INF4705 — Analyse et conception d'algorithmes

TP1

l'analyse empirique et hybride des algorithmes.

Fait par

Thierry Skoda

1634927

21 Octobre 2016

École Polytechnique de Montréal

Introduction

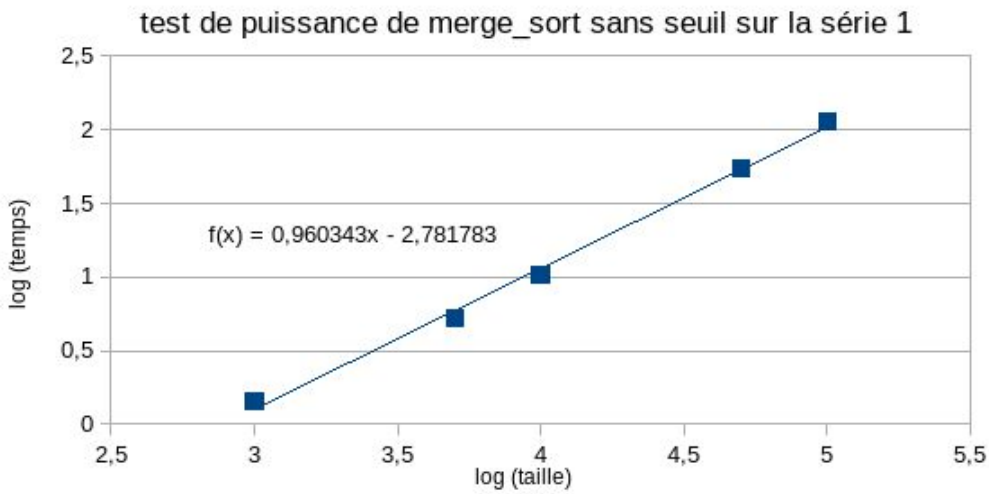
En tant qu'étudiant en Génie logiciel, durant notre parcours académique, nous avons appris plusieurs algorithmes ainsi que la complexité pour chacun d'eux. En effet, nous avons appris que chaque algorithme à ses avantages et ses désavantages. Cependant, il est important de comprendre que ce qui est écrit dans les livres n'est pas toujours exactement ce qui se passe dans la réalité. Pour ce laboratoire, nous avions la tâche de mettre en pratique trois approches d'analyse de l'implémentation d'un algorithme (Test de puissance, test du rapport et test des constantes). Les algorithmes que nous avons analysés sont le Merge_sort et le Bucket_sort. Par ailleurs, pour chacun d'entre eux, il fallait faire une deuxième version afin d'y ajouter un seuil de récursivité. Pour la suite du rapport, nous allons appliquer le test de puissance pour chacune des séries d'exemplaires et citer la notation asymptotique en meilleur cas, cas moyen et pire cas. Par la suite, nous allons faire le test du rapport pour chacune des séries d'exemplaires en utilisant chacune des analyses en meilleur cas, en cas moyen et en pire cas. Puis, encore pour chacun des algorithmes et chacune des séries, nous allons préciser l'analyse asymptotique théorique en calculant les constantes en jeu qui est le dernier test (test des constantes). Ensuite, nous allons discuter de l'impact que peut avoir un seuil de récursivité sur ces algorithmes et, finalement, nous allons indiquer sous quelles conditions nous utiliserions chacun de ces algorithmes.

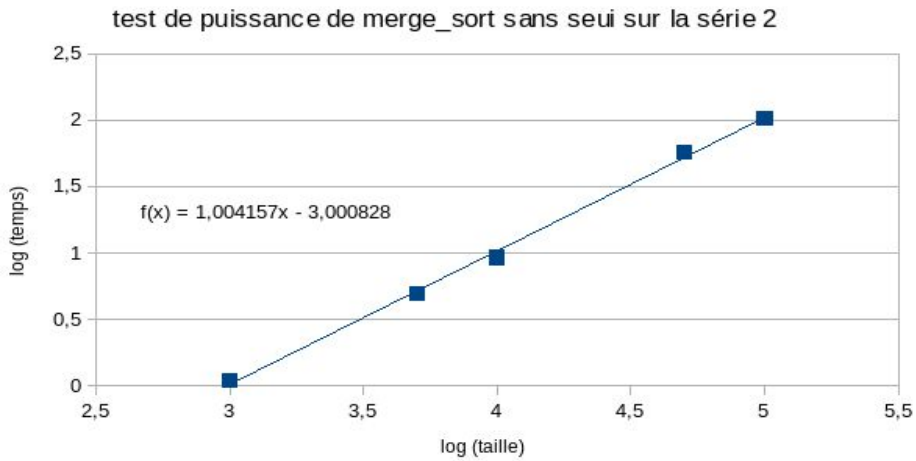
ANALYSE

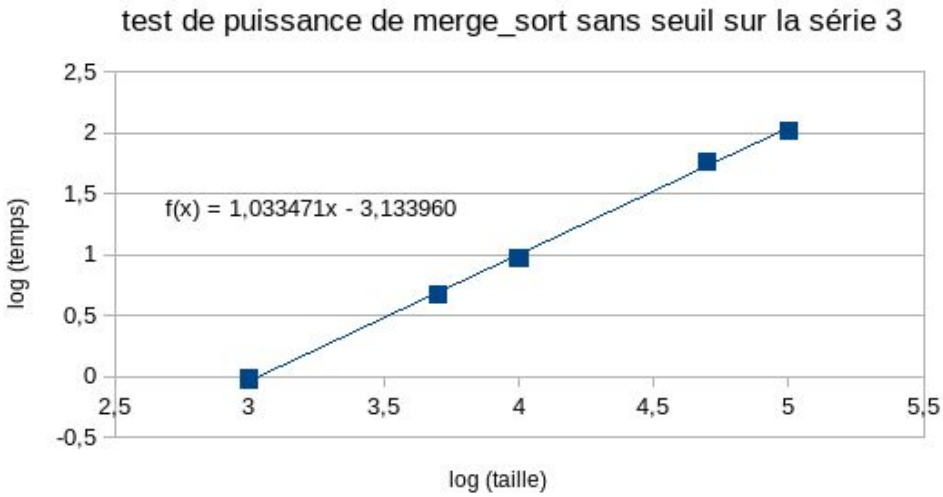
i) Test de puissance

On suppose que le temps de calcul des algorithmes y s'écrivent sous la forme “y = a x^m + b” où “x” est la taille des exemplaires, “a” et “b” sont des constantes et “m” la pente de la droite de la régression linéaire des graphiques du log du temps d'exécution en fonction du log de la taille des exemplaires. (on utilise pour chacun des graphiques suivant le log en base 10).

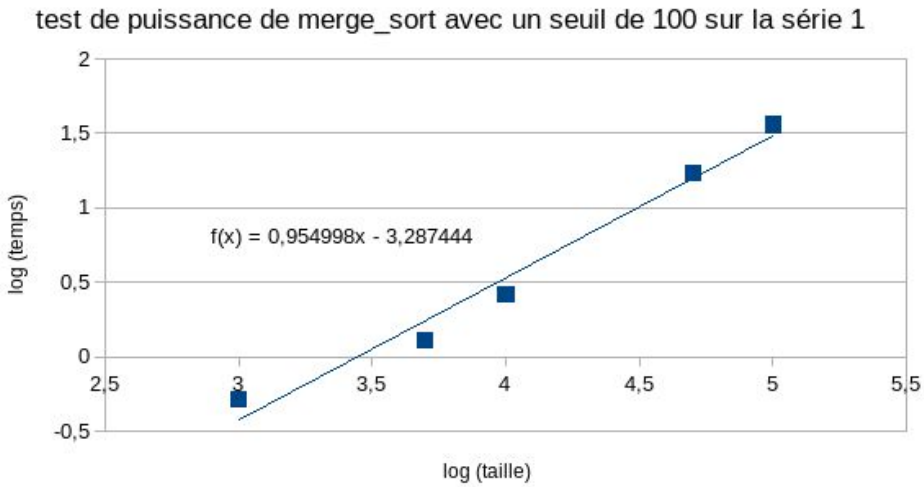
Test de puissance pour le Merge_sort sans seuil



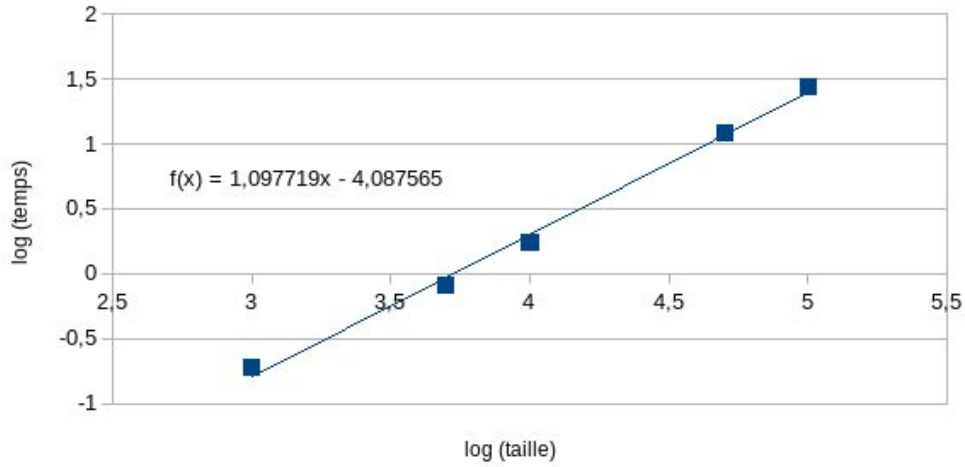




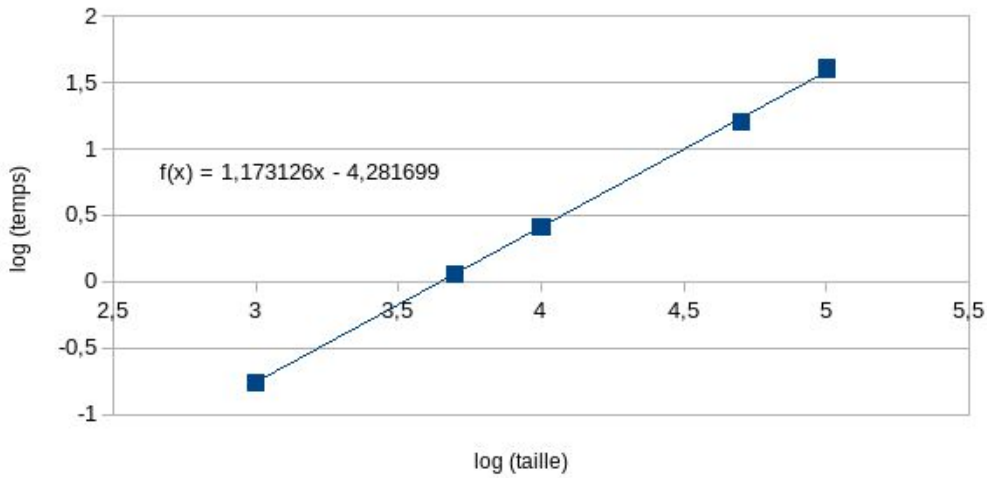
Test de puissance pour le Merge_sort avec un seuil de 100



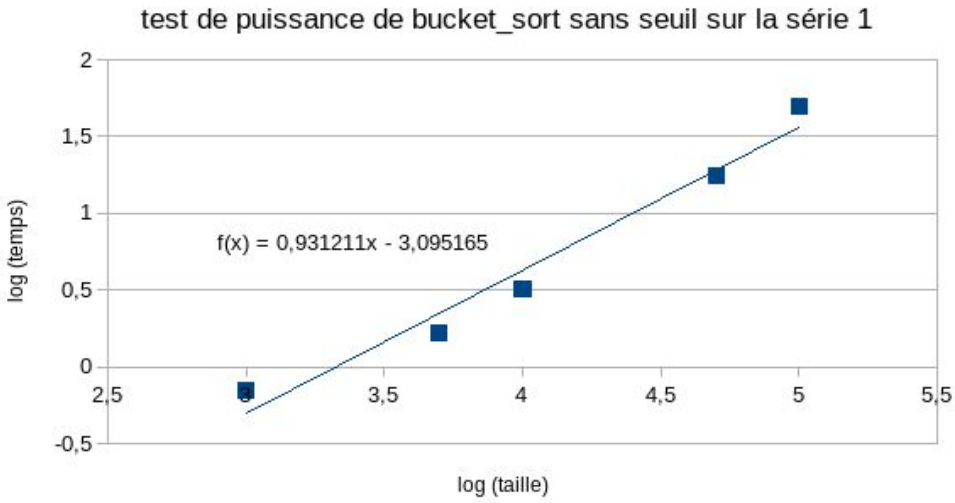
test de puissance de merge_sort avec un seuil de 100 sur la série 2

