Propriétés et caractéristiques d'un algorithme

Un bon algorithme doit posséder deux propriétés essentielles :

- ✗ Il doit **terminer** : il ne doit donc pas « tourner en rond » ou continuer indéfiniment ; il doit fournir le résultat en un nombre fini d'opérations.
- ✗ Il doit être **correct** : c'est-à-dire qu'il doit produire le résultat attendu dans toutes les situations.

Enfin il possède deux caractéristiques :

- ✗ Son **coût en temps** (complexité temporelle) : le nombre d'opérations nécessaires à son exécution (où la durée de son exécution).
- ✗ Son **coût en espace** (complexité spaciale) : la quantité d'espace mémoire nécessaire à son exécution. On s'intéresse uniquement au coût en temps.

Le coût en temps (complexité en temps)

En général, on ne calcule pas le coût exact d'un algorithme mais son ordre de grandeur. On classe alors les différents types de coûts par familles. Si on note n la taille des données d'entrée, le coût de l'algorithme est une fonction de n, noté O(f(n)).
Voici les principaux types de complexité ainsi que des ordres de grandeurs de temps d'exécution en fonction de n. Nous partirons du principe qu'une instruction élémentaire s'exécute en 1µs. Un temps astronomique est supérieur à un 10^{18} années.

Type de complexité	Notation	t pour n = 10	t pour n = 1000	t pour n = 10^5
Constante	O(1)	1µs	1µs	1µs
Logarithmique	O(log(n))	3µs	10µs	17µs
Linéaire	O(n)	10µs	1ms	0,1s
quadratique	$O(n^2)$	100µs	1s	2,8h
exponentielle	$O(2^n)$	1000µs	astronomique	astronomique
Factorielle	O(n!)	3s	astronomique	astronomique

Le tableau précédent récapitule les complexités de référence par ordre croissant :

- ✗ **1** ***complexité constante*** : temps constant pour accéder à une valeur d'une liste
- ✗ **$\log_2$(n)** ***complexité logarithmique*** : recherche dichotomique dans une liste triée
- ✗ **n** ***complexité linéaire*** : parcours d'une liste
- ✗ **n.$\log_2$(n)** complexité quasi-linéaire : tri fusion (terminale), le tri de python
- ✗ **n^2** ***complexité quadratique*** : double parcours de liste imbriqués (tri par insertion)
- ✗ **n^p** avec p > 2 ***complexité polynomiale*** : n parcours de liste imbriqués
- ✗ **2^n** complexité exponentielle : problème du sac à dos (algo naïf)
- ✗ **n!** complexité factorielle : problème du voyageur de commerce (algo naïf)

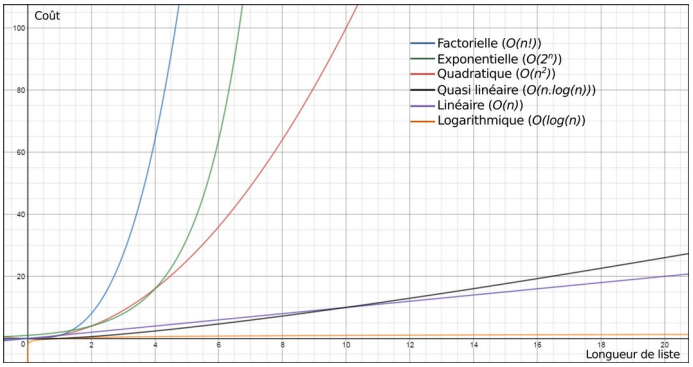

Figure 1: Allure des principales complexités