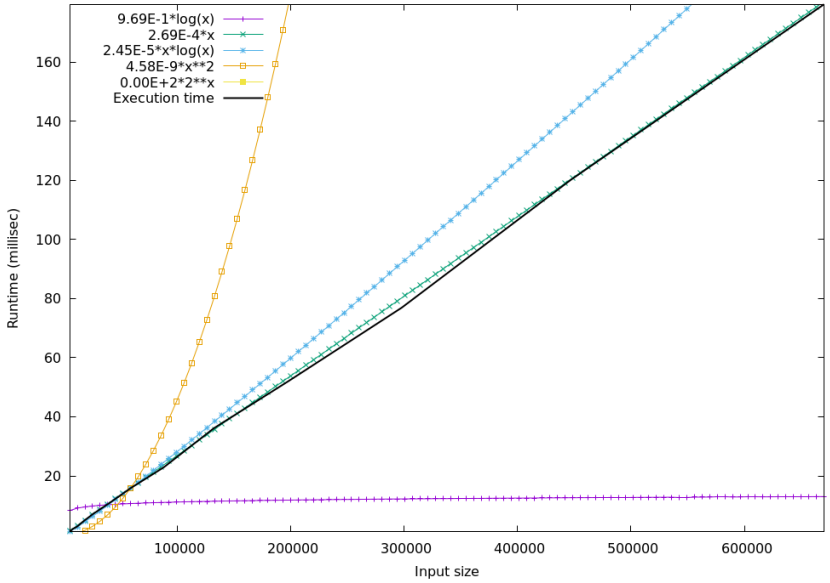
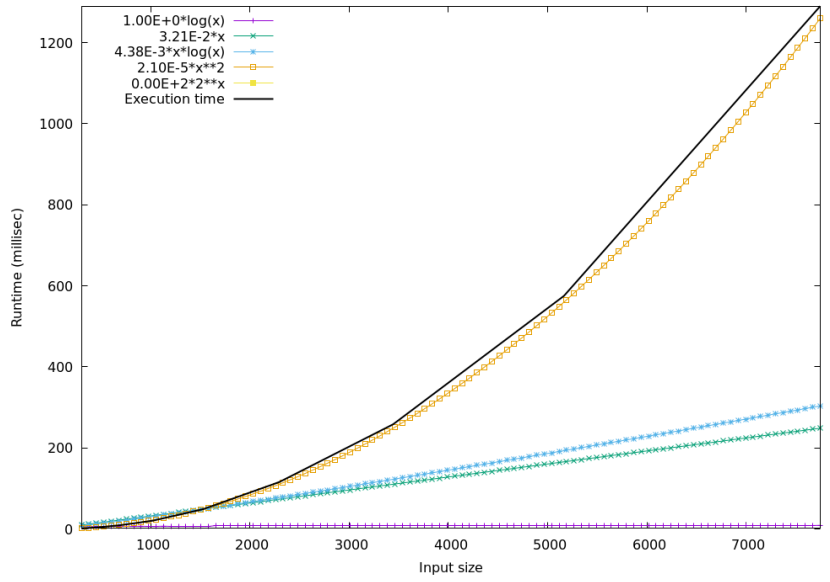
Hugo Rey - TP 1

Problème 1

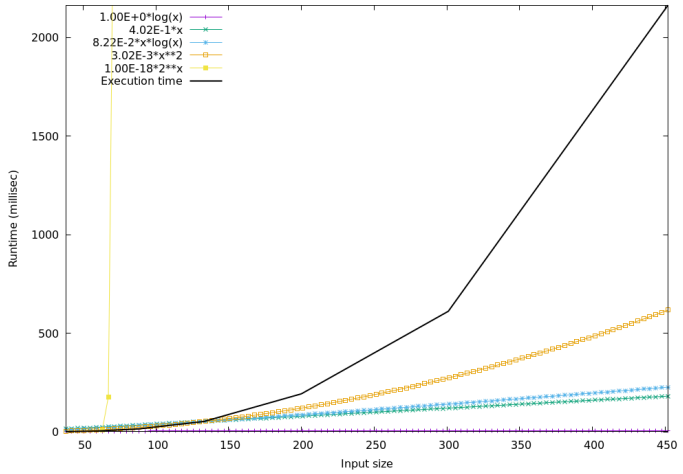
Pour chaque type de complexité j'ai changé les paramètres (pas et borne supérieure) afin que le temps d'exécution ne soit pas trop long.