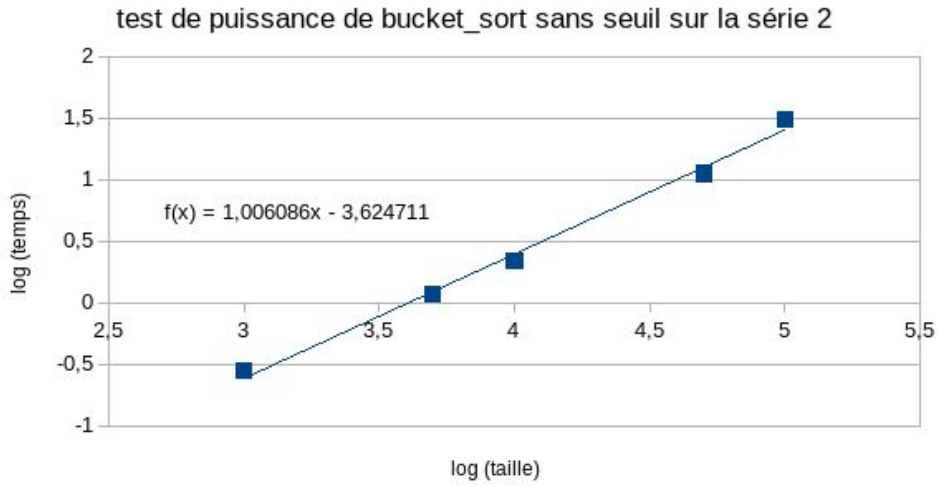


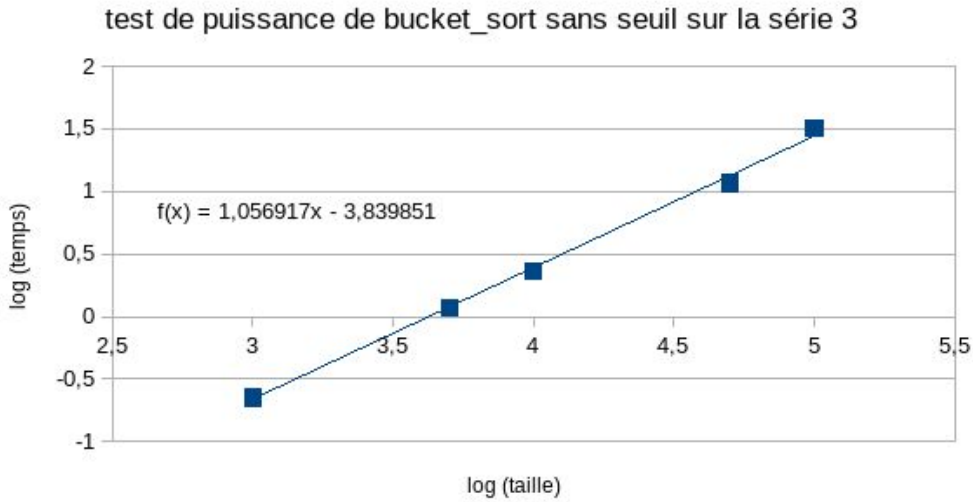
test de puissance de merge_sorte avec un seuil de 100 sur la série 3



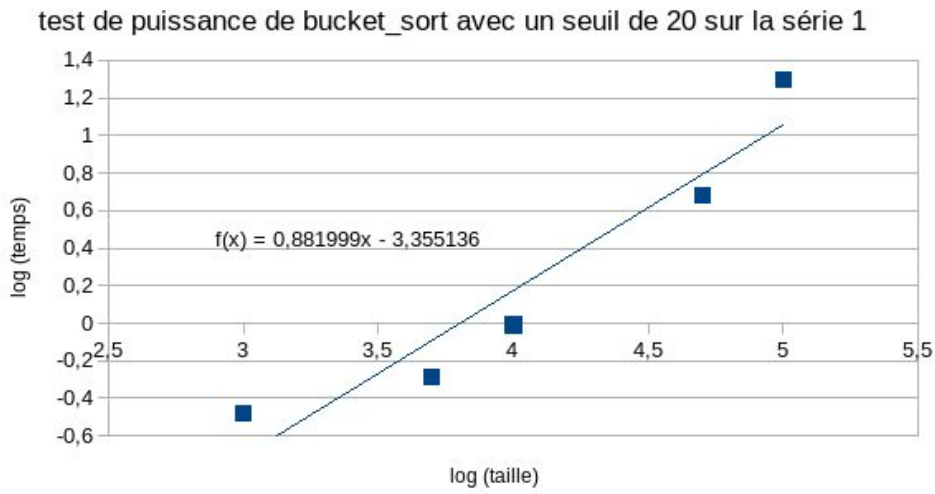
Test de puissance du Bucket_sort sans seuil

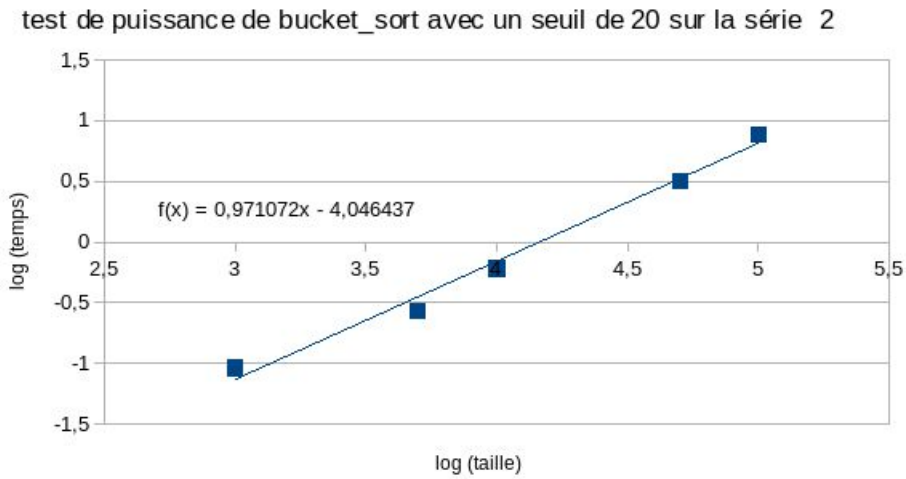


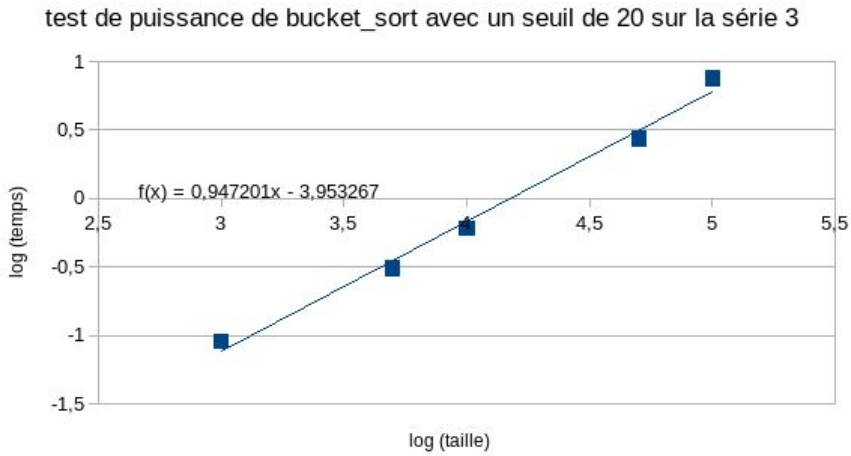




Test de puissance du Bucket_sort avec un seuil de 20







Analyse finale

Pour chacun des graphiques, on peut remarquer qu'on peut faire passer une droite à travers les points. On peut donc approcher la complexité des algorithmes par un polynôme avec un exposant de m (qui est le coefficient de chacune des droites) et une constante multiplicative évalué à 10^b.

ii) Complexité théorique

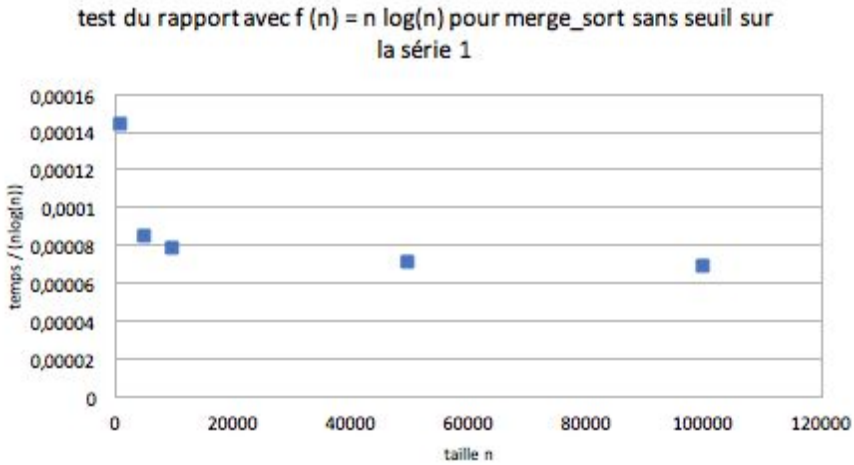
La complexité théorique du Merge_sort sans seuil et avec seuil est en n*log(n) en meilleur cas, en cas moyen et en pire cas.

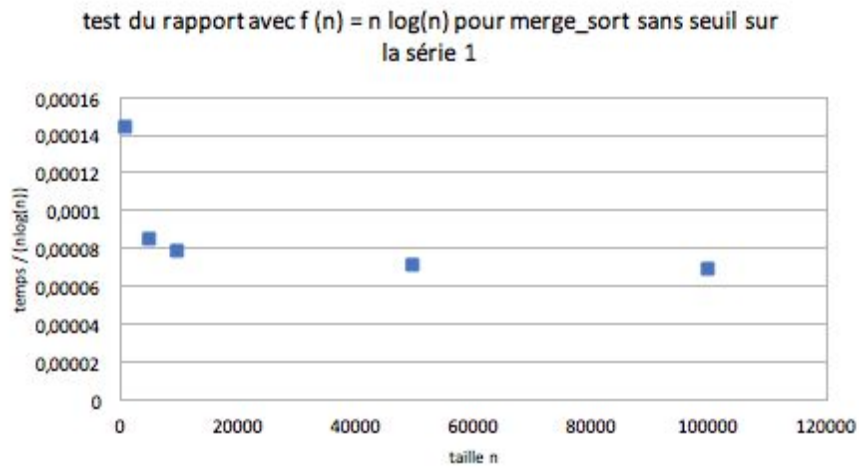
La complexité théorique du Bucket_sort sans seuil et avec seuil est en n*log(n) en meilleur cas et cas moyen. Par contre en pire cas sa complexité est en n^2.

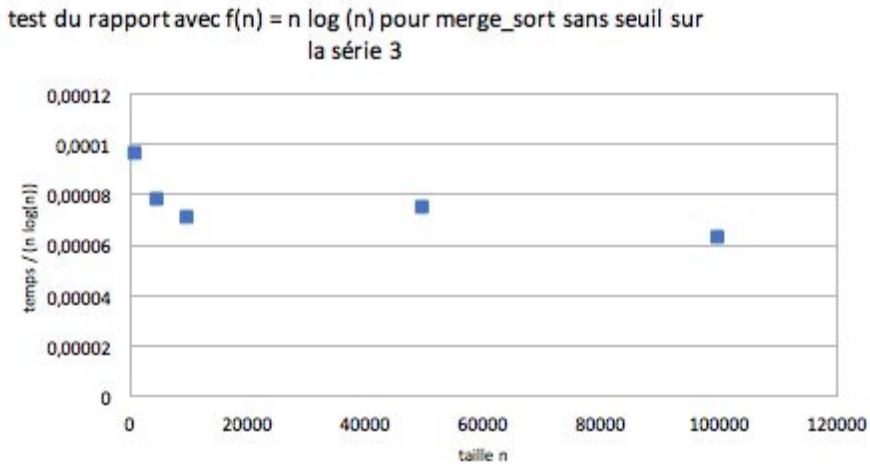
iii) Test du rapport

Test du rapport pour Merge_sort sans seuil

Avec n log(n) qui est à la fois la complexité théorique en pire cas, en cas moyen et en pire cas de Merge_sort

