Thomas Rubini

I)

Github contenant le projet: https://github.com/ThomasRubini/ComplexiteAlgo (https://github.com/ThomasRubini/ComplexiteAlgo) (si j'y pense)

1)

Tri par sélection:

Dans le tri par sélection, la liste est séparée en 2 parties: la partie non triée (à gauche), et la partie triée (à droite, qui gagne 1 de taille à chaque itération)
A chaque itération, nous trouvons le nombre le plus petit de la liste. Puis, nous étendons la partie triée de 1 sur la gauche, et nous échangeons ce nouvel élément de la liste triée avec celui que nous venons de trouver (le plus petit de la liste non trié)

Tri fusion (Merge sort)

Le tri fusion est un algorithme récursif qui fonctionne sur la base de "Diviser pour mieux régner". Dans cet algorithme, nous divisons la liste en 2 parties égales, que nous divisons encore, jusqu'a obtenir des listes de taille 1 (qui sont donc triées). Ensuite, nous fusionnons les listes triées jusqu'a reformer la liste initiale (mais triée).

Tri à bulles

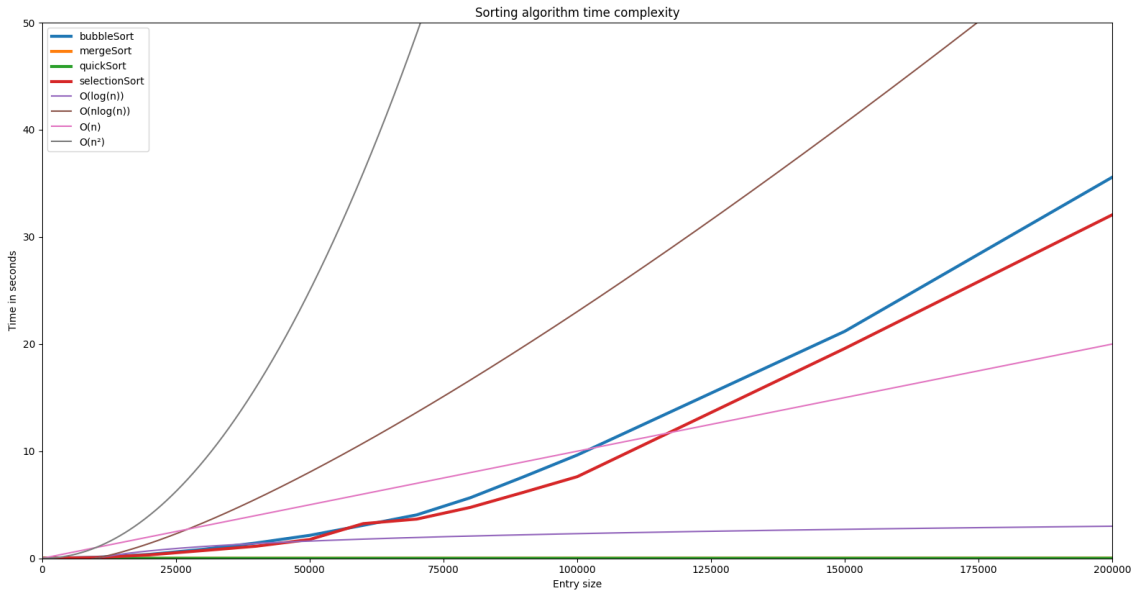
A chaque itération, cet algorithme va comparer les éléments côte-à-côte de la liste, et les inverser si le premier est plus grand que le second. Ainsi, on peut garantir qu'après chaque itération `i`, les `i` derniers éléments de la liste seront triés.
L'algorithme s'arrête quand il fini une itération sans avoir besoin d'inverser des éléments

Tri rapide (quicksort)

Le tri rapide est un algorithme récursif qui fonctionne sur la base de "Diviser pour mieux régner". Il sélectionne un pivot dans la liste, puis partitionne la liste de facon à ce que tout les éléments avant le pivot soient plus petits que lui, et que tout les éléments avant le pivot soient plus grands que lui. Cette opération est ensuite répétée sur les 2 sous-listes crées, jusqu'a obtenir des listes de taille 1.

