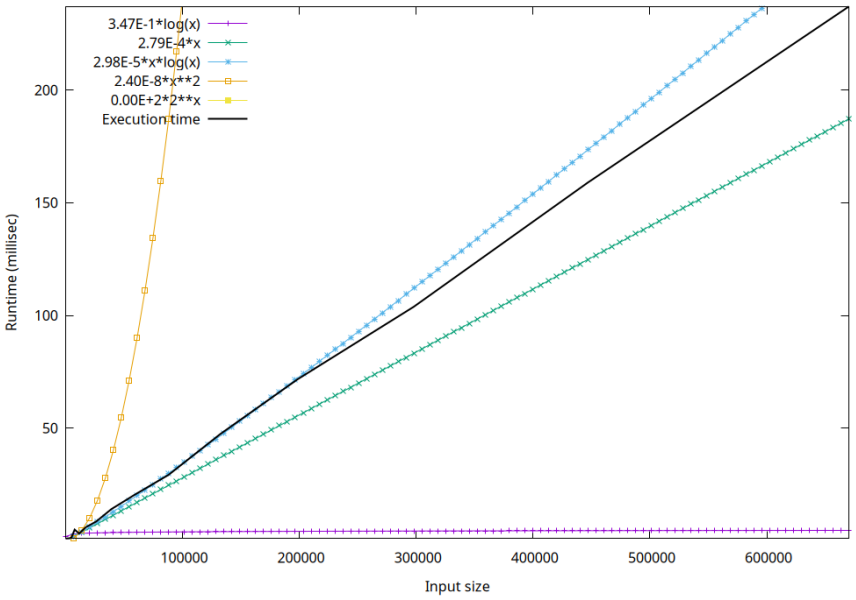
# Analyse d'Algorithmes - TP1

Documentation

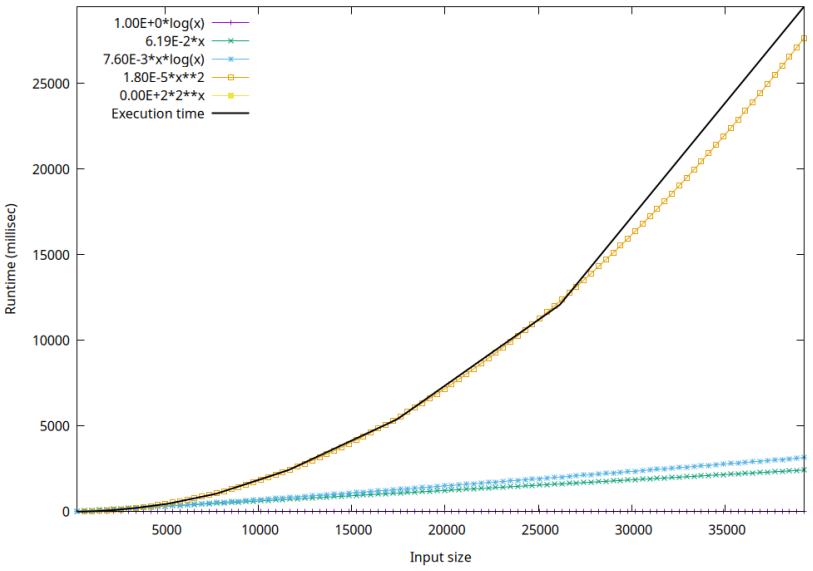
*Simon Pichenot*

## Problème 1

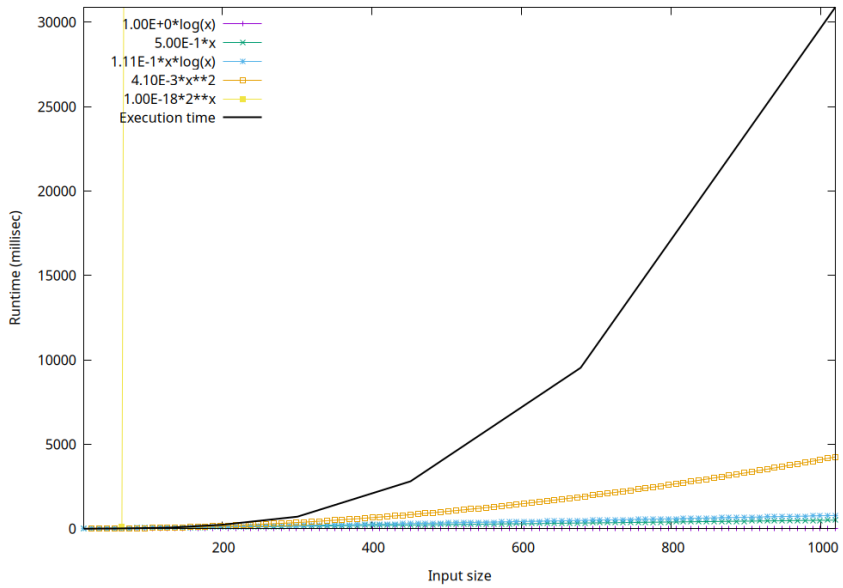
Pour le problème 1 je récupère les graphs suivants :