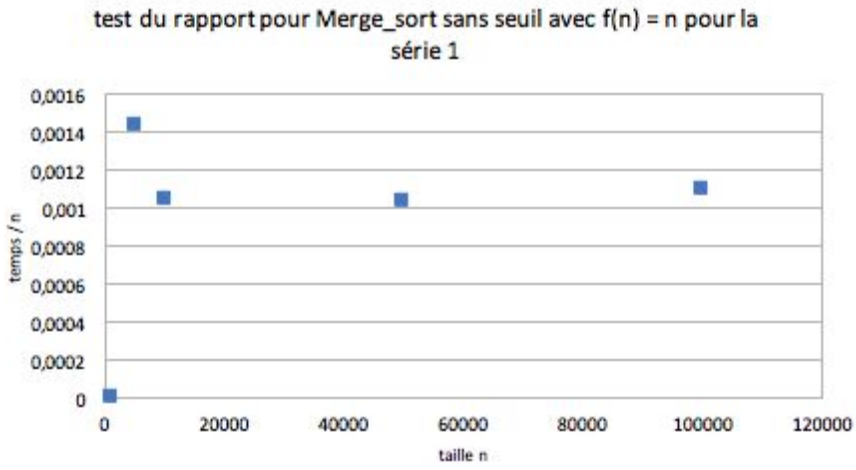


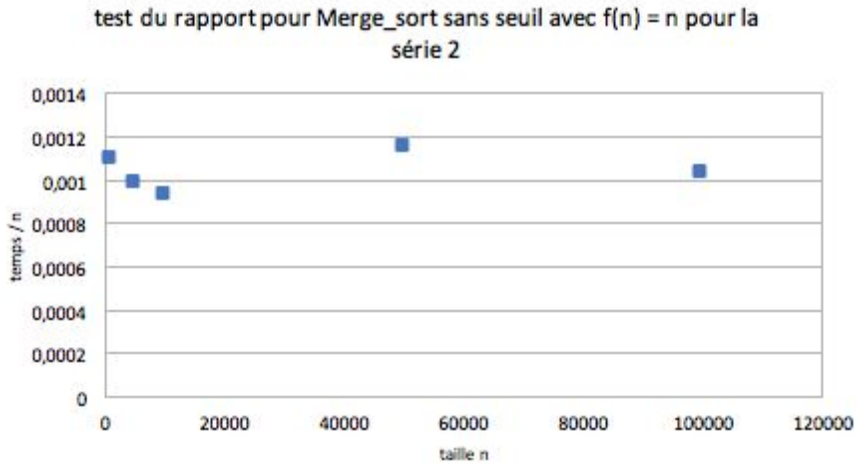


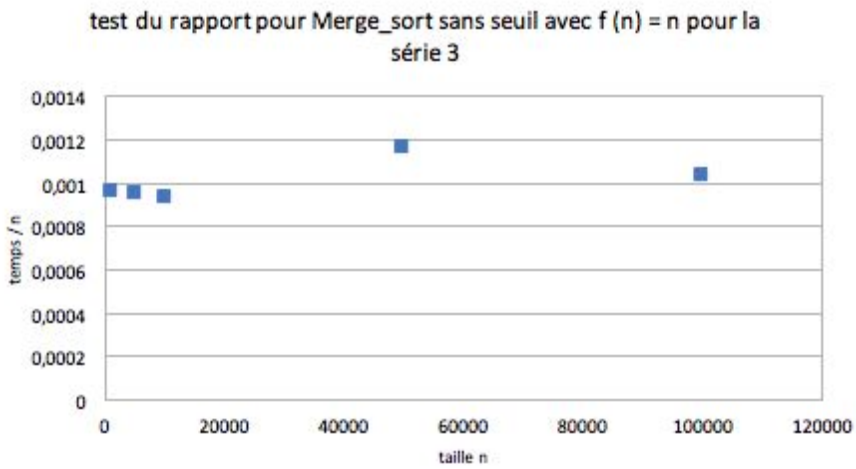


On observe que pour chacune des 3 séries le rapport temps / (nlog(n)) tend vers une constante. On en déduit que la complexité asymptotique de merge_sort est de O(n*log(n)).

Test du rapport pour Merge_sort sans seuil avec f(n) = n

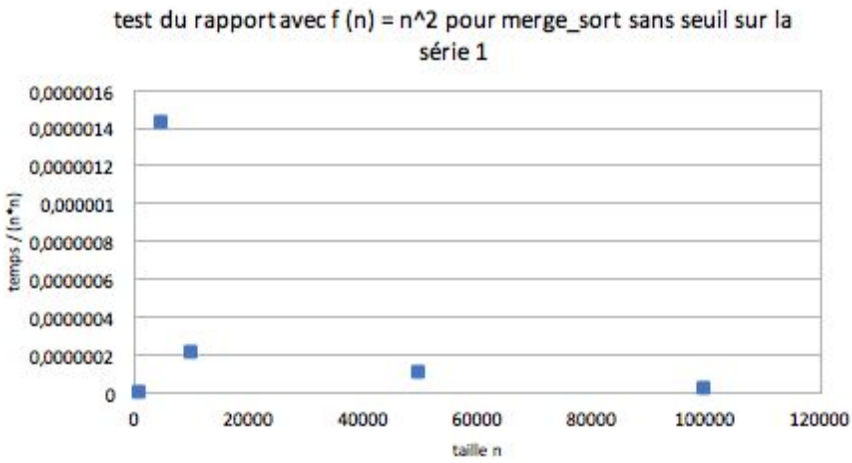


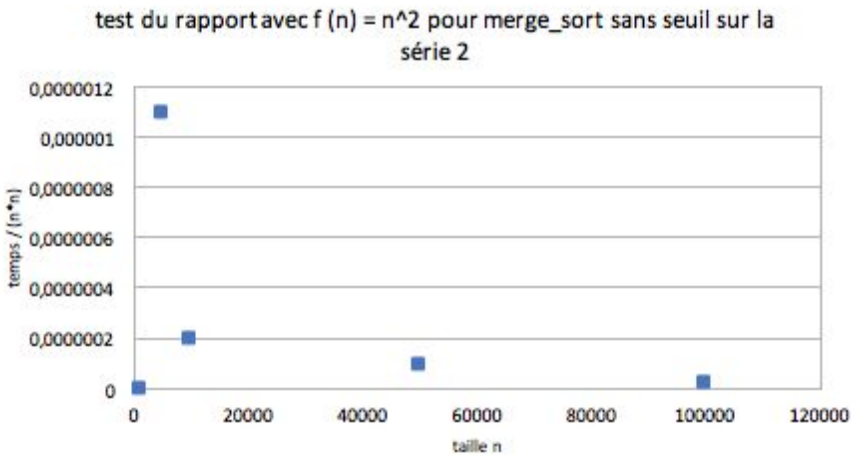


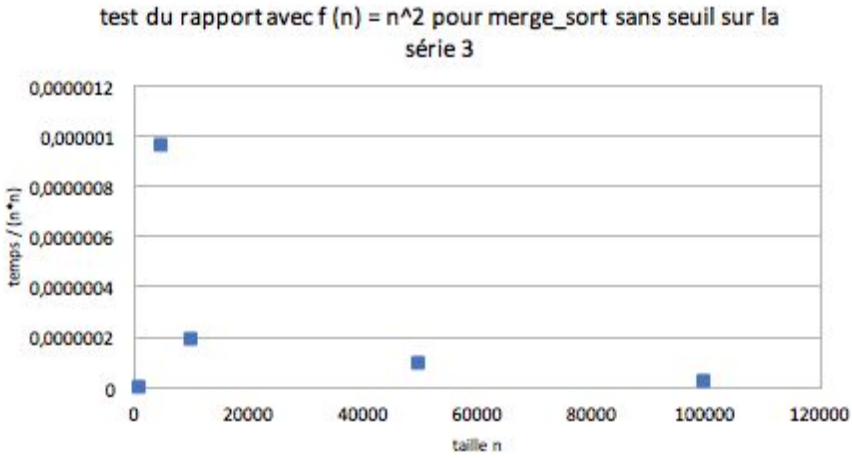


Il semble que le rapport temps / n tend lentement vers plus l'infini pour la série 1 et aussi les séries 2 et 3 si on ne considère pas l'avant dernier point. On en déduit que "f(n) = n" est une légère sous-estimation de la complexité de merge_sort sans seuil.

Test du rapport pour Merge_sort sans seuil avec f(n) = n^2



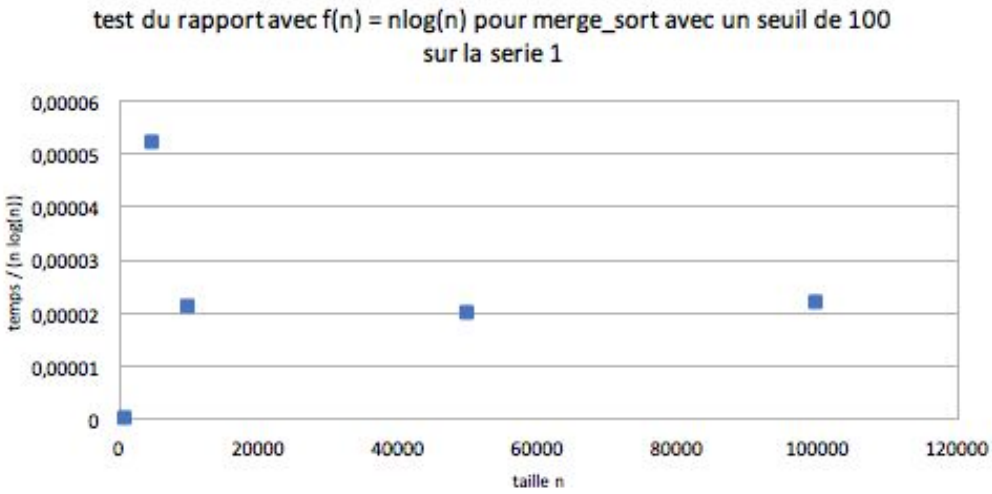


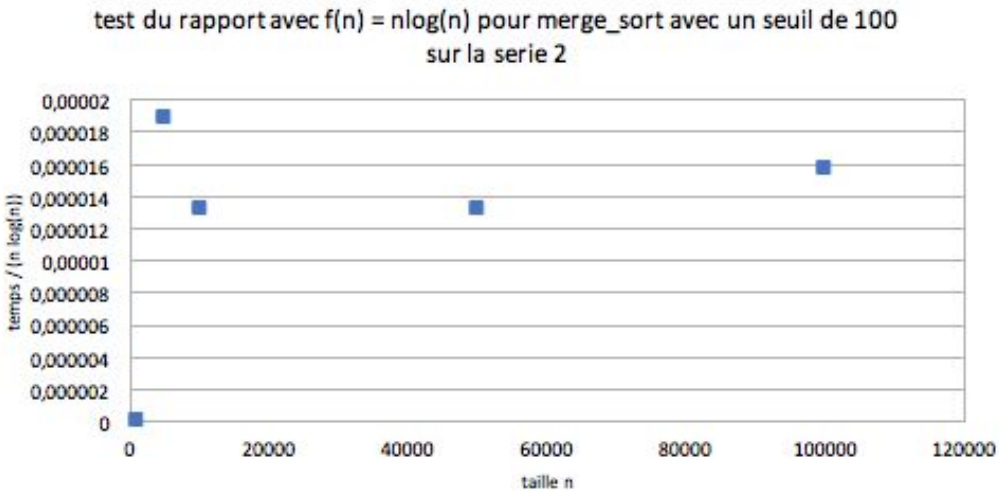


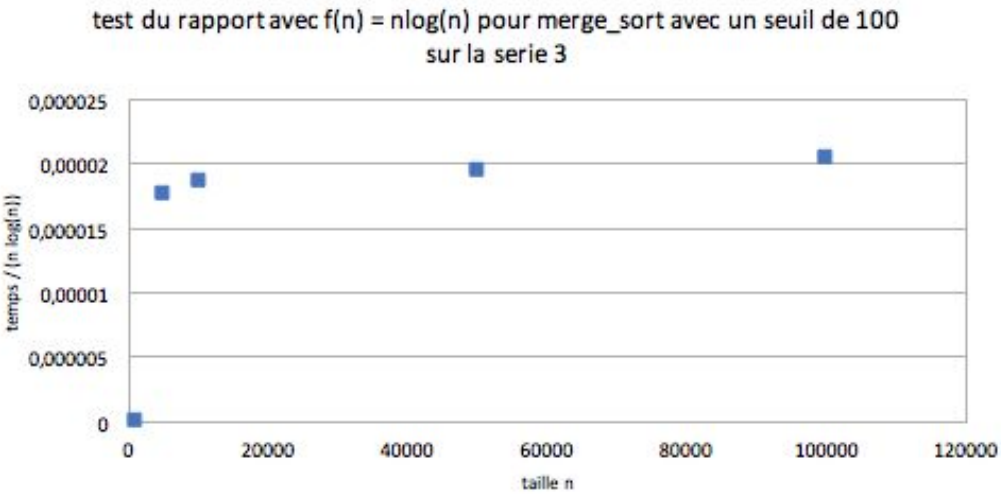
On observe que pour chacune des séries le rapport Temps / n^2 tend vers l'infini lorsque n augmente. On en déduit que "f(n)=n^2" est une surestimation de la complexité de Merge_sort sans seuil.

