# Complexité des algorithmes de tri

**Mesures expérimentales sur des algorithmes de tri de listes chaînées**

Oscar Plaisant, Maximilien Poncet

## Introduction

Ce projet a pour but de mesurer expérimentalement les différences de temps d'exécution entre différents algorithmes de tri.

Tous les algorithmes sont implémentés sur des listes chaînées.

Les types d'implémentations sont variés : algorithmes itératifs, récursifs, en place ou non.

## Algorithmes et implémentations

Voici une liste des types d'algorithmes implémentés

- Tri par insertion
  - Tri par insertion récursive (la fonction d'insertion est récursive)
  - Tri par insertion itérative (la fonction d'insertion est itératifs)1
- Tri à bulles
  - Tri à bulles récursif (implémenté en utilisant les listes chaînées comme des piles)
  - Tri à bulles itératif en place (les valeurs sont déplacés dans la liste elle-même)
- Tri par éclatement-fusion (avec une fusion itérative)
- Tri rapide ($1^{er}$ élément comme pivot)
- Tri par seaux (sur des entiers positifs seulement)

# Comparaison des algorithmes

## Complexité temporelle

Voici la complexité temporelle théorique de chaque algorithme, où $n$ est le nombre d'éléments de la liste à trier.

| Algorithme | Meilleur des cas | Moyenne | Pire des cas |
|---|---|---|---|
| tri par insertion | $O(n)$ | $O(n^2)$ | $O(n^2)$ |
| tri à bulles | $O(n)$ | $O(n^2)$ | $O(n^2)$ |
| tri par éclatement-fusion | $O(n \times \log n)$ | $O(n \times \log n)$ | $O(n \times \log n)$ |
| tri rapide | $O(n \times \log n)$ | $O(n \times \log n)$ | $O(n^2)$ |
| tri par seaux | $O(n)$ | $O(n)$ | $O(n^2)$ |

Les complexités données ici sont théoriques et asymptotiques. Le but de ce projet est justement d'obtenir des valeurs plus concrètes, notamment des valeurs pour des listes plus petites, et pour lesquelles ces complexités ne s'appliquent pas.

Les tests menés ensuite ne sont pas toujours faits sur les meilleurs et pires des cas pour les algorithmes utilisés. Ils sont faits sur des listes aléatoires (complexité moyenne), sur des listes croissantes (meilleur des cas pour le tri à bulles) et sur des listes décroissantes (meilleur des cas pour le tri par insertion, pire des cas pour le tri à bulles).

## Prédictions théoriques

Nous allons utiliser les complexités asymptotiques théoriques pour essayer de prédire les données obtenues expérimentalement ensuite.

Ces complexités sont asymptotiques ; elles prédisent donc uniquement des résultats pour des listes avec un grand nombre d'éléments.

Les meilleurs et pires des cas ne sont pas les mêmes selon les algorithmes. Nous allons donc comparer les complexités moyennes.

On observe 3 complexités différentes parmi ces algorithmes : $O(n^2)$, $O(n \times \log n)$ et $O(n)$.

La plus grande longueur de liste que l'on ait testée est 100000. Pour $n = 100000$, on a $n^2 = 10^{10}$ et $n \log n = 5 \cdot 10^5$.

Donc, pour cette taille de liste, les algorithmes de complexité $O(n^2)$ devraient être environ $10^5$ fois plus lents que les algorithmes de complexités $O(n \log n)$, qui devraient être eux-mêmes $10^5$ fois plus lents que les algorithmes de complexités $O(n)$

On peut faire le calcul en général :

$O\left(\frac{n^2}{n}\right) = O(n)$ : les algorithmes en $O(n)$ sont $n$ fois meilleurs que ceux en $O(n^2)$

$O\left(\frac{n^2}{n \log n}\right) = O\left(\frac{n}{\log n}\right)$ : les algorithmes en $O(n \log n)$ sont $\frac{n}{\log n}$ fois meilleur que ceux en $O(n^2)$

$O\left(\frac{n \log n}{n}\right) = O(\log n)$ : les algorithmes en $O(n)$ sont $\log n$ fois meilleurs que ceux en $O(n \log n)$

## Résultats expérimentaux

Nous allons comparer le temps d'exécution de différents algorithmes selon deux paramètres : le type de listes à trier (listes aléatoires, listes croissantes et listes décroissantes), et la longueur de ces listes (10, 100, 1000, 10000 et 100000 éléments).

