Lycée Loges

Chapitre 6

Informatique théorique TP

Simon Dauguet
simon.dauguet@gmail.com

8 janvier 2026

On va comparer ici le temps d'exécution de trois algorithmes permettant de calculer $(n!)^n$.

```
1  from time import time
2  def Complexite1(n:int) -> float :
3      t=time()
4      P=1
5      for i in range(1,n+1):
6          for j in range(1,n+1):
7              P=P*j
8      return(time()-t)
9
10 def Complexite2(n:int) -> float :
11     t=time()
12     P=1
13     for i in range(1,n+1):
14         P=P*i**n
15     return(time()-t)
16
17 def Complexite3(n:int) -> float :
18     t=time()
19     P=1
20     for i in range(1,n+1):
21         P=P*i
22     P=P**n
23     return(time()-t)
```

1. Estimer la complexité, a priori, de chacun de ces algorithmes. On ne prendra pas en compte l'appel de la fonction `time()`. On supposera, dans un premier temps naïvement, que l'élévation à la puissance n correspond à n multiplications.
2. Quel est le rôle de la fonction `time()` ?
3. Tester les trois algorithmes. Commenter.
4. Faire une fonction `Gompa() -> None` qui fait apparaître sur un même graphe les trois temps d'exécutions des trois algorithmes précédents pour tous les entiers entre 1 et 150.

   En fait, la fonction $x \mapsto x^n$ est codée de manière à minimiser le nombre de calculs (ce qui augmente considérablement son efficacité par rapport aux n multiplications naïve). Son code utilise le principe de l'exponentiation rapide et c'est son efficacité qui explique la différence notable entre les algorithmes. Il s'agit d'une fonction récursive, ce qui veut dire qu'elle s'appelle elle-même (la récursivité sera étudié dans le prochain chapitre).