Test du rapport pour Merge_sort avec un seuil de 100

Test du rapport pour Merge_sort avec un seuil de complexité de 100 avec n log(n) qui est à la fois la complexité en pire cas, en cas moyen et en pire cas

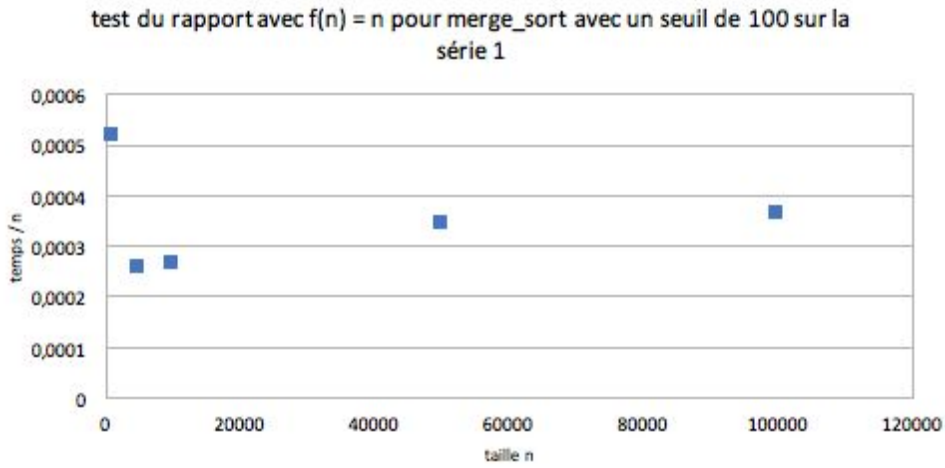


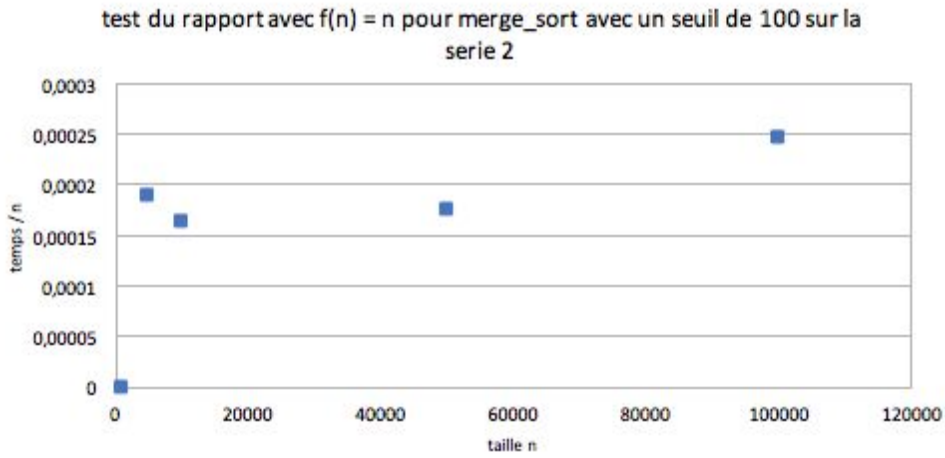


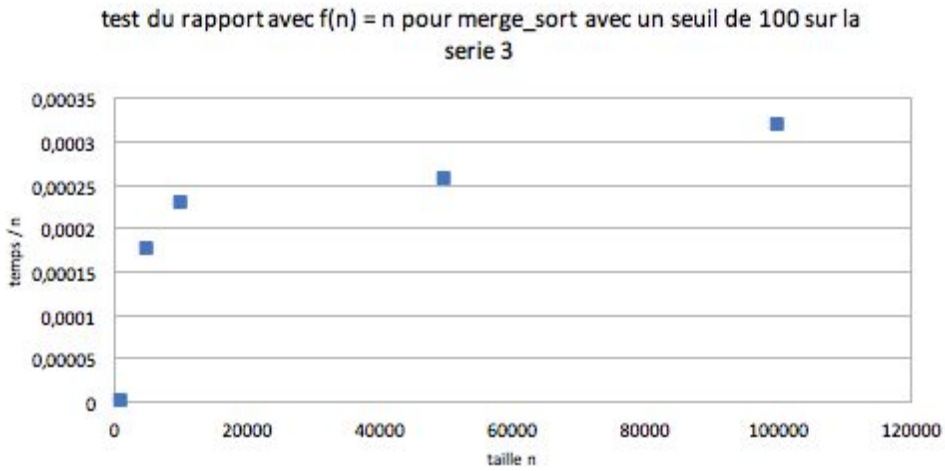


On observe pour les séries 1 et 3 que le rapport temps / (nlog(n)) tend vers une constante. On en déduit que la complexité de merge_sort avec un seuil de 100 est en n*log(n).

Test du rapport pour Merge_sort avec un seuil de 100 ave f(n) = n

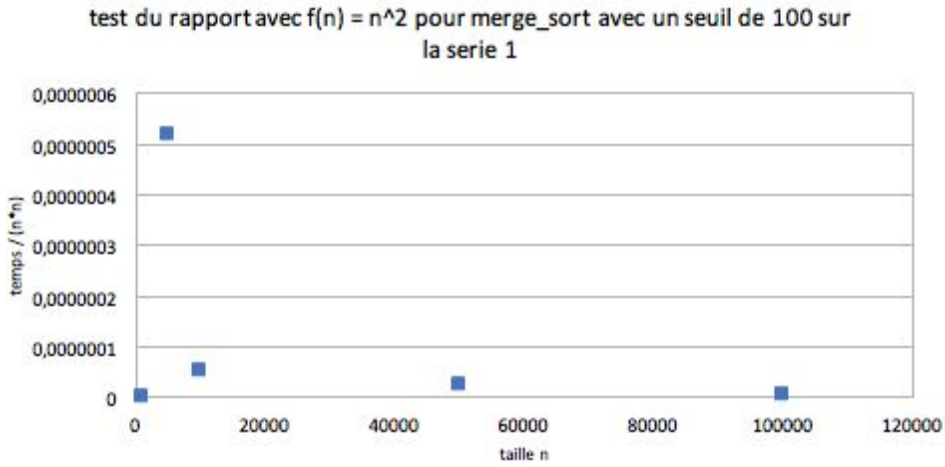


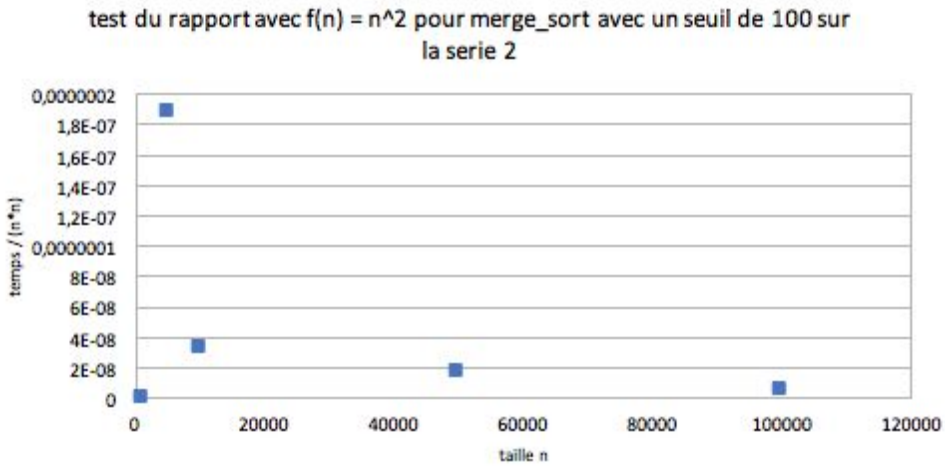


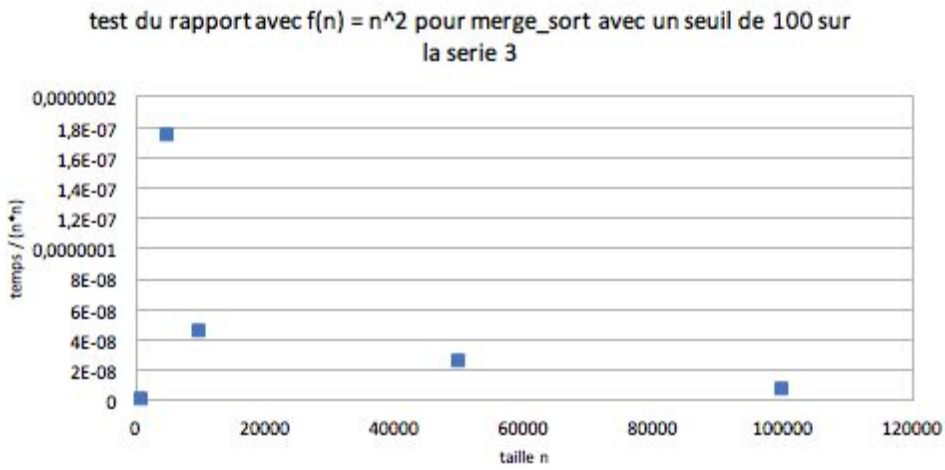


On remarque que pour chacune des séries le rapport Temps / n tend vers l'infini lorsque n augmente. On en déduit que "f(n) = n" est une sous-estimation de la complexité de Merge_sort avec un seuil de 100.

Test du rapport pour Merge_sort avec un seuil de 100 ave f(n) = n^2



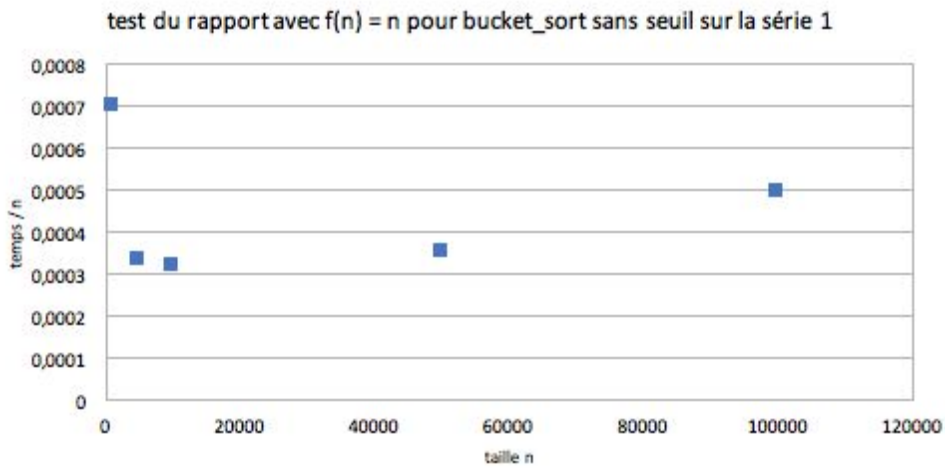


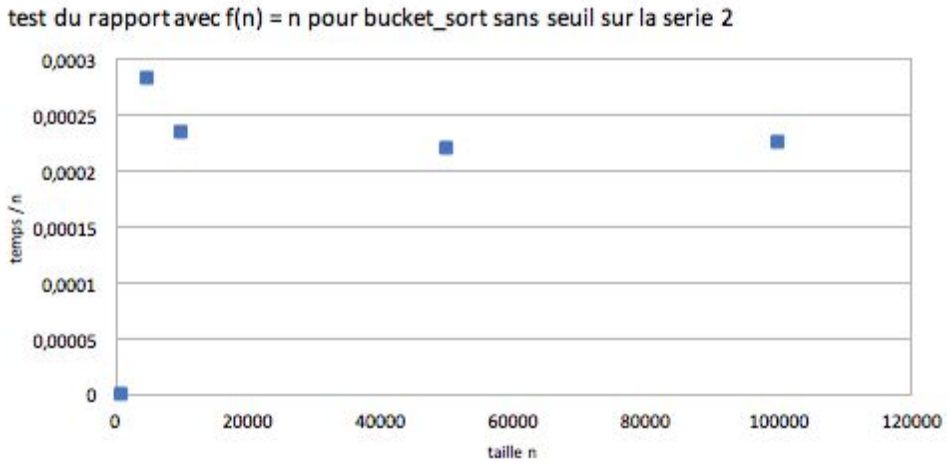


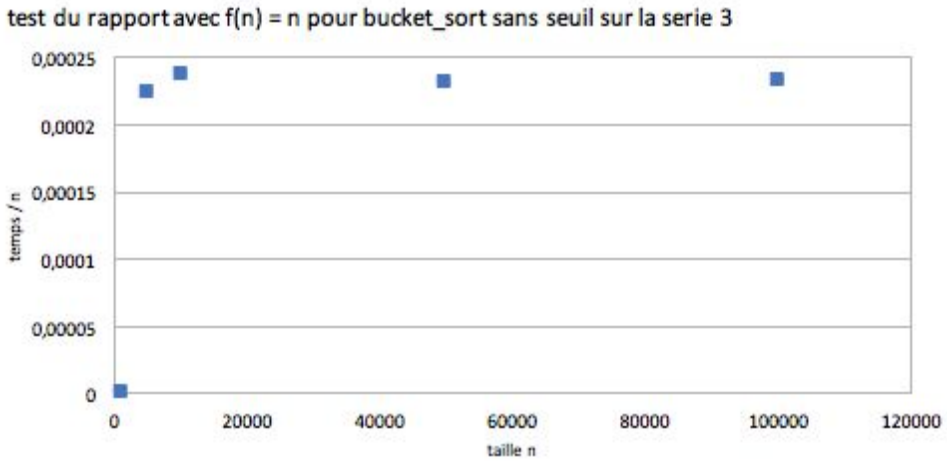
On remarque que pour chacune des séries le rapport Temps / n^2 tend vers lorsque n augmente. On en déduit que “f(n)=n^2” est une surestimation de la complexité de Merge_sort avec un seuil de 100.