7)

La première chose qui vient à l'esprit est de comparer les 4 algorithmes:
(script: `joli_dessin_selection_bubble.py` , données: `output_selection_bubble.csv`)

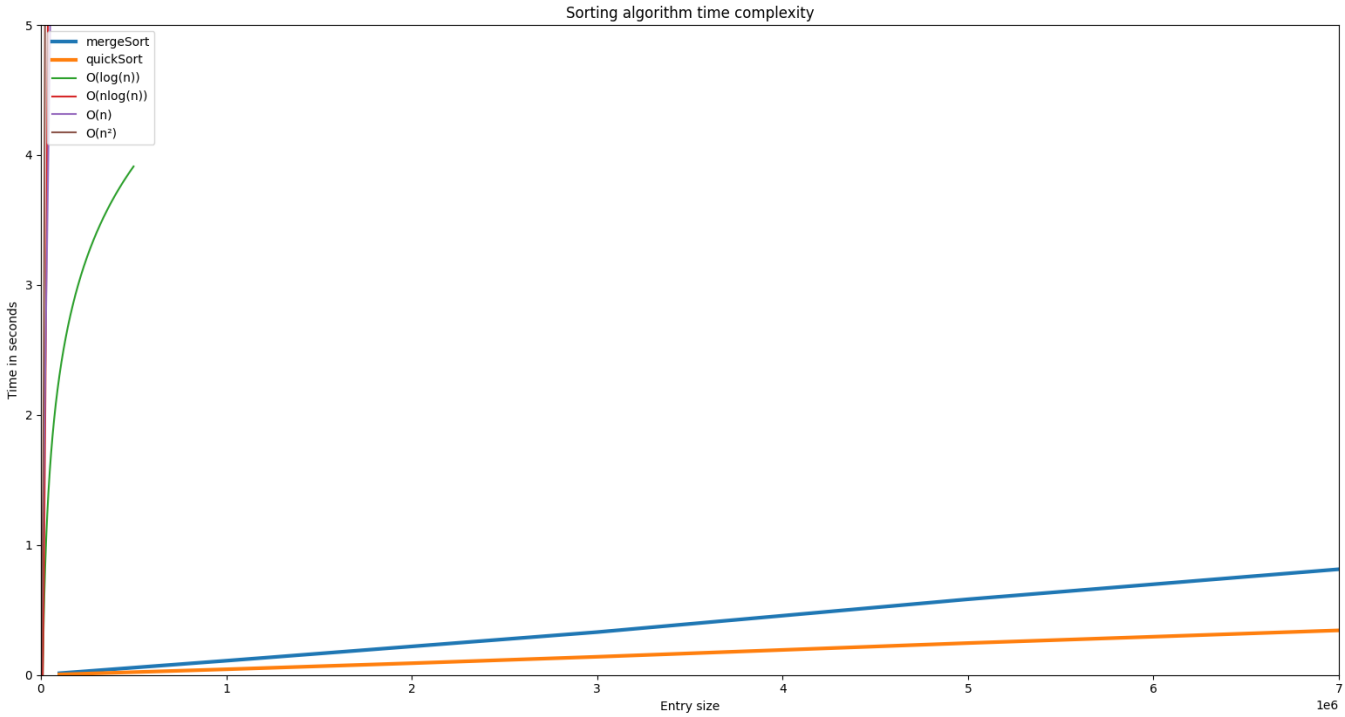


Note: les lignes en gras correspondent aux algorithmes testées. Les autres lignes sont des lignes de référence de complexitée, multipliées par 10000 (par exemple O(n) est une fonction f(n) = 10000n)

Malheureusement, il ne m'a pas été possible de passer à des tailles de tableau plus grandes, car les algorithmes de tri à bulle et d'insertion étaient très lent.
Nous allons donc déja essayer de classifier ces algorithmes, puis continuer avec les 2 autres.