Chaque mesure est faîte sur 10 échantillons (sauf pour les fonctions qui ont mis plus de 5s à s'exécuter). Les données brutes sont dans le fichier `src/saved_statistics/data`.

Sur les graphiques qui suivent, le temps est mesuré en secondes. Chaque point représente la moyenne de 10 échantillons (à part pour les fonctions qui ont mis plus de 5s à s'exécuter).

Sur chaque point, on a ajouté un disque dont l'aire en pixels est l'écart-type des valeurs des 10 échantillons.

Un grand disque est ajouté sur les valeurs qui n'ont été testés que sur 1 échantillon.

### Listes aléatoires

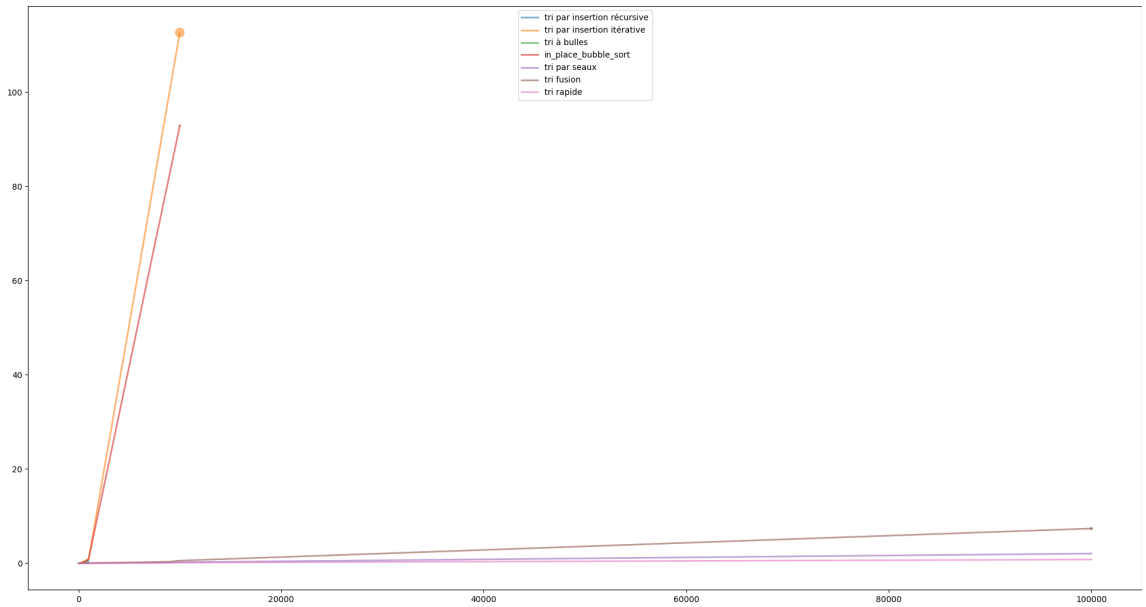
On remarque que les algorithmes de complexité $O(n^2)$ ne réussissent pas, ou bien prennent énormément de temps à trier des listes de plus de $10^4$ éléments. Il y a 2 raisons à cela :