Test du rapport pour Bucket_sort sans seuil

Test du rapport pour Bucket_sort sans seuil avec f(n) = n

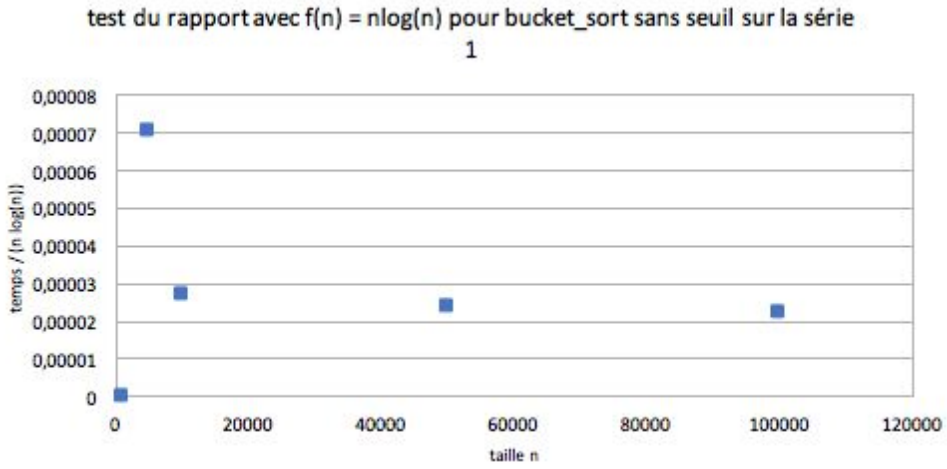


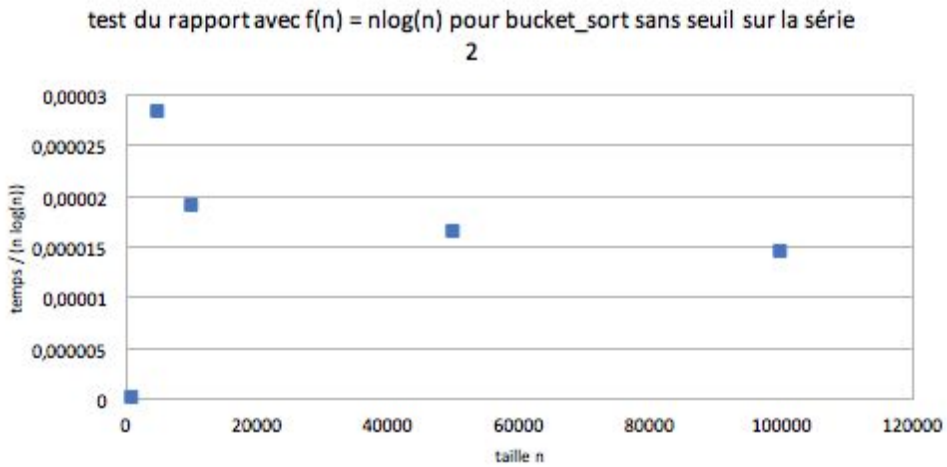


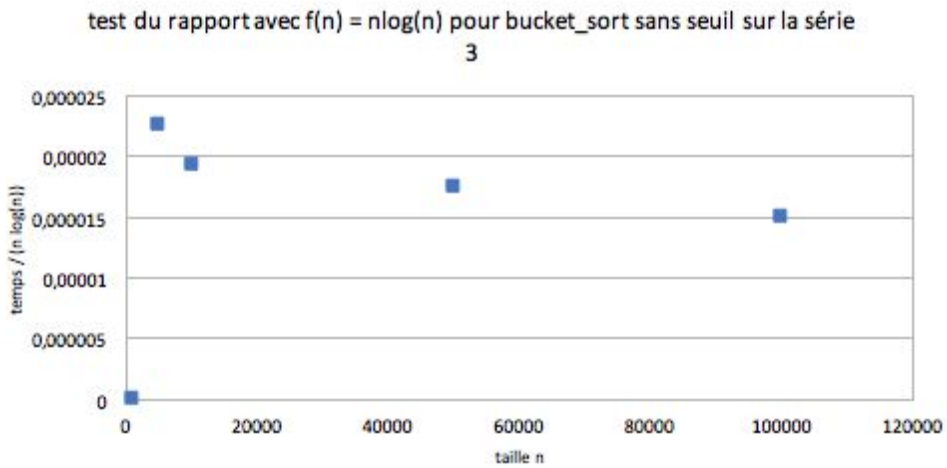


On observe que pour la série 1, le rapport temps / n tend vers l'infini lorsque n augmente. Cependant il semble tendre vers une constante pour les séries 2 et 3. On en déduit que "f(n)=n" est une sous-estimation de la complexité de bucket_sort sans seuil pour des exemplaires du type de ceux de la série 1 alors que la complexité est en n pour des exemplaires de type de ceux des séries 2 et 3.

Test du rapport pour Bucket_sort sans seuil avec f(n) = n*log(n) qui est la complexité en cas moyen et en meilleur cas

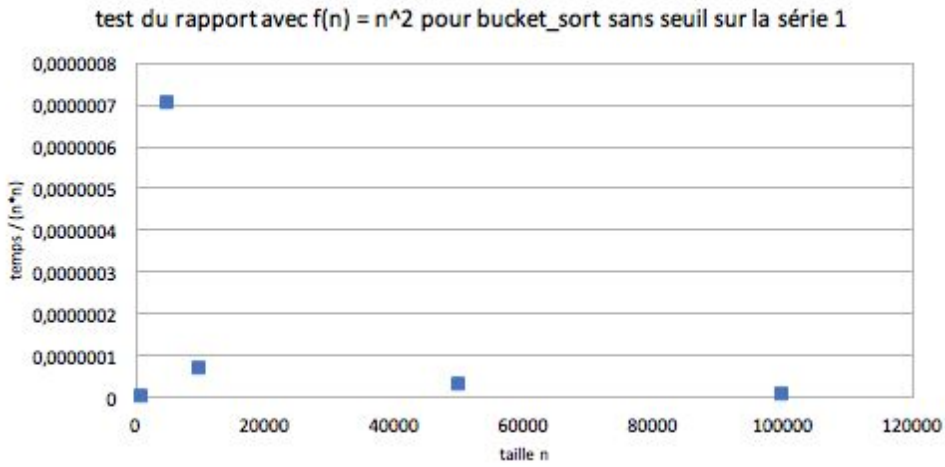


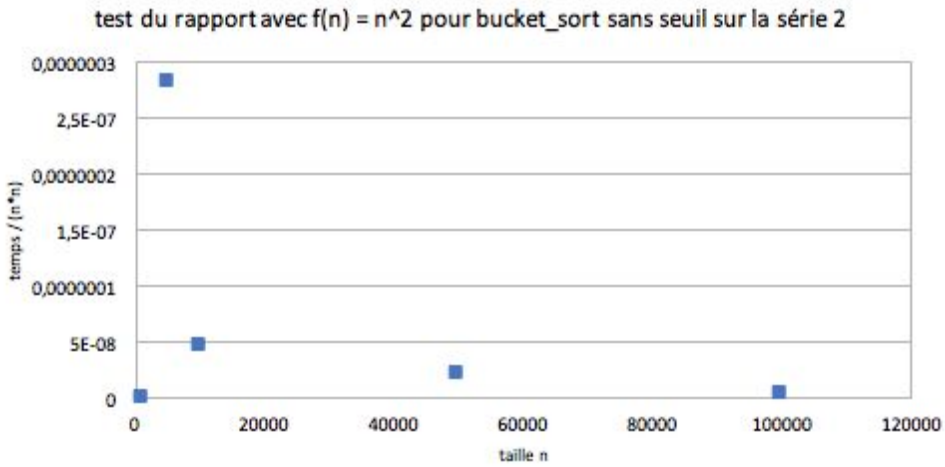


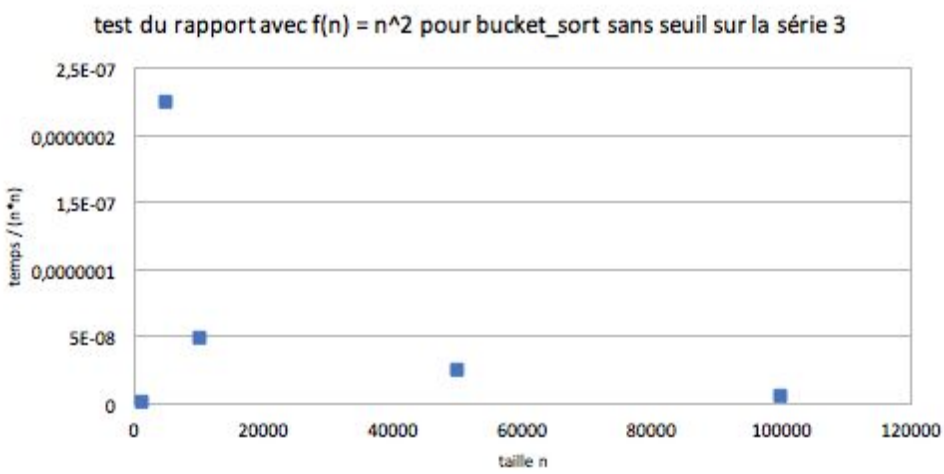


On observe que pour la série 1, le rapport temps / n tend vers une constante lorsque n augmente. Cependant il semble tendre lentement vers 0 pour les séries 2 et 3. On en déduit que “f(n) = n*log(n)” est une légère surestimation de la complexité de bucket_sort sans seuil pour des exemplaires du type de ceux des séries 2 et 3 alors que la complexité est en n*log(n) pour des exemplaires de type de ceux de la série 1.

Test du rapport pour Bucket_sort sans seuil avec “f(n) = n^2” qui est la complexité en pire cas



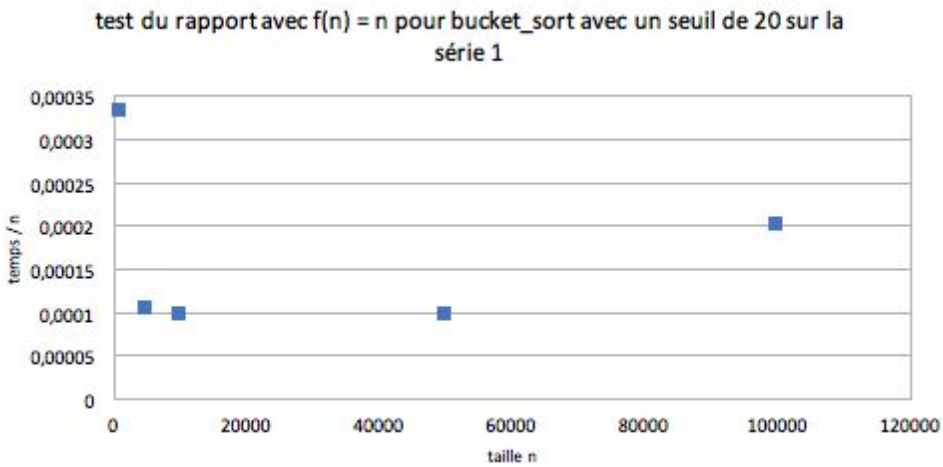


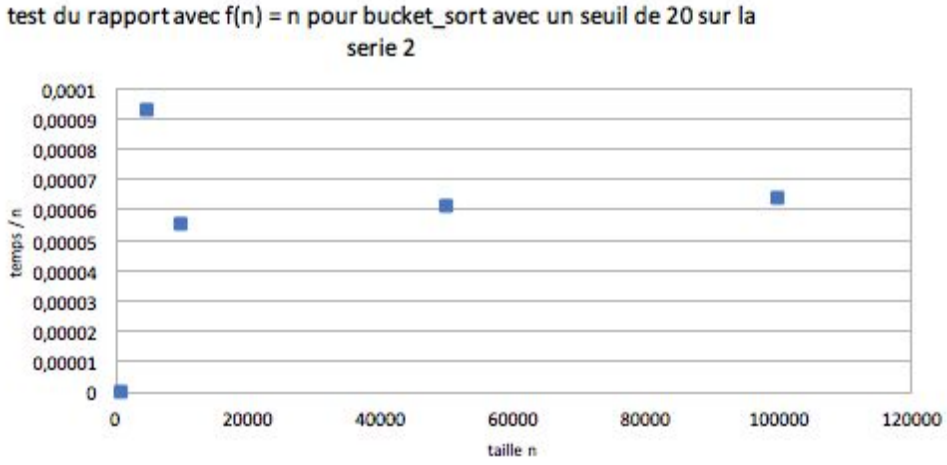


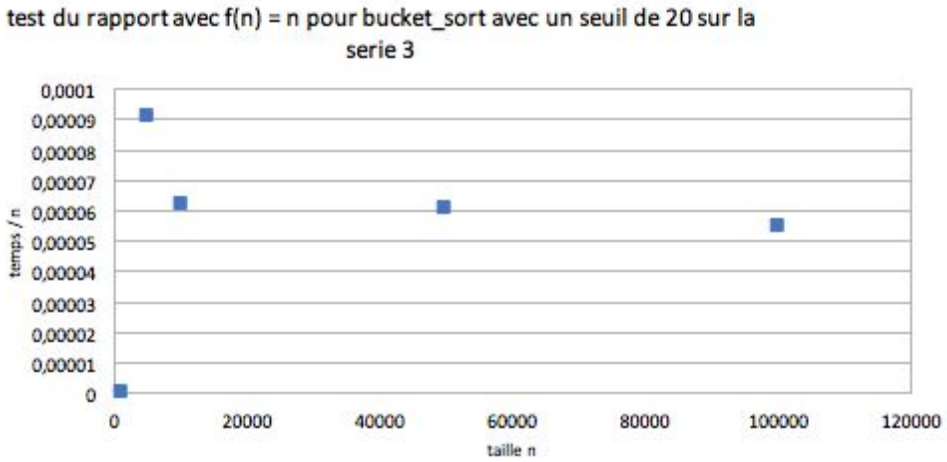
On remarque que pour chacune des séries le rapport Temps / n^2 tend vers l'infini lorsque n augmente. On en déduit que n^2 est une surestimation de la complexité de bucket_sort sans seuil.