L'algorithme par insertion et de tri à bulles semblent être en nlog(n), car leur courbes sont ressemblantes.
Cependant, on voit sur wikipedia que dans le cas moyen, la complexité en temps de ces algorithmes est de n^2. On peut donc s'attendre à ce qu'avec une taille d'entrée plus conséquence, ces courbes ressemblent plus à n^2

Comparons maintenant les algorithmes par fusion et le tri rapide:

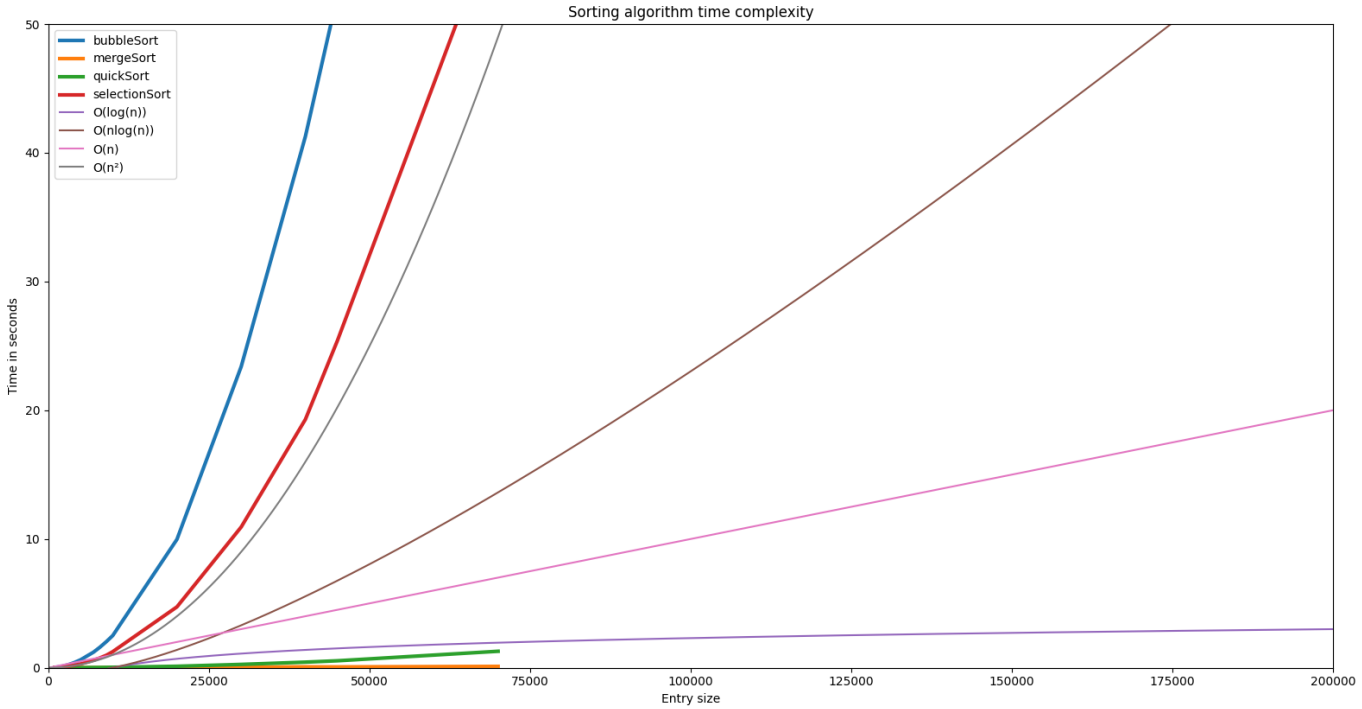


On note que ces algorithmes sont beaucoup pus rapides: ils parvienenent à trier une liste de taille 7e6 (7 millions) en environ une seconde.
Sur wikipedia, on voit que ces deux algorithmes sont sensés avoir une complexité moyenne en temps de nlog(n). Cependent, sur ce graphique, nlog(n) est la courbe rouge que l'on peut voir sur la gauche. Je ne suis pas sûr de comment interpréter ce résultat, il me parait douteux que ces courbes se mettent à se ressembler avec une taille d'entrée plus grande.