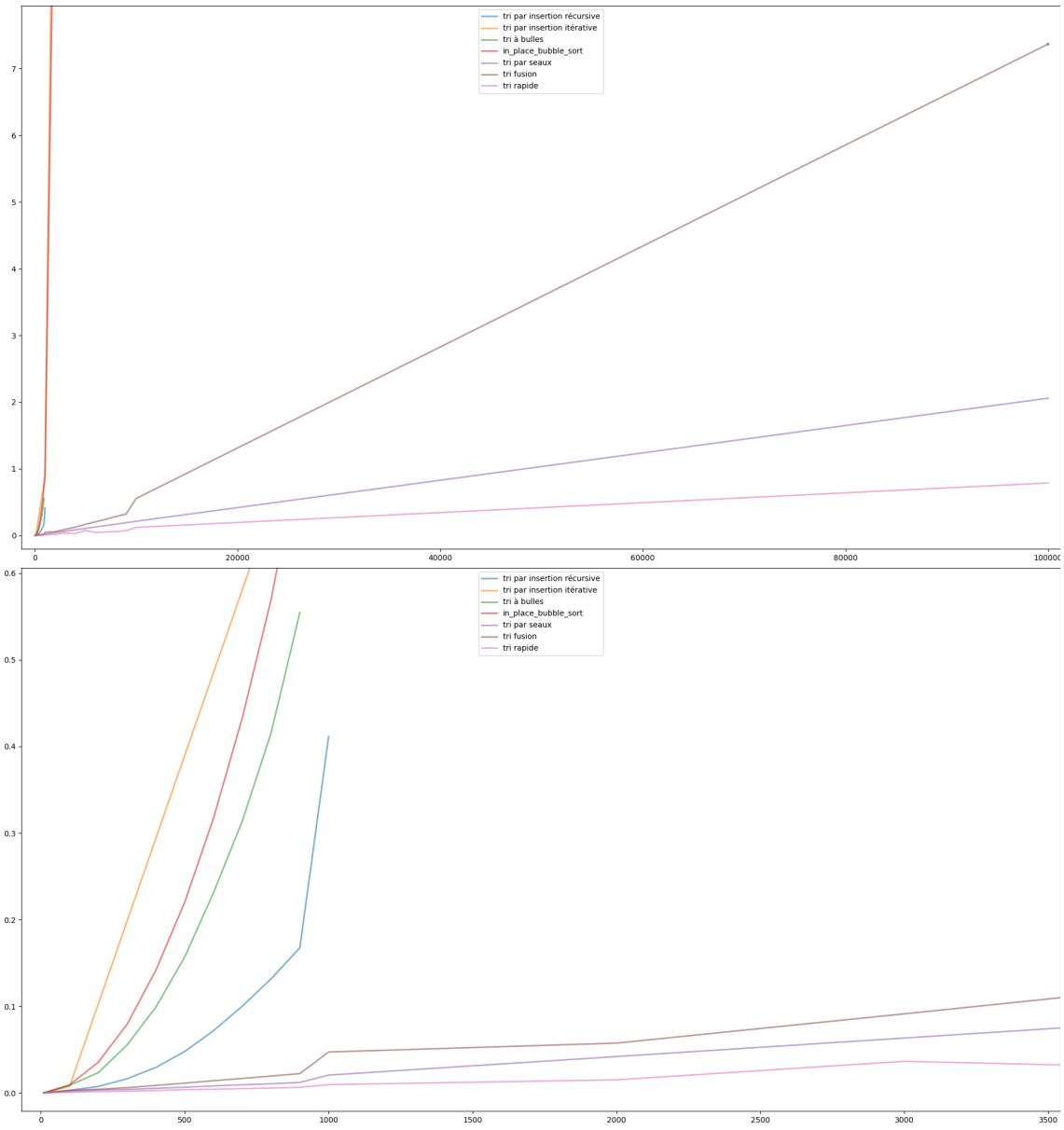
- Le temps d'exécution devient très grand (au-delà de la minute). Le programme de récolte des données arrête de tester les algorithmes quand ils prennent trop de temps.
- Si ces algorithmes sont codés récursivement, il faut faire de l'ordre de $10^16$ appels récursif pour trier une liste de $10^4$ éléments. Cela remplit la pile d'appels et mène à une erreur

Il n'y a donc que 3 algorithmes qui permettent de trier de très grandes listes aléatoires : le tri par éclatement-fusion, le tri rapide et le tri par seaux.