Test du rapport pour Bucket_sort avec un seuil de 20

Test du rapport pour Bucket_sort avec un seuil de 20 avec f(n) = n

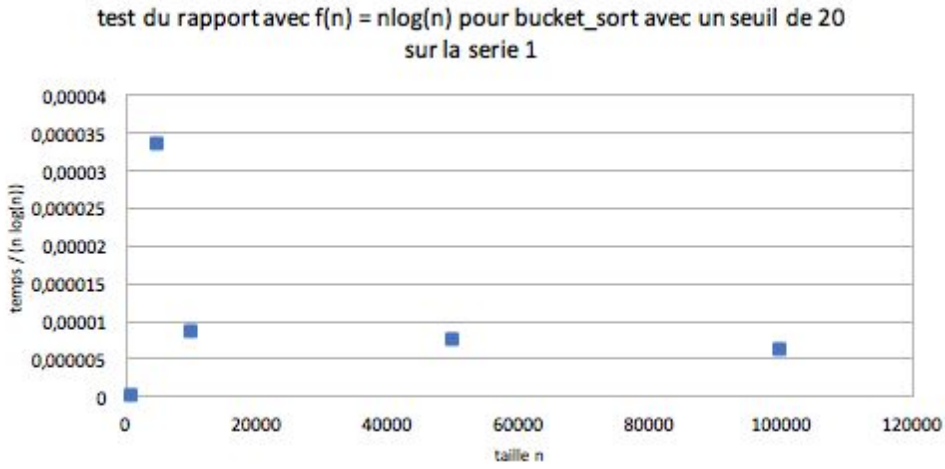


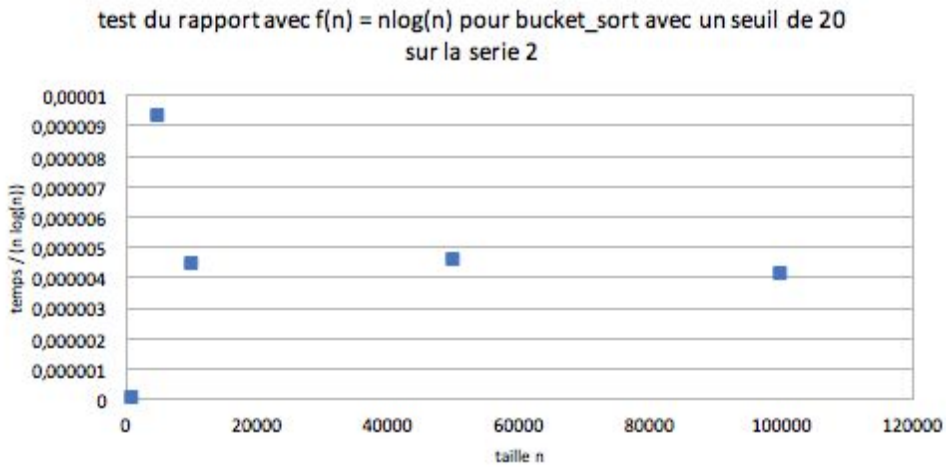


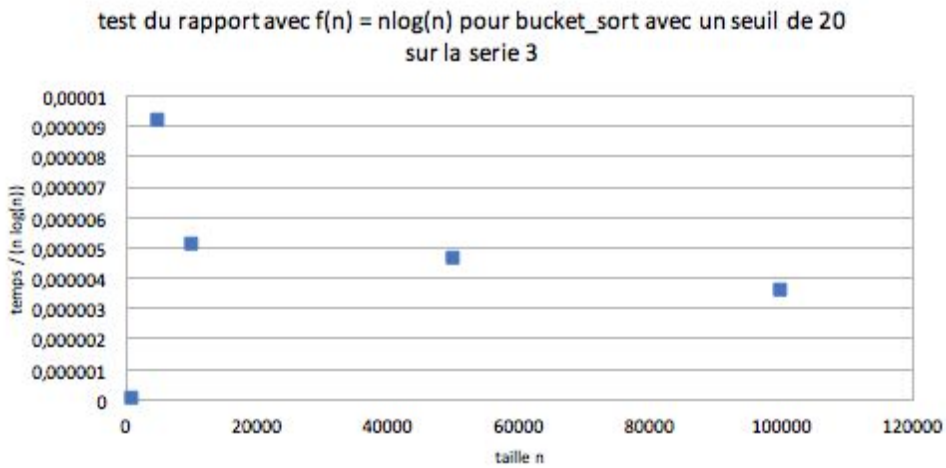


On observe pour les séries 1 et 2 que le rapport temps / n tend vers l'infini. On en déduit que "f(n)=n" est une sous-estimation de bucket_sort avec un seuil de 20.

Test du rapport pour Bucket_sort avec un seuil de 20 avec f(n) = n*log(n) qui est la complexité en cas moyen et en meilleur cas

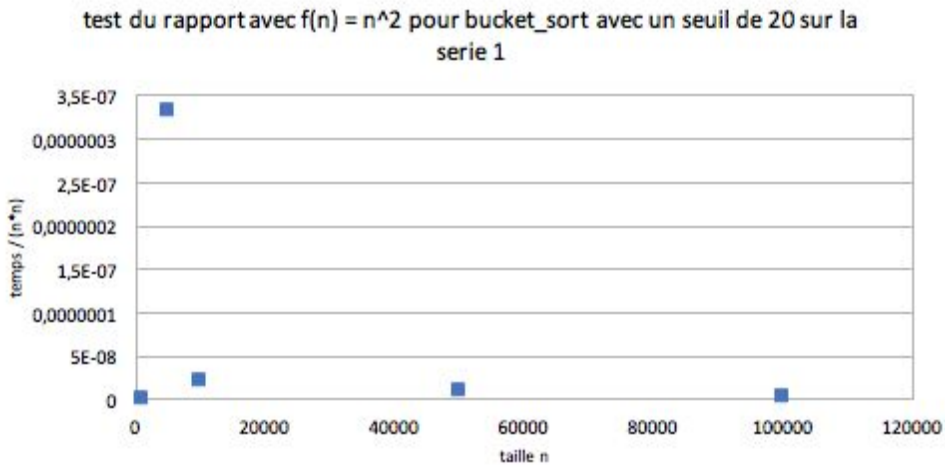


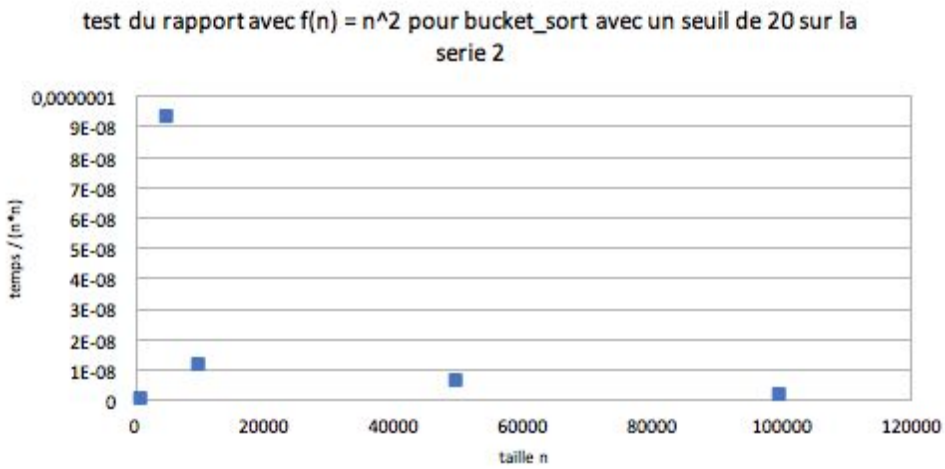


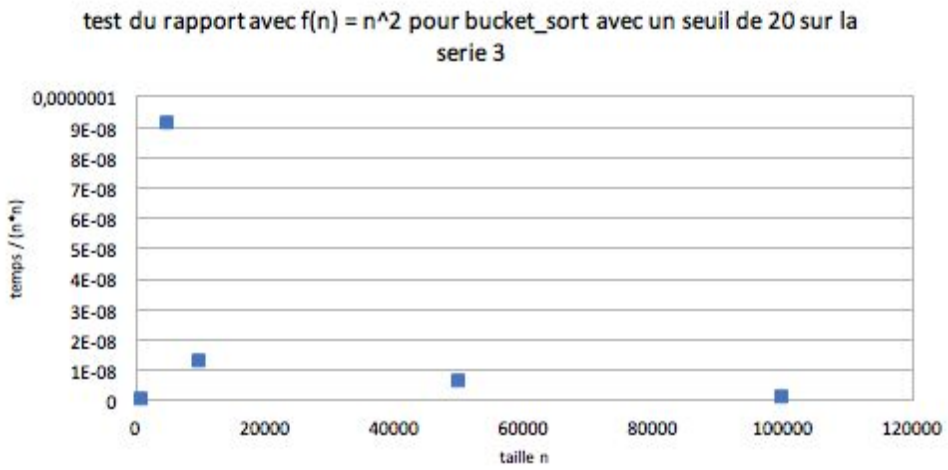


On observe que pour les séries 1 et 2 que le rapport temps / n*log(n) tend vers une constante. On en déduit que la complexité asymptotique de bucket_sort avec un seuil de 20 est de O(n*log(n))

Test du rapport pour Bucket_sort avec un seuil de 20 avec f(n) = n^2 qui est la complexité en pire cas







On remarque que pour chacune des séries le rapport Temps / n^2 tend vers lorsque n augmente. On en déduit que “f(n)=n^2” est une surestimation de la complexité de bucket_sort avec un seuil de 20 seuil.

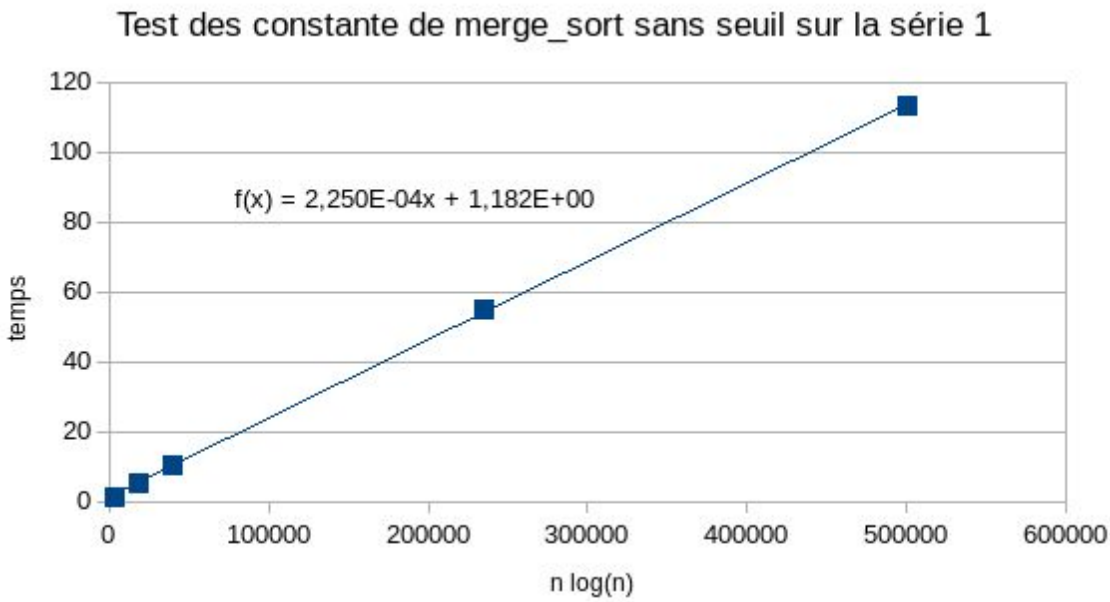
Analyse finale

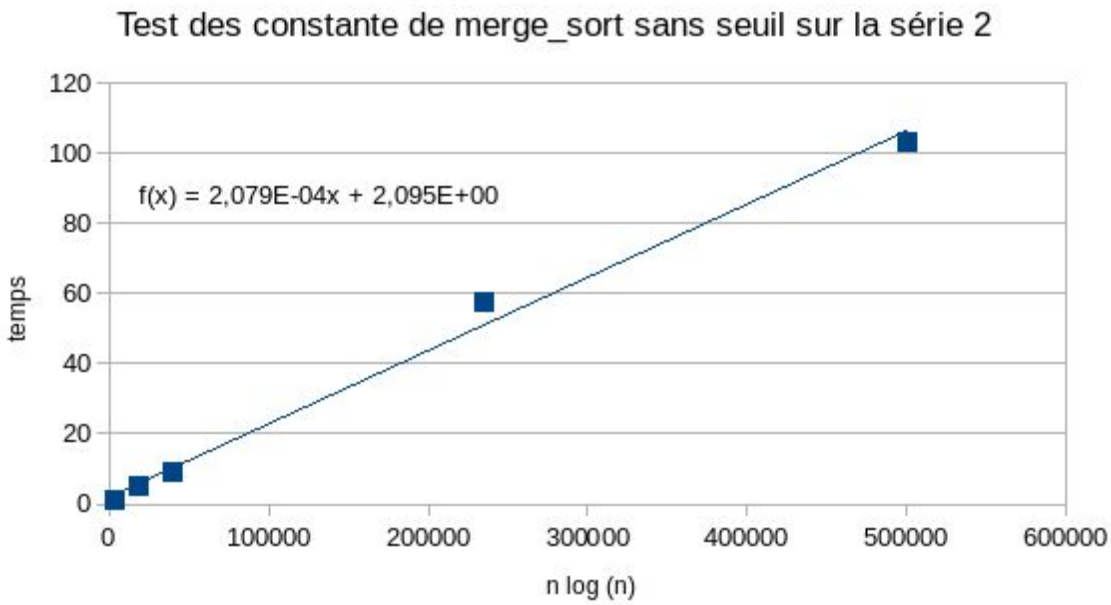
Pour chacun des graphiques précédents, on peut déduire que la complexitié asymptotique est de O(nlogn) pour chacun des algorithmes. En effet, même si certains de nos graphiques ont certains points hors contexte, il est possible de les ignorer et d’arriver à une bonne conclusion.