II)

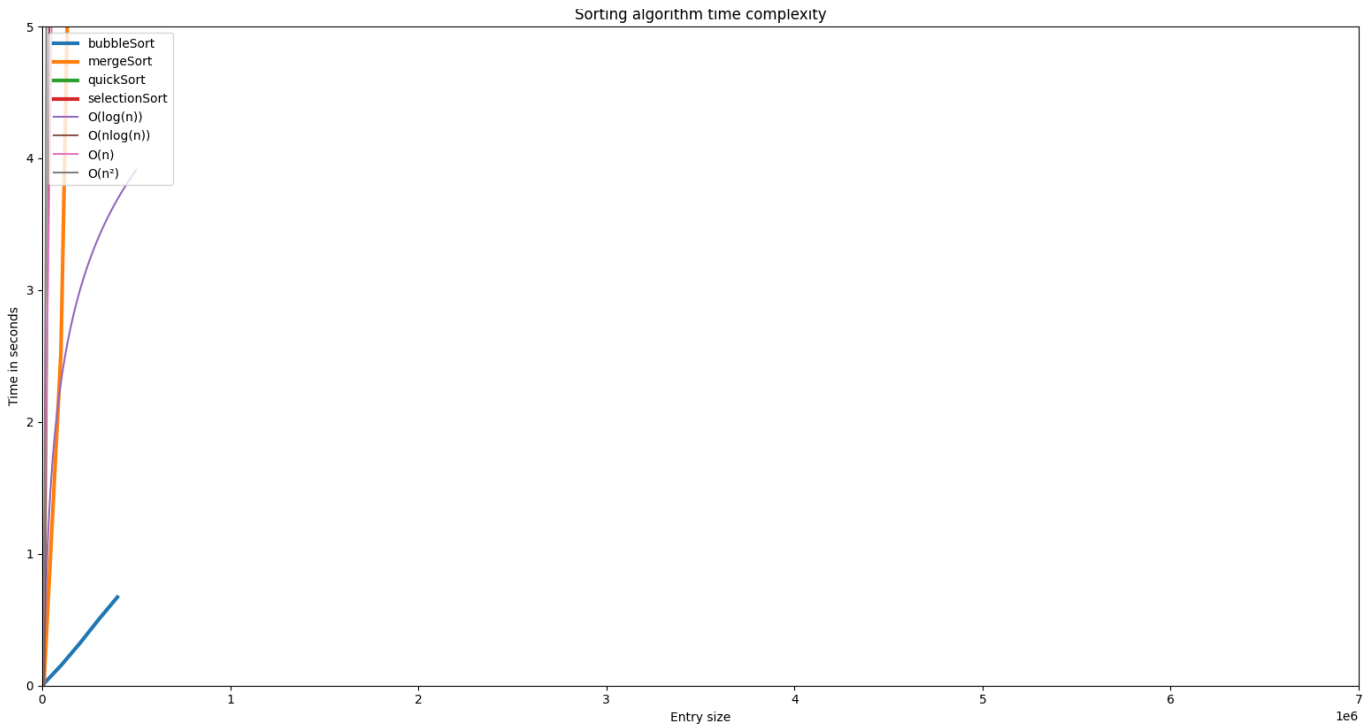
Nous allons maintenant recoder tout les algorithmes vus précédemment avec python

Voici le graphique permettant de comparer les 4 algorithmes. J'ai gardé la même échelle que pour le précédent, mais il y a maintenant une différente notable:

