}{8}n^3, \quad n^2 + \sqrt{n}, \quad n^2\sqrt{n}, \quad \sqrt{n}, \quad 2^n, \quad 4^n, \quad 2^{n+4}, \quad \log n, \quad \log(n^2)$$

Θ_1	Θ_2	Θ_3	Θ_4	Θ_5	Θ_6	Θ_7

Exercice 6 : multiplication of the russian peasant

Let's consider the following multiplication algorithm, said « *of the russian peasant* » :

```
def russian_multiplication (m, n):
  res = 0
  while n != 0 :
    if n%2 == 1 : res += m
    m *= 2
    n //= 2
  return res
```

1. Test this algorithm for the values `m = 13, n = 14`.

2. Show that this algorithm computes the multiplication of `m` by `n`.

3. Compute the number of additions, of multiplications by 2 and of divisions by 2 carried out by this algorithm in the worst case as a function of `n`.

4. In base 2, how do you perform the multiplication by 2 ? The division by 2 ? How do we test if a binary number is even or odd ?
Write the numbers 13 et 14 in base 2 and apply the algorithm of the above multiplication to these numbers.

5. Compare with the usual multiplication algorithm.

Exercice 7 : Prime numbers

1. Propose an algorithm `is_prime(p)` *as naïf as possible* allowing to determine if an integer `p` is prime. What is its complexity ? How can we improve it ?

Now, consider the following algorithm :

```
def eratosthene(n) :
  tab = [False, False] + [True]*(n-1) # tab = [False,False,True,True,...,True]
  for i in range(2, n+1) :
    if tab[i] :
      for k in range(2*i, n+1, i) : # from 2i, by steps of i
        tab[k] = False
  return tab
```

2. Run the algorithm for `n = 10`.

3. What does the table calculated by `eratosthene(n)` represent ? Justify.

4. Calculate an upper bound of the number of additions and of multiplications of integers carried out by the algorithm for an input `n`.

5. To witness the practical impact of complexity differences, you can compare on your machine the computing times of `[ p for p in range(10**6) if is_prime(p) ]` and of `[ p for p,b in enumerate(eratosthene(10**6)) if b ]`.