Le tri par éclatement-fusion est le plus lent des trois. Le tri rapide est le plus rapide.

On pourrait s'attendre à voir que le tri par seaux est plus rapide que le tri rapide, puisque sa complexité est de $O(n)$ contre $O(n \log n)$. Cependant, on observe expérimentalement que le tri-rapide est quasi-linéaire pour des listes de moins de $10^5$ éléments. De plus, l'implémentation du tri par seaux, avec des listes chaînées plutôt qu'un tableau ou un dictionnaire pour stocker les seaux, le rend plus lent d'un facteur important, et cela accentue encore la différence.






### Listes croissantes

Les listes croissantes sont générées avec la fonction iota, c'est-à-dire que ce sont des listes contenant les $n$ premiers entiers naturels dans l'ordre croissant.

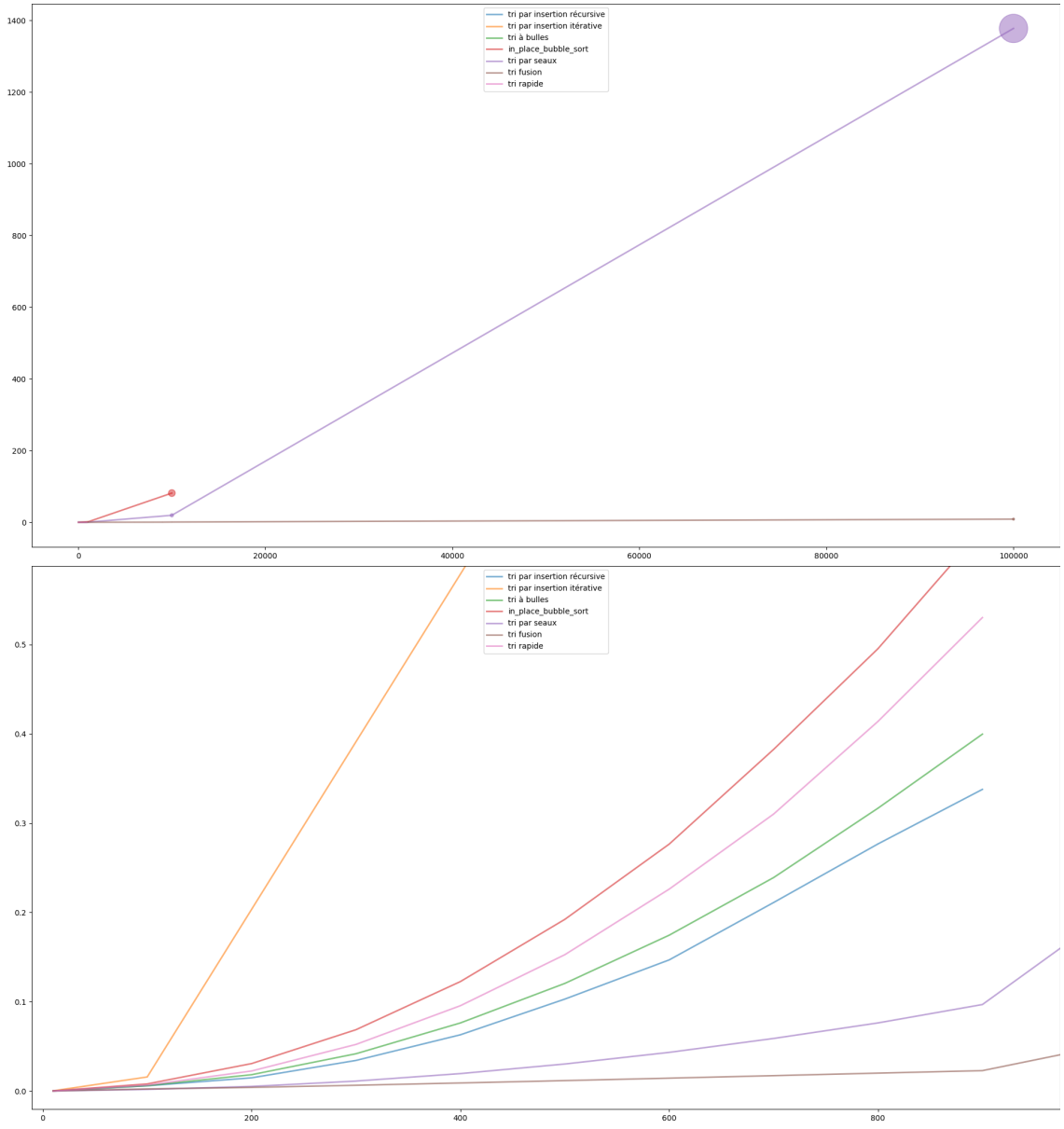
On observe d'abord que le tri par seaux est très peu efficace sur ce type de listes. Cela vient du fait que ces listes contiennent beaucoup d'éléments différents. Dans une implémentation du tri par seaux avec un tableau ou un dictionnaire pour stocker les seaux, le temps d'accès n'est pas changé quand on a beaucoup d'éléments différents. Mais comme on utilise une liste chaînée, lorsque le nombre d'éléments différents n'est plus négligeable (100 précédemment, pour les listes aléatoires), la complexité du tri par seaux devient $O(k \times n)$, où $k$ est le nombre d'éléments distincts dans la liste. Comme dans le cas de listes croissantes, on a $k = n$, la complexité du tri par seaux devient $O(n^2)$