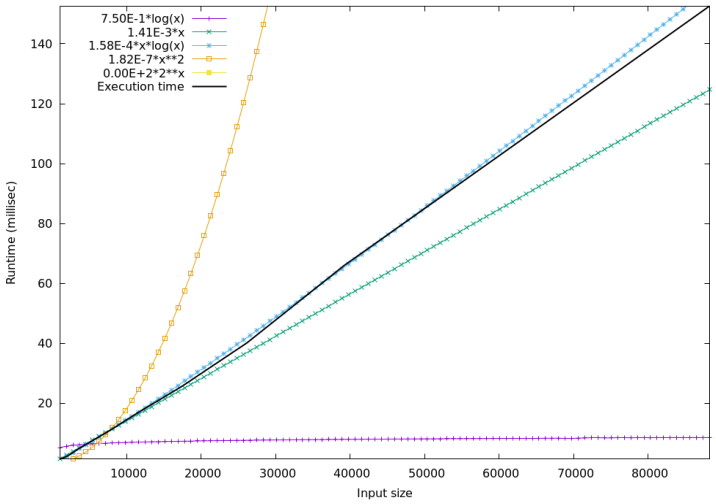
fctAlgo(), avec un temps d'exécution respectant la complexité O(n)



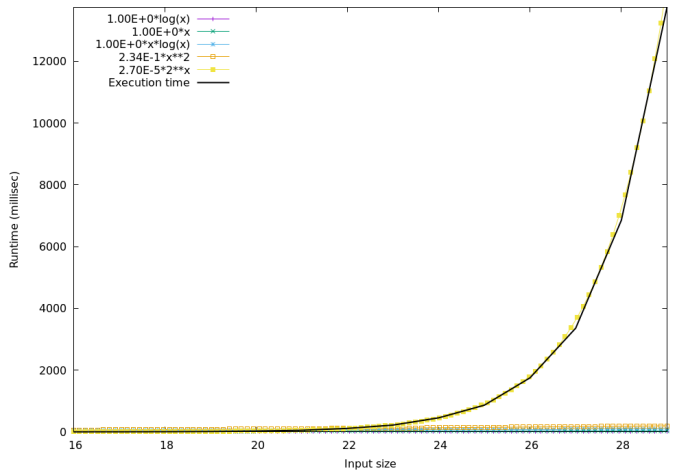
fctAlgo2(), avec un temps d'exécution respectant la complexité O(n^2)



fctAlgo3(), avec un temps d'exécution respectant la complexité O(n^3)



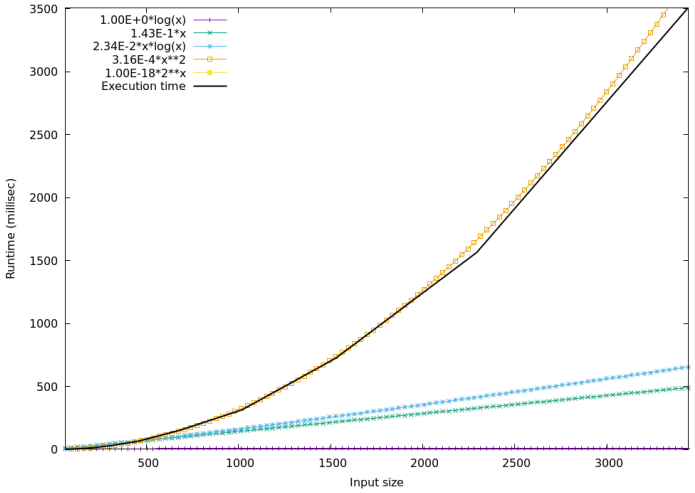
fctAlgo4(), avec un temps d'exécution respectant la complexité O(n log n)



