

# Sur la complexité en temps de la factorielle

(complexite\_fac.pdf)

Jean-Paul Roy

10 juillet 2019

*Il s'agit de la rédaction d'une solution à l'exercice 11.29 du livre **Python. Apprentissage actif**, publié aux éditions Ellipses en 2019.*

On rappelle ci-dessous la définition itérative de la factorielle, mais pour la complexité – à l'existence de la pile près – nous pouvons raisonner sur la relation de récurrence `fac(n) == n * fac(n-1)`, les produits étant réalisés en séquence.

```
1 def fac(n) :                                # n entier >= 0
2     '''Retourne n! avec un calcul itératif.'''
3     res = 1
4     for i in range(2,n+1) :
5         res = res * i
6     return res
7
8 >>> fac(100000)                             # 100000!, mamma mia...
9 28242294...00000000                        # 456574 chiffres
```

a) En base 10, le nombre de chiffres d'une puissance de 10, disons  $n = 10^p$ , est égal à  $p + 1$ . Or  $p$  est par définition le logarithme en base 10 de  $n$ . Donc le nombre de chiffres est en  $\mathcal{O}(\log n)$  au moins sur les puissances de 10. Un peu trop rapide ?

Alors plus précisément, si  $k$  est le nombre de chiffres de  $n$ , encadrons  $n$  entre deux puissances de 10 consécutives :

$$10^{k-1} \leq n < 10^k$$

Les puissances nous gênent, on les élimine avec un logarithme (oui, en base 10) :

$$k - 1 \leq \log(n) < k$$

Mais cette inéquation signifie que  $k - 1$  est la partie entière  $\lfloor \log(n) \rfloor$  du nombre réel  $\log(n)$ . Donc  $k$  est en  $\mathcal{O}(\log n)$ . Le fait de travailler en base 10 n'est pas un cas particulier puisque  $\mathcal{O}(\log_a n) = \mathcal{O}(\log_b n)$ . En effet, tous les logarithmes sont proportionnels entre eux :  $\forall a, b \exists K \forall x \log_a(x) = K \log_b(x)$ .

**b)** Multiplier deux entiers  $a$  et  $b$  ( $a \geq b$  grands) chiffre à chiffre *comme à la main* revient à produire une matrice de chiffres ayant  $\log(b)$  lignes et  $\log(a) + \log(b) \leq 2 \log(a)$  colonnes, puis à additionner toutes ces colonnes de chiffres, ce qui a un coût total majoré par  $2 \log(a) \times \log(b)$ , donc en  $\mathcal{O}(\log(a) \log(b))$ .

$$\begin{array}{r}
 \begin{array}{rcccccc}
 & 3 & 4 & 2 & 1 & 5 & = a \\
 * & & & & 2 & 6 & 1 & = b \\
 \hline
 & & & & 3 & 4 & 2 & 1 & 5 \\
 & 2 & 0 & 5 & 2 & 9 & 0 \\
 & 6 & 8 & 4 & 3 & 0 \\
 \hline
 & 8 & 9 & 3 & 0 & 1 & 1 & 5
 \end{array}
 \end{array}$$

**c)** Solution geek : on se rend dans SAGE, sur <https://sagecell.sagemath.org> et on entre en Python un petit code symbolique pour calculer la limite convoitée :

```

10 ----- SAGECELL -----
11 k,n = var('k,n')                # au lieu de Symbol en sympy...
12 limit(sum(log(k),k,1,n)/(n*log(n)),n=infinity)
13 ----- Evaluate -----
14 1

```

Et zou ! Sinon, on retrouve les manches.

La quantité  $\log(n!)$  s'écrit sous la forme d'une somme :

$$\log(n!) = \log(2) + \dots + \log(n) = \sum_{k=2}^n \log(k)$$

et le lecteur ayant eu une introduction minimale au calcul intégral reconnaîtra une somme de Riemann, que l'on peut encadrer par deux intégrales en raisonnant géométriquement en termes d'aires de rectangles de largeur 1.

$$\int_1^n \log(x) dx \leq \sum_{k=2}^n \log(k) \leq \int_2^{n+1} \log(x) dx$$

Les deux intégrales sont faciles à calculer au lycée, puisqu'une primitive de  $\log(x)$  est  $x \log(x) - x$ . Ceci fait, il est tout aussi facile de vérifier que chacune d'elle est équivalente<sup>1</sup>

1. Deux fonctions  $f$  et  $g$  strictement positives sont *équivalentes* à l'infini si  $\frac{f}{g}$  tend vers 1.

à  $n \log(n)$ . Donc :

$$\lim_{n \rightarrow +\infty} \frac{\sum_{k=2}^n \log(k)}{n \log(n)} = 1$$

et ainsi  $\boxed{\log(n!)}$  est bien équivalent à  $\boxed{n \log(n)}$ .

*NB. Une autre solution utiliserait la formule de Stirling.*

**d)** Soit  $c_n$  le coût du calcul de **fac(n)** si l'on comptabilise le nombre d'opérations à 1 chiffre (en temps constant) effectuées. Alors  $c_n - c_{n-1}$  n'est autre que le coût de la multiplication de  $(n-1)!$  par  $n$ , qui est en  $\mathcal{O}(\log((n-1)!) \log(n))$  par b). Or, par c) :

$$\log((n-1)!) \sim \log(n!) \sim n \log(n)$$

Finalement,  $c_n - c_{n-1} \sim n \log^2(n)$

**e)** D'après la question précédente,  $c_n \sim c_{n-1} + n \log^2(n)$ . Si l'on déroule cette récurrence, il vient :

$$c_n \sim \sum_{k=2}^n k \log^2(k)$$

et un encadrement de cette somme de Riemann par deux intégrales comme en c) aboutirait au résultat final  $\boxed{c_n \sim n^2 \log^2(n)}$ .

*NB. Un raisonnement plus hardi pourrait exprimer qu'en passant du discret au continu, une somme devient une intégrale. Or :*

$$\int_2^n x * \log^2(x) dx = \frac{x^2}{4} (2 \log^2(x) - 2 \log(x) + 1) = \mathcal{O}(x^2 \log^2(x))$$

obtenu à la main, ou via SAGE en demandant  $\boxed{\text{integrate}(x*\log(x)**2,x)}$

(P)CQFD

(Presque) Ce Qu'il Fallait Démontrer...