On remarque, que les algorithmes de complexité $O(n^2)$ ne peuvent pas trier des listes de plus de 1000 éléments. Les raisons sont les mêmes que pour les listes aléatoires.

On remarque également que le tri rapide semble avoir une complexité de $O(n^2)$ (la courbe de ses temps d'exécution est une parabole). Cela est logique, puisque notre implémentation du tri rapide choisit

toujours le premier élément de la liste comme pivot. Cela implique que le pivot est toujours le plus petit élément de la liste, et donc que toutes les autres valeurs seront supérieures. Dans ce cas, le tri rapide devient équivalent à un tri par sélection (récursif), et à donc bien une complexité de $O(n^2)$.

Dans ce cas, le tri le plus efficace en temps est le tri par éclatement fusion.



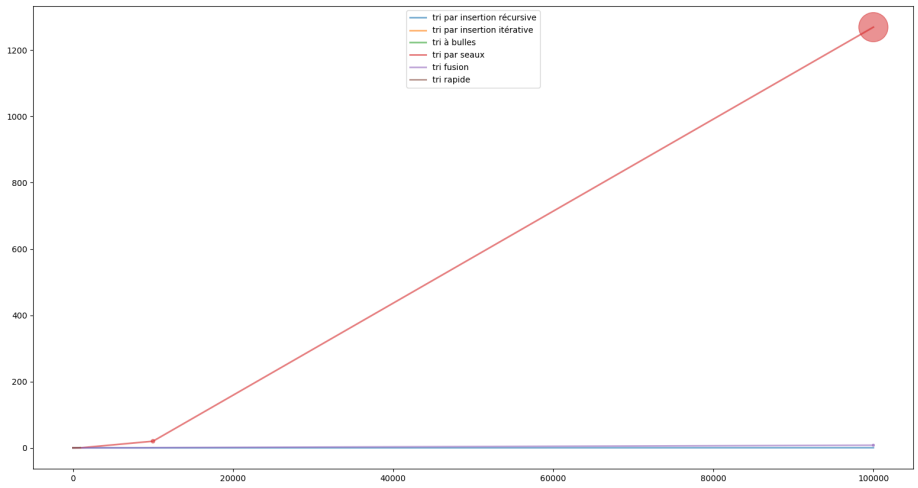

### Listes décroissantes

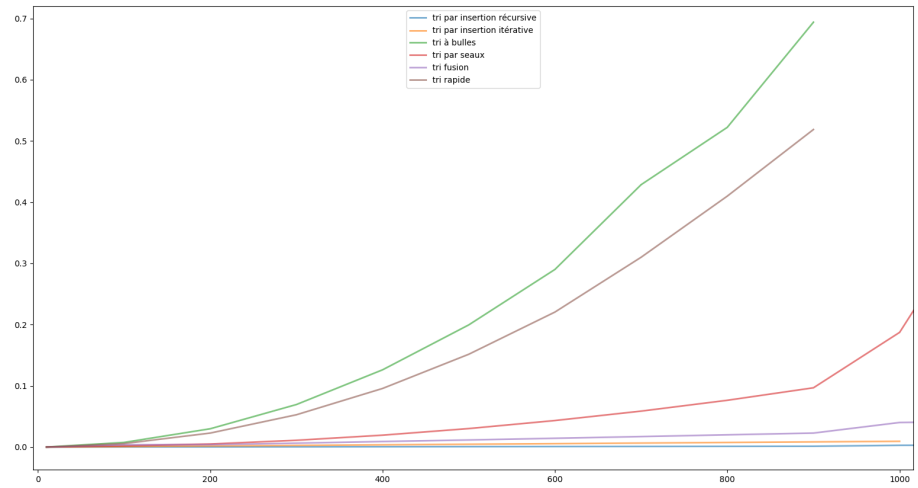
Les listes décroissantes sont des listes contentant les $n$ premiers entiers naturels dans l'ordre décroissant.

Comme dans le cas des listes croissantes, on observe que le tri par seaux à une complexité de $O(n^2)$, à cause du grand nombre d'éléments distincts, et de son implémentation à base de listes chaînées.

On observe encore que les algorithmes en $O(n^2)$ ne peuvent pas trier des listes de plus de 1000 éléments.

Cependant, on remarque que le tri par insertion récursive n'est pas en $O(n^2)$, mais en $O(n)$. Cela est dû au fait que, dans le cas d'une liste décroissante, le tri par insertion récursive est équivalent à un retournement de la liste (fait avec deux piles, des listes chaînées dans notre cas), qui est bien en $O(n)$. L'algorithme va, à chaque étape, insérer le premier élément non encore parcouru en première place (car il est plus grand que tous ceux parcourus). Cela n'est pas vrai pour le tri par insertion itérative, qui reste en $O(n^2)$, car son implémentation ne permet pas cette optimisation.