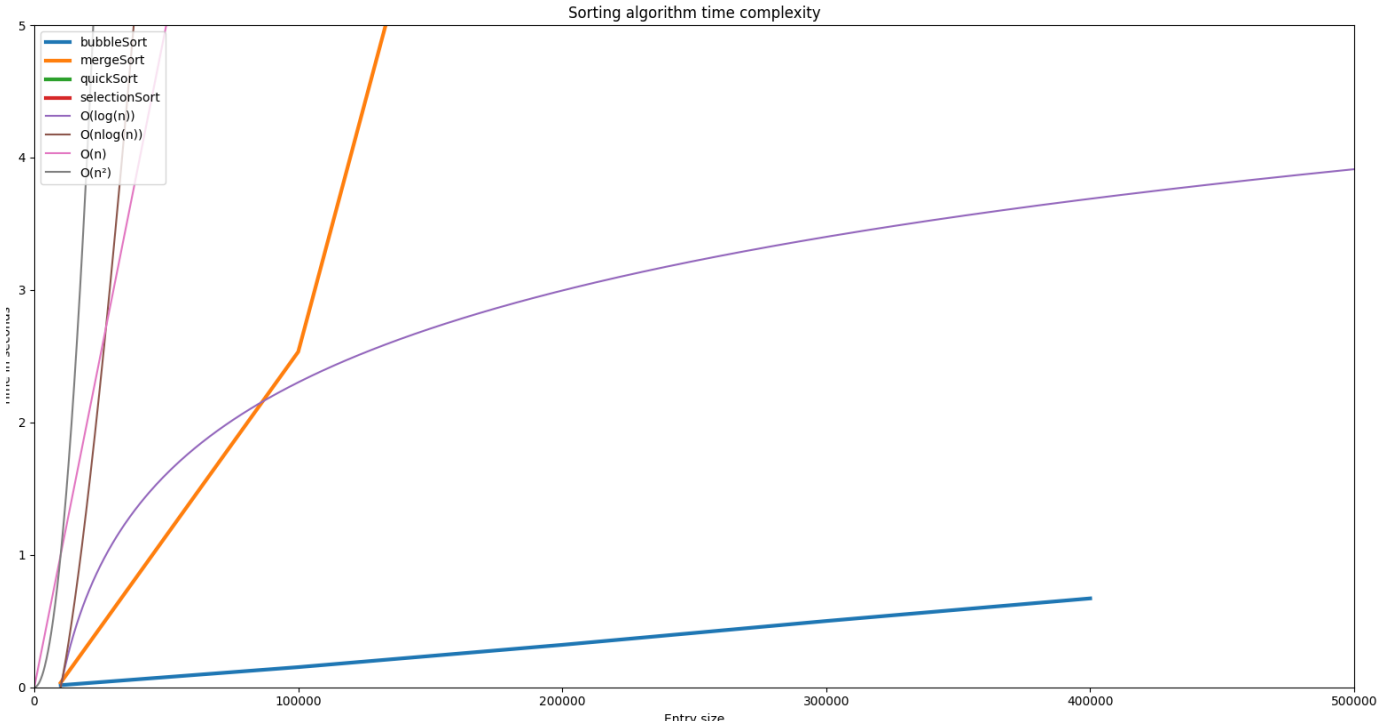


Comment on peut le voir, je n'ai même pas pu arriver jusqu'a mon cas précédent de 200000. Les algorithmes de tri à bubbles et tri par selection ont directement suivi la complexité de n^2, m'empêchant très rapidement d'augmenter la taille de l'entrée.
Ceci est curieux, car je ne m'attendais pas à ce que changer de langage puisse faire une différence aussi soulignée, ou l'on dirait presque que l'algorithme à changé de complexité.

Une fois ces deux algorithmes supprimés, on se retrouve avec ce schéma, pour l'algorithme par fusion et le tri rapide:



Ce schéma n’est pas très lisible, je vais donc changer l’échelle



On voit que la complexité est similaire à celle de n (similairement au programme en C++), mais elle est légérement plus élevée. je pense pouvoir attribuer ca au changement de langage