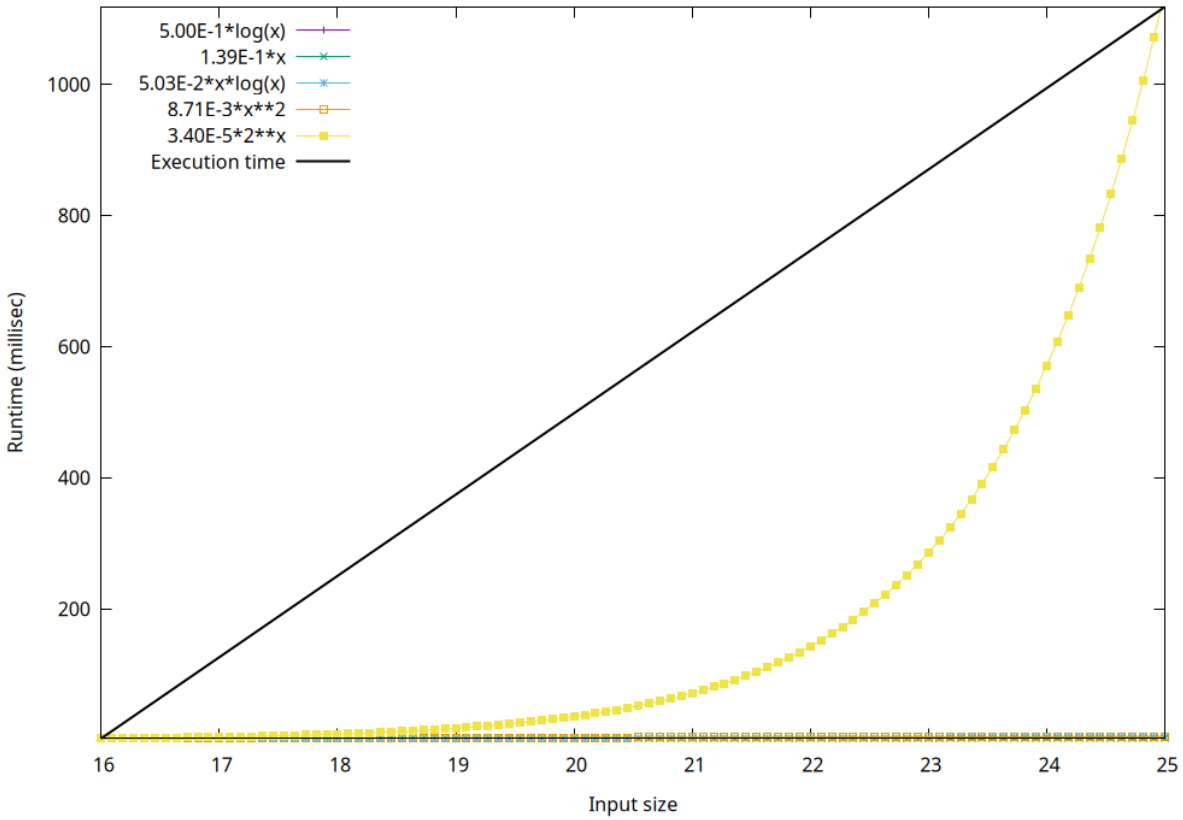

*Fonction en O(n)*



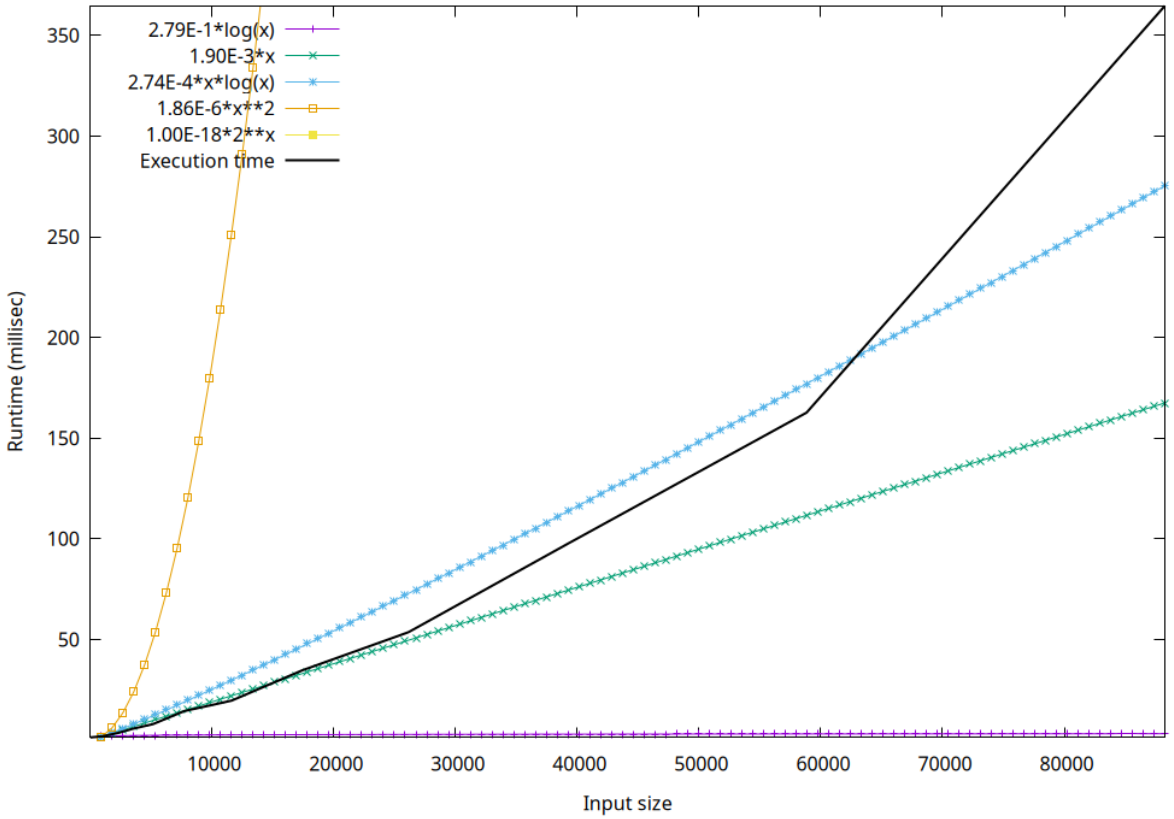

*Fonction en O(n²)*




*Fonction en O(n³)*




*Fonction en O(2^n)*

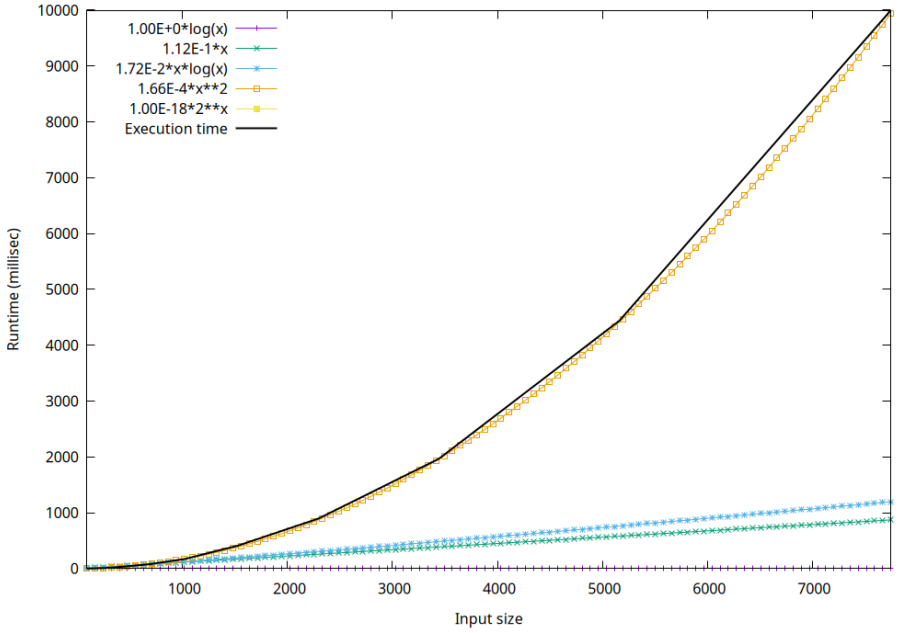



*Fonction en O(n log n)*

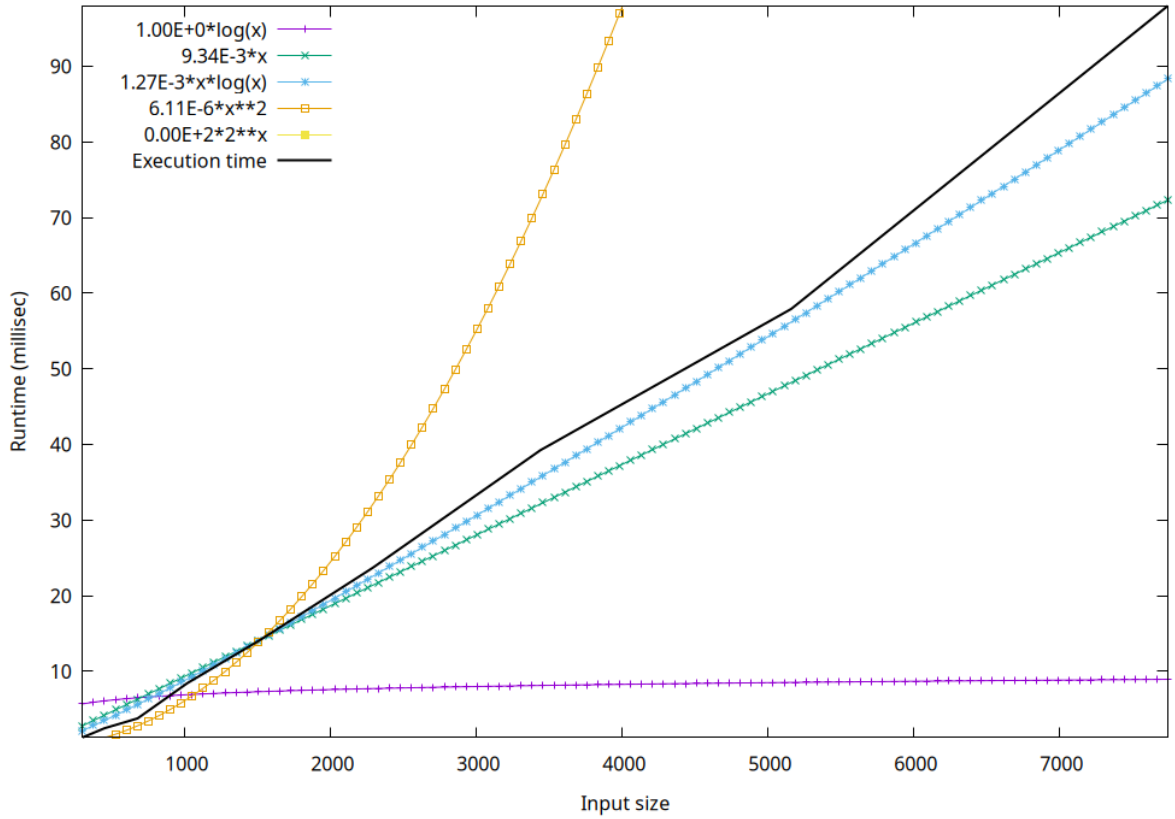
# Problème 2

Pour cette partie voir code probleme2.py