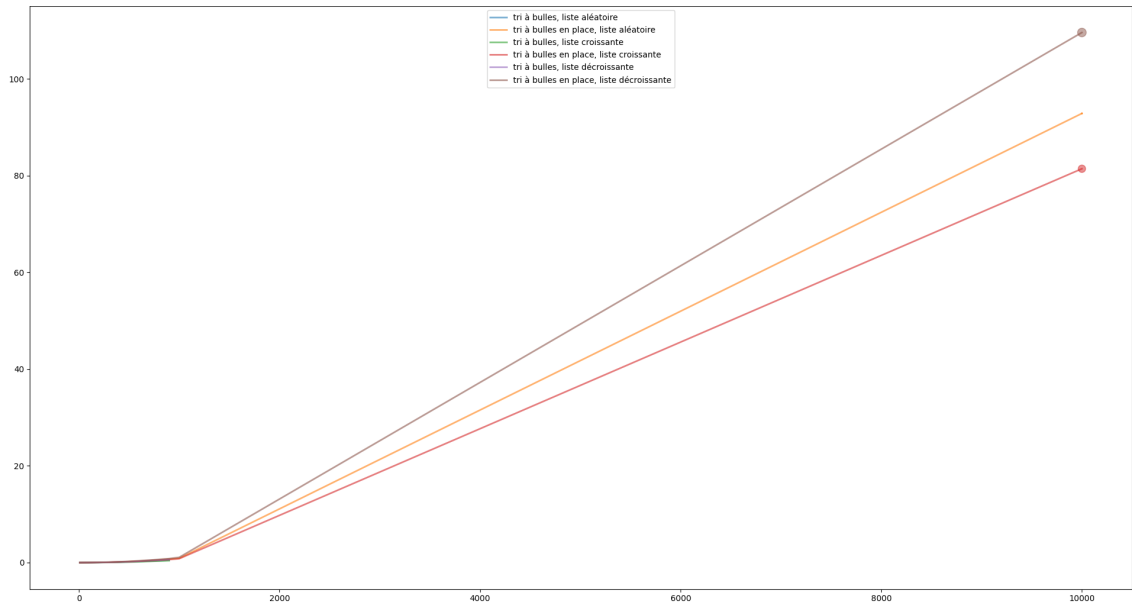


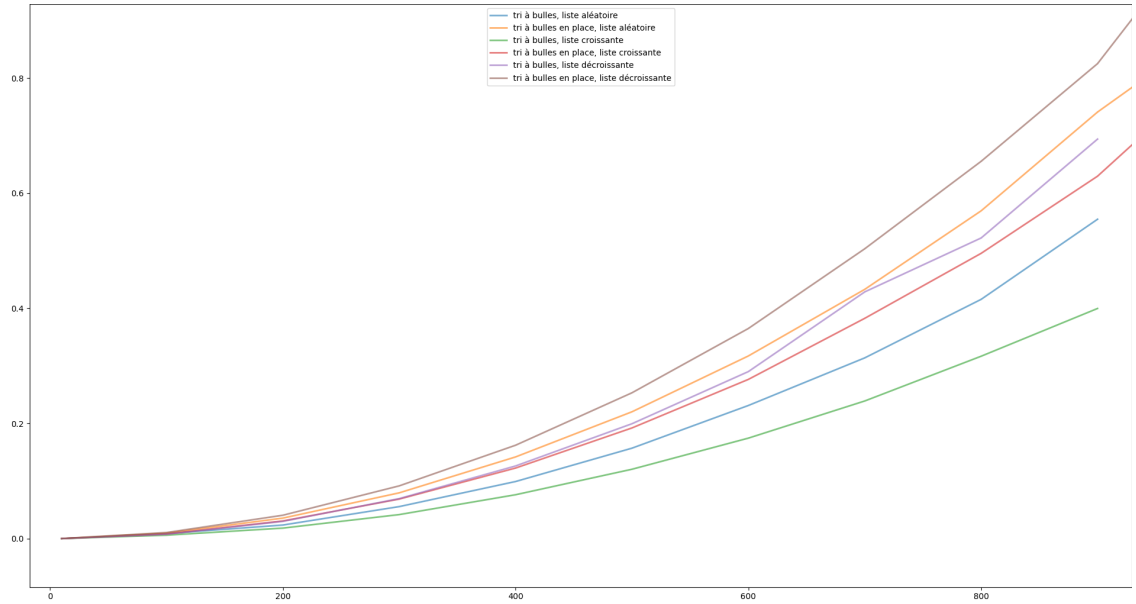




### Cas du tri à bulles

Il est intéressant de comparer les temps d'exécution du tri à bulles selon le type de listes.

On remarque qu'il est plus efficace sur des listes déjà triées, et moins efficace sur des listes décroissantes. Cela est dû au fait que le tri à bulle est long lorsqu'il y à beaucoup de "bulles", c'est-à-dire de valeurs à déplacer vers la fin de la liste. Notamment, les "tortues" (les grandes valeurs au début de la liste, et les petites valeurs à la fin) ralentissent beaucoup le tri à bulles, puisqu'il met beaucoup de temps à les déplacer. Justement, les listes croissantes minimisent le nombre de tortues, et les listes décroissantes les maximisent.


tri à bulles, liste aléatoire
tri à bulles en place, liste aléatoire
tri à bulles, liste croissante
tri à bulles en place, liste croissante
tri à bulles, liste décroissante
tri à bulles en place, liste décroissante
0
20
40
60
80
100
0
2000
4000
6000
8000
10000


tri à bulles, liste aléatoire
tri à bulles en place, liste aléatoire
tri à bulles, liste croissante
tri à bulles en place, liste croissante
tri à bulles, liste décroissante
tri à bulles en place, liste décroissante
0.0
0.2
0.4
0.6
0.8
0
200
400
600
800