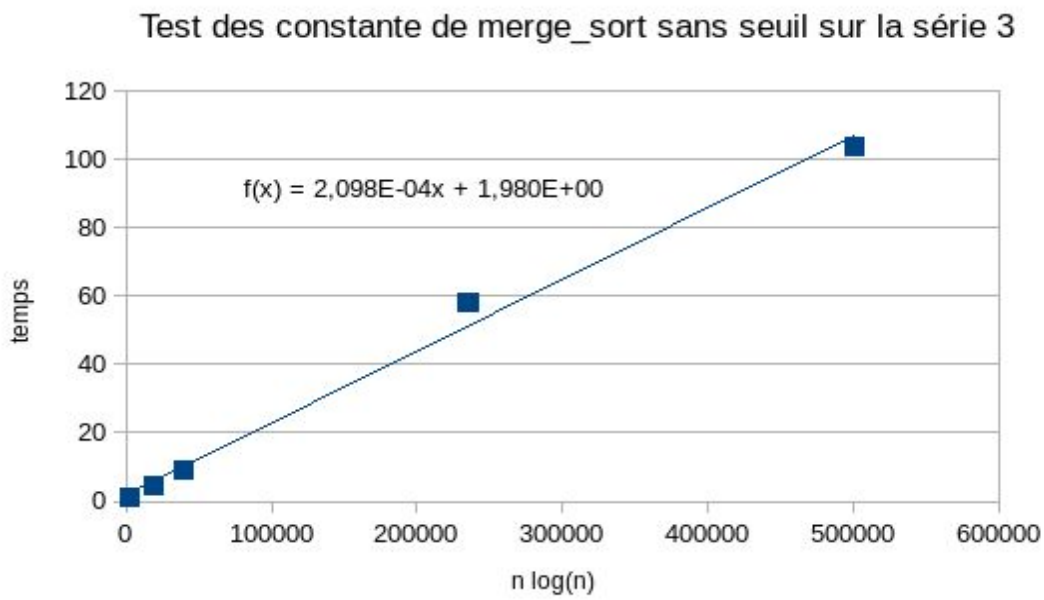
iv) test des constantes

Pour chacun des algorithme nous appliquons le test des constante avec n log(n), n étant la taille des exemplaires.
La complexité des algorithme s’écrit donc sous la forme T(n) = A nlog(n) + C où A et C sont des constantes.

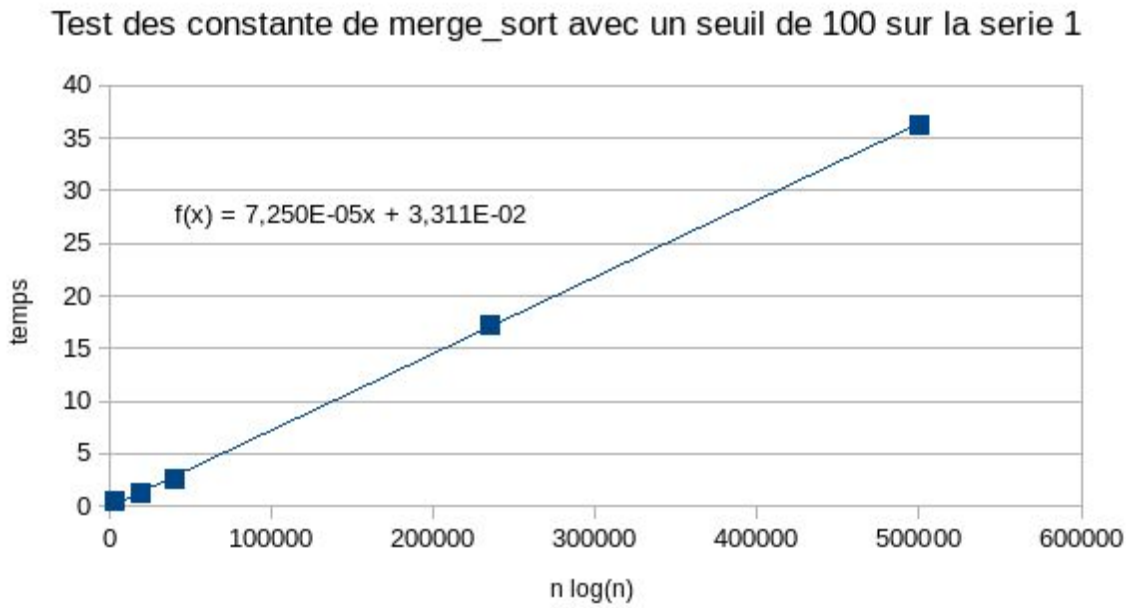
test des constantes pour le Merge_sort sans seuil



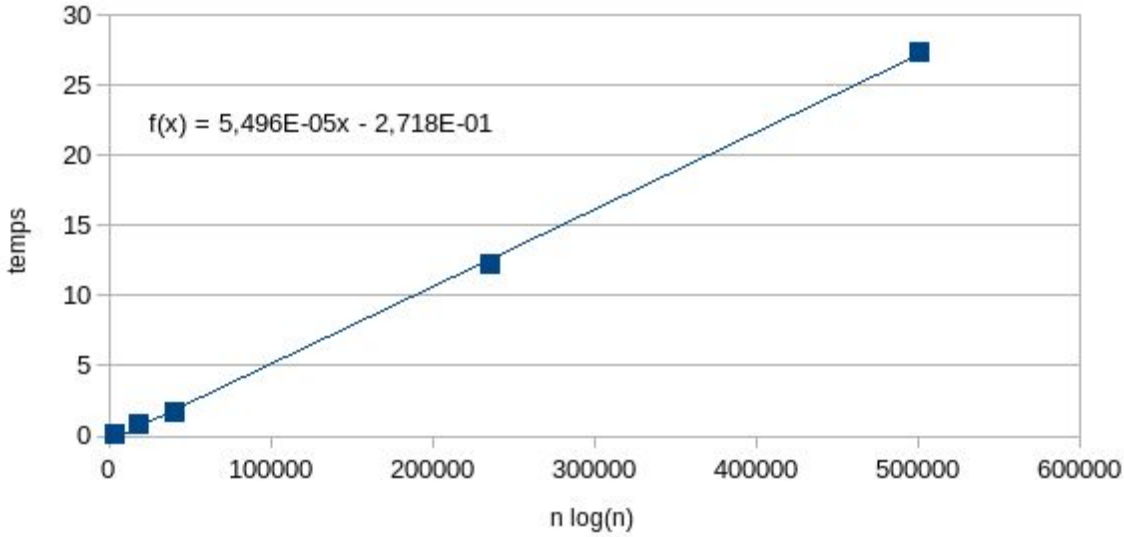




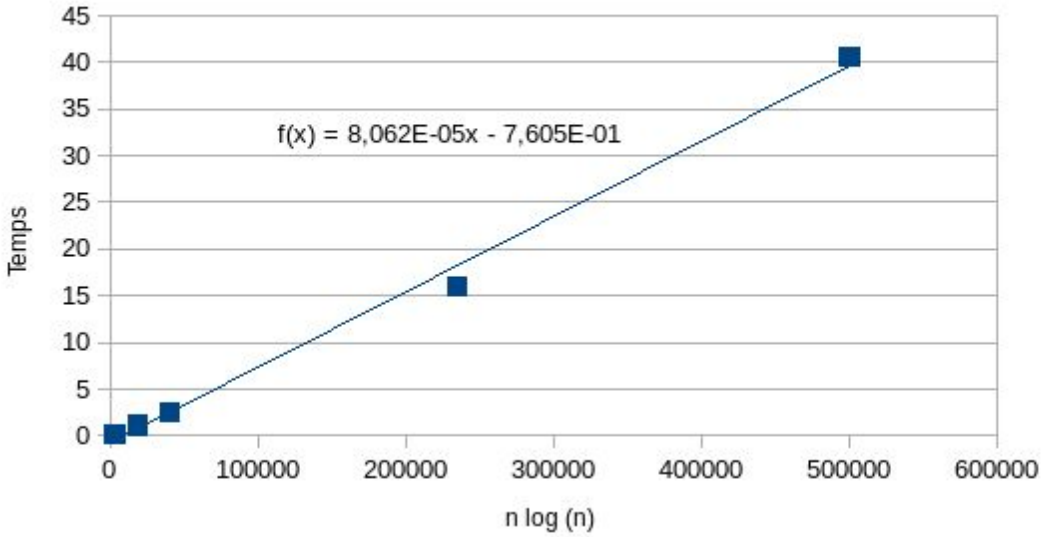
test des constantes pour le Merge_sort avec un seuil de 100



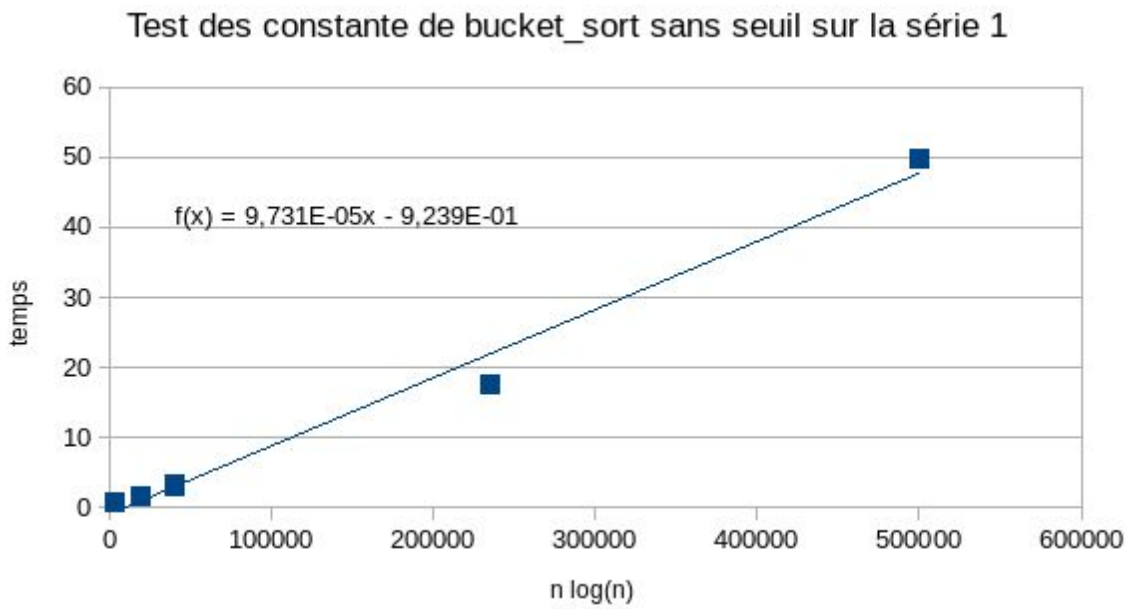
Test des constante de merge_sort avec un seuil de 100 sur la serie 2