b) La complexité de l'algorithme est en O(n²). On a la courbe de test ci- dessous.



En utilisant la méthode diviser pour régner on arrive à diminuer le temps a O(n log n).
On a la courbe de test ci-dessous.




e) Sachant que mon processeur à une cadence de 2.9 Ghz et que la taille d'entrée est de $10^8$. Le code est en python donc on considère 100 microsecondes pour le temps de traitement d'une opération.
On a donc pour la méthode naïve :
$(10^8)^2$ = 10^16 => (10^16)/10^7 * 2.9 = 344827586 = 10.9 ans

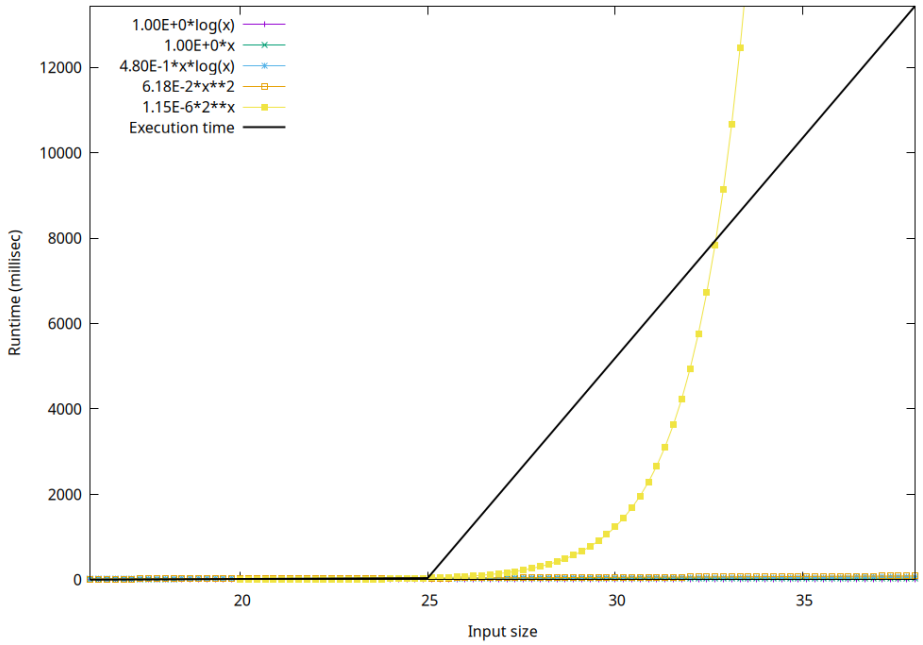
Et pour la méthode diviser pour régner :
($10^8$ * log($10^8$) = 8 * $10^8$ => (8 * $10^8$)/10^7 * 2.9 = 27s

# Problème 3

Pour le code de ce problème voir fichier probleme3.py

b) La taille de l'entrée est n et que l'algorithme itère sur n alors la complexité dépend de n.

c)




Nous avons peut de point car le temps d'exécution est très long mais nous pouvons voir que la complexité est proche de O(n²).

d) L'algorithme est long car dans la boucle nous faisons beaucoup de fois le même calcul pour vérifier nos valeurs du tableau.

f) Avec l'optimisation apportée à cet algorithme nous avons un temps linéaire O(n).

g)