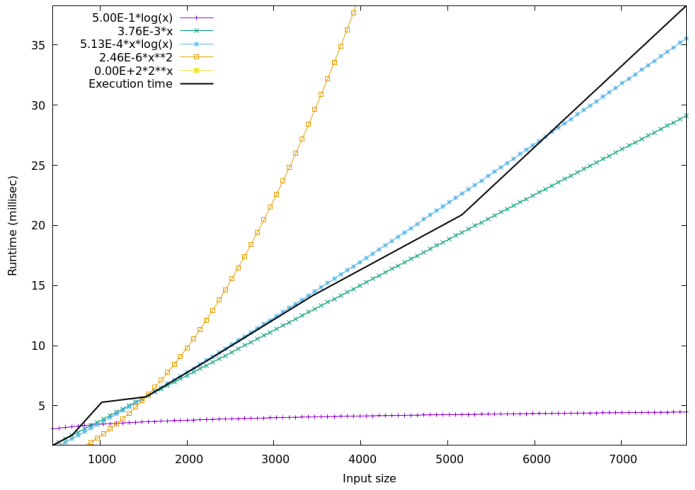
fctAlgo5(), avec un temps d'exécution respectant la complexité O(e^n)

Problème 2

a. Code fourni.
b. La complexité est de O(n²), ceci est dû à la double boucle. Si on vérifie avec graphChronoGenerator.py on obtient le graph suivant, ce qui le confirme:



c. Voir dmin.py
d. En testant nous obtenons le graph suivant :

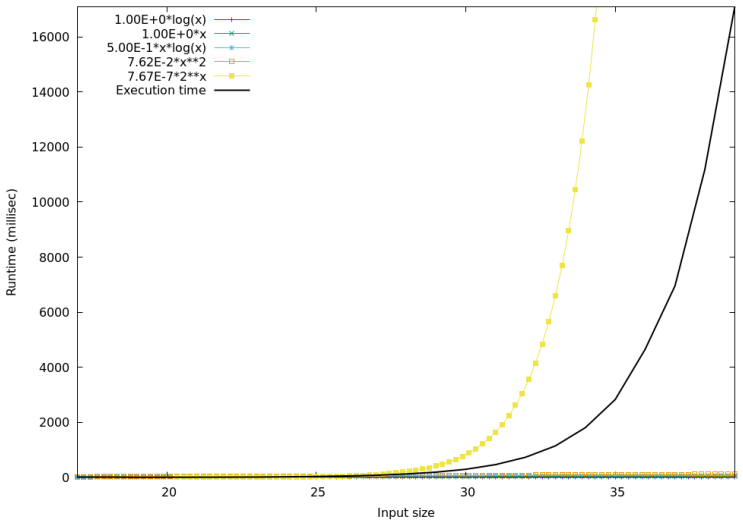


Le temps d'exécution suit bien la complexité attendue

e. Pour une taille d'entrée de 10^8, Pour cela il faut appliquer la complexité puis on divise par le nombre d'opérations par secondes de la machine. Ma machine à 2.70GHz donc 2.70 * 10^9. On enlève un facteur 10^2 à cause du python ce qui donne 2.70 * 10^7
    i. Pour la méthode naïve : ((10^8)^2)/(2.70 * 10^7) = 370370370.37 sec soit **11.7 ans environ**
    ii. Pour la méthode dpr : (10^8 * log(10^8))/(2.70 * 10^7) = **29.6296296296 sec**

Problème 3

a) voir code (fonction **sommeMin**)
b) La fonction a une entrée de taille n. Sa complexité n'est pas polynomiale.
c) Une fois testé on peut voir que le temps d'exécution est cohérent à la complexité calculée.



d) Dans le pire des cas, la fonction effectue des appels récursifs pour chaque valeur possible de x (1, 3 et 5) jusqu'à ce que i atteigne 0 (cas de base). Cela signifie que la profondeur maximale de la récursion est proportionnelle à i, car pour chaque valeur de x, elle peut réduire i de 1 dans le pire des cas. À chaque niveau de la récursion, la boucle for itère trois fois (pour 1, 3 et 5). Donc, la complexité totale de la fonction est exponentielle en fonction de i dans le pire des cas. **O(e^n)**
e) Voir code (fonction **sommeMinMemo**) la fonction est itérative car python donnais des erreurs "maximum recursion depth") pour les grands tableaux.
f) Puisque la mémoisation est en place, le calcul ne sera effectué que 1 fois par position de fin. Donc N fois. La complexité est donc de O(n)
g) On peut voir que le temps d'exécution est cohérent avec la complexité.