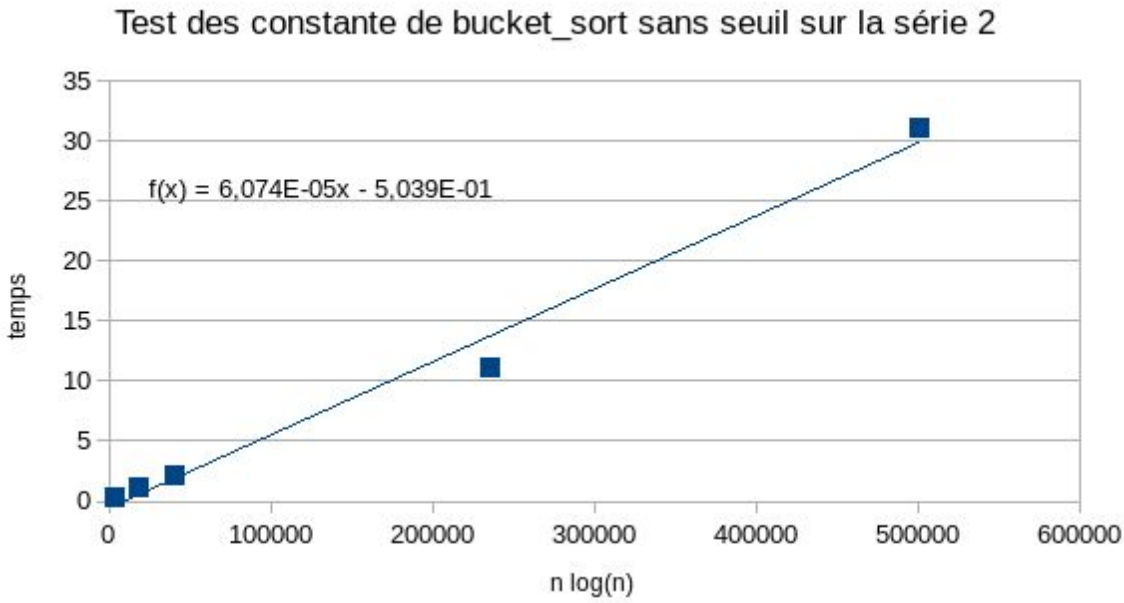


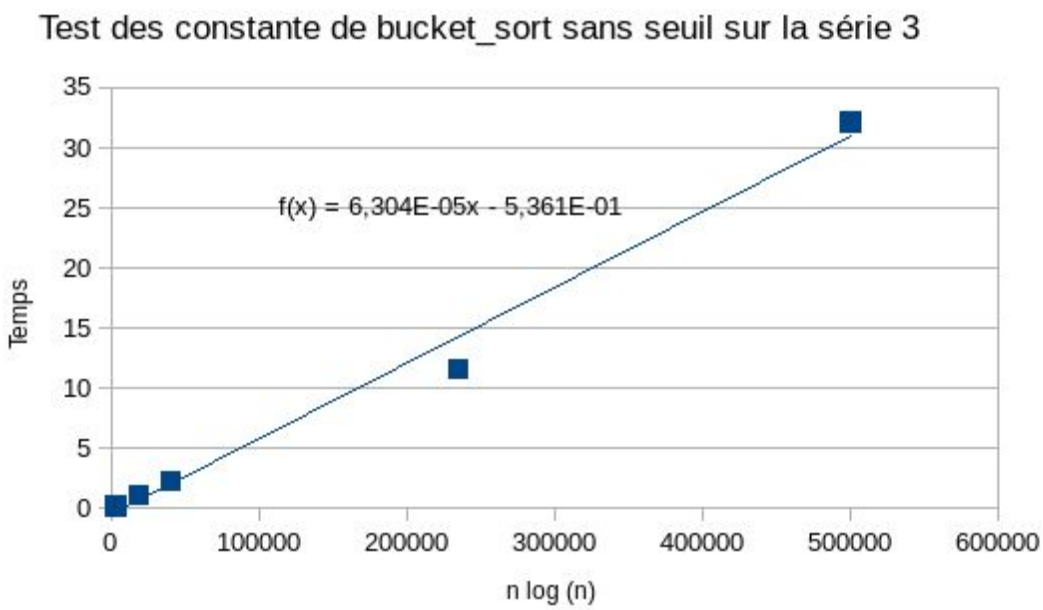
Test des constante de merge_sort avec un seuil de 100 sur la série 3



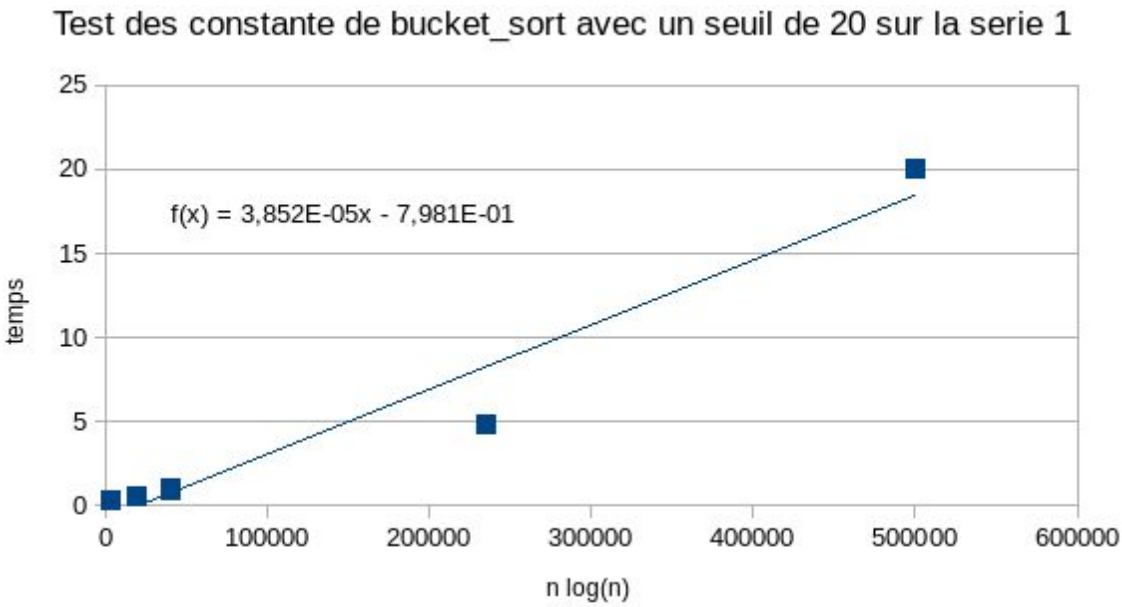
test des constantes pour le Bucket_sort sans seuil

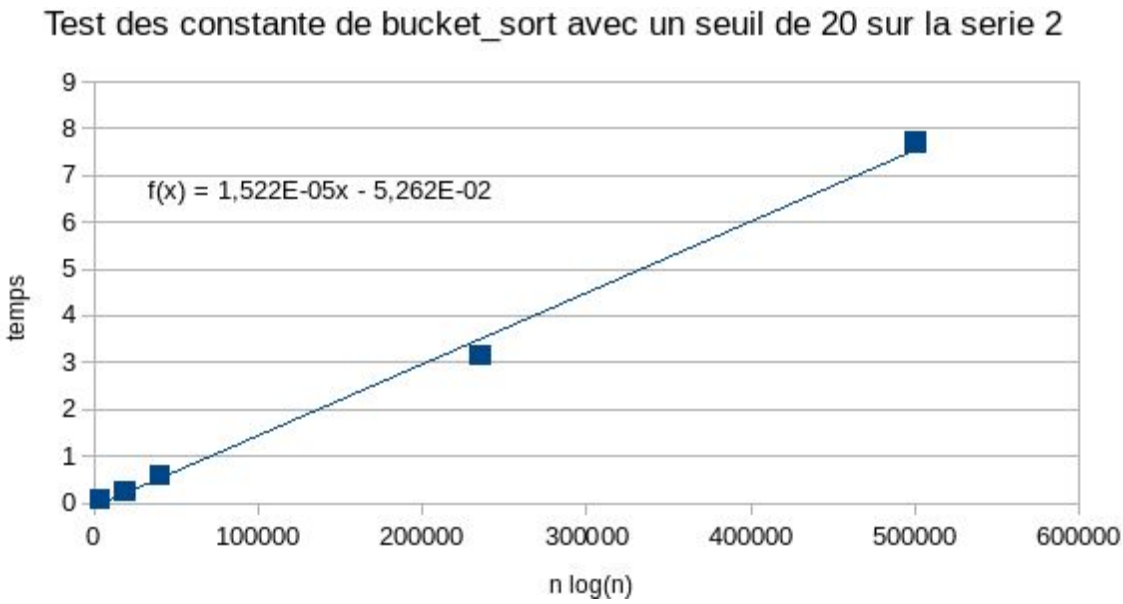


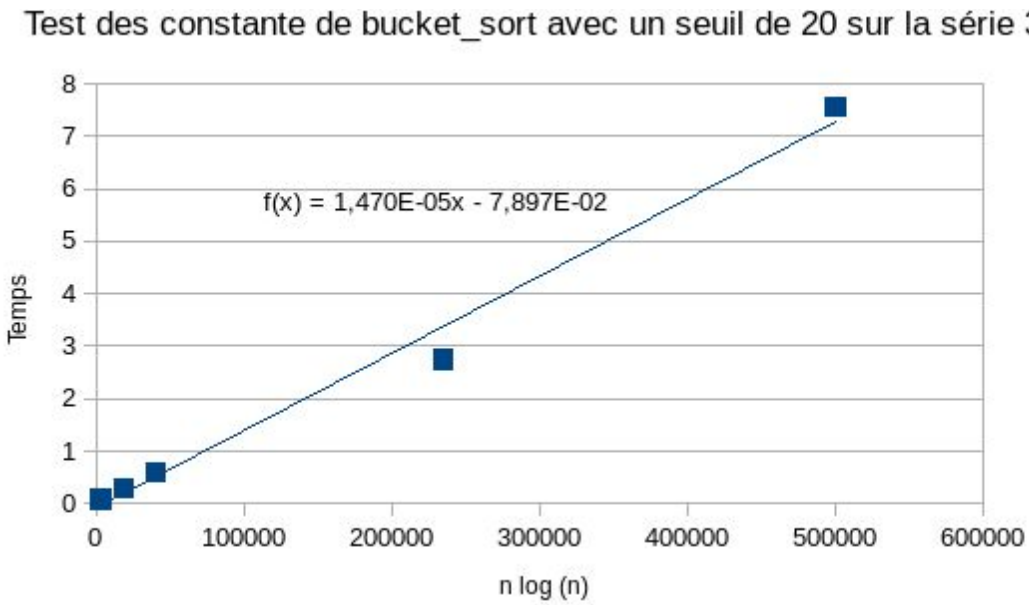




test des constantes pour le Bucket_sort avec un seuil de 20







Analyse

Pour merge_sort sans seuil on trouve un facteur A moyen de 2,14*10^(-4).

Pour merge_sort avec un seuil de 100 on trouve un facteur A moyen de 7*10^(-5).

Pour bucket_sort sans seuil on trouve un facteur A moyen de 7,3*10^(-5).

Pour bucket_sort avec seuil de 20 on trouve un facteur A moyen de 2,25*10^(-5).

On en déduit que pour des exemplaire de grande taille le merge_sort avec un seuil de 100 est 3 fois plus efficace que celui sans seuil.

On en déduit que pour des exemplaire de grande taille le bucket_sort avec un seuil de 20 est 3 fois plus efficace que celui sans seuil.
Par conséquent avoir recours à un seuil peut faire permet de réduire significativement (ici par 3) le temps de calcul.

v) Impact du choix d'un seuil de récursivité

Un bon choix de seuil de récursivité à un grand impact sur la performance de l'algorithme en pratique. En effet, tout dépendamment de la taille et des éléments de l'échantillon, on peut arriver à un point qu'il est rendu inutile de subdiviser le problème et de réutiliser l'algorithme de départ. En effet, rendu à un certain moment dans un algorithme, il est mieux d'arrêter la récursivité et de tout simplement utilisé un algorithme de base itératif comme "Insertion_sort". Bien que cette algorithme ait une complexité théorique plus élevée que l'algorithme initial, elle peut être plus performant. Par exemple, pour le "bucket_sort", en utilisant constamment la récursivité sur les "buckets", cela coûte cher en performance, car il faut initialiser le tableau, créer les "buckets" et ensuite les trier. Pour un petit ensemble, cela peut être inefficace tandis qu'un simple "Insertion_sort" peut régler le problème. Autrement dit, un bon choix de seuil de récursivité peut nous permettra de rapidement augmenter l'efficacité de l'algorithme.

vi) Sous quelles conditions utilisez chacun de ces algorithmes?

Finalement, après tout cette analyse, nous avons remarqué qu'utiliser un algorithme sans seuil peut vite devenir un problème. En effet, comme on l'a vu avec les graphiques, la performance est très affectée lorsqu'il n'y a pas de seuil avec un seuil les algorithmes étaient jusqu'à 3 fois plus performant. Par ailleurs, malgré que les deux algorithmes sont en O(nlogn), il est existe certaines conditions où un est meilleur qu'un autre. Effectivement, le merge_sort est préférable lorsque nous voulons nous assurer d'avoir une complexité d'environ O(nlogn), car il est plus stable. Pour le bucket_sort, il est important de bien choisir la grosseurs des buckets et le nombre de bucket. Ce choix va avoir un impact direct avec la performance de l'algorithme. Un mauvais choix de ces variables peut nous amener au pire cas soit O(n^2). Donc, si il est possible de faire quelques calculs afin de calculer ses deux constantes et de bien les choisirs, alors le bucket_sort peut être une très bonne option. Toujours en regardant notre analyse, on remarque qu'en temps moyen le bucket_sort nous donne une meilleure performance (environ 7/2,25 = 3,11 fois plus performant pour des exemplaires de grande tailles). Ensuite, les éléments de l'ensemble vont avoir un très grand impact sur le bucket_sort. En effet, un ensemble de donnée ayant tous des valeurs très proche peut faire en sorte que nous nous retrouvions avec plusieurs buckets vident. De ce fait, si les données ne sont pas uniformément distribuées à travers nos buckets, cet algorithme peut vite devenir mauvais.