Rapport de projet : Algorithme et Programmation 4.

Sujet : Analyse des algorithmes de tri.

Participants :

- Massiles GHERNAOUT.
- Julien FURET.
- Clément FLAMBARD.

Description du projet :

Les algorithmes de tri sont des méthodes pour classer des données dans un ordre défini. Il existe plusieurs algorithmes de tri différents, chacun ayant ses propres avantages et inconvénients. Dans ce rapport, nous nous concentrerons sur trois algorithmes de tri :

- le tri à bulles.
- le tri rapide.
- le tri par dénombrement.

Afin de comparer les différents algorithmes de tri, on utilise la complexité temporelle:

- La complexité temporelle est obtenue à partir d'une fonction réelle qui prend en entrée la taille de l'ensemble à trier et retourne la durée nécessaire à intégrer la notion d'ordre voulue dans celui-ci.

L'objectif du projet est de comparer 3 algorithmes de tri afin de déterminer qui est le plus efficace en se basant sur la complexité énoncée ci-dessus, mais également sur le nombre total d'opérations élémentaires.

Ces résultats doivent être présentés de façon claire et précise, c'est pour cette raison que nous avons choisi de les représenter sous la forme de graphiques en courbes.

Analyse du tri à bulles :

Description :

Le tri à bulles est un tri classique qui permet de réaliser un ordre dans un ensemble de données munies d'une clé de comparaison. Le tri se fait principalement par des échanges entre deux éléments successifs.

Le principe, c'est de faire des comparaisons entre ces éléments dans le but de remonter les plus grands à la fin. L'appellation est inspirée d'une analogie vis-à-vis des bulles d'air qui remontent dans un tuyau rempli d'eau.

Notre implémentation est optimisée en deux temps :

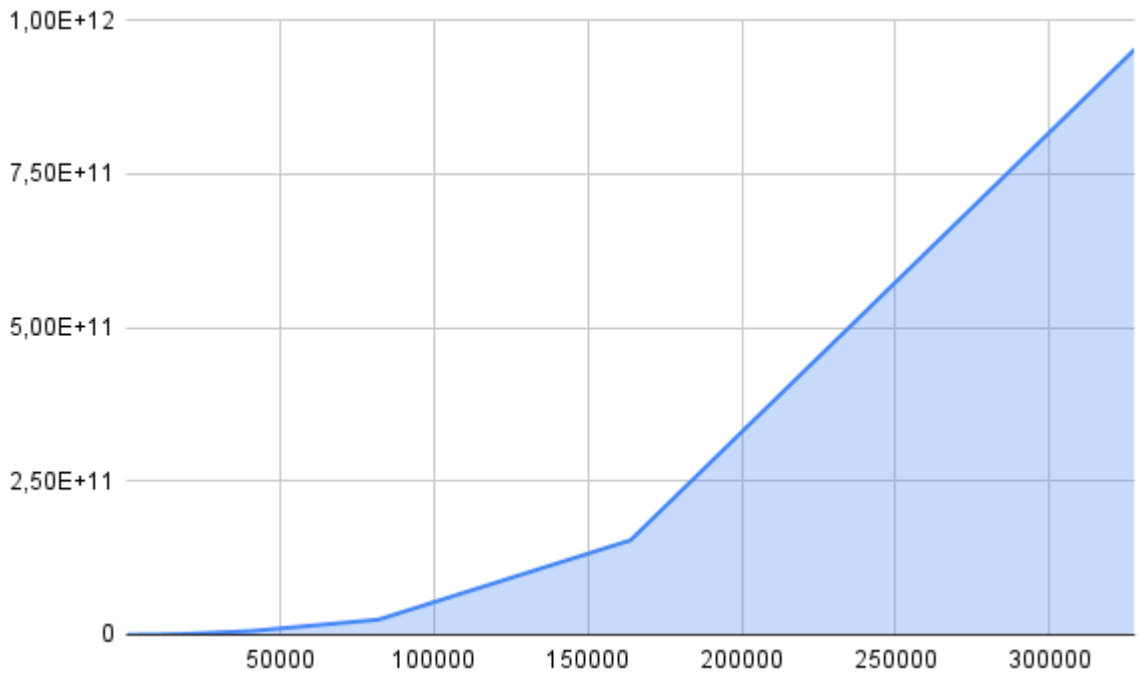
En premier lieu :

On garde en mémoire, à l'aide d'une variable, si il y a eu un changement dans l'itération précédente, si ce n'est pas le cas, alors le tableau est trié et on arrête l'algorithme.

En deuxième lieu :

On garde en mémoire, à l'aide d'une variable, l'index (la position) du dernier changement dans l'ensemble d'éléments, et adapte la boucle afin que celle-ci s'arrête à cet index-ci lors de l'itération suivante.

Courbe d'évolution du temps en fonction de la taille de l'ensemble d'éléments :



Commentaires :

On voit que la courbe d'évolution du temps en fonction de la taille d'éléments est quadratique ***[f(n) = n²]***, ce qui est conforme à la complexité théorique du tri à bulles.

La complexité obtenue dans la courbe est la complexité moyenne (sur un ensemble de données évolutif).

Toutefois, la complexité au pire est aussi importante, et pour le tri à bulle, elle est retrouvée quand le tableau est trié dans le sens inverse au tri visé. Dans ce cas, les optimisations implémentées perdent leurs sens, et on aura un nombre d'itérations égal à ***n * (n - 1),*** ou ***n*** est le nombre d'éléments dans notre ensemble.

Analyse des opérations élémentaires :

Suivant notre implémentation, dans le pire cas, le tri à bulle réalise des comparaisons entre tous les éléments à chaque itération, et on a ***n * (n - 1)*** itérations. Or, il compare les éléments deux à deux, du coup, le nombre de comparaisons est égale à ***n * (n - 1) / 2*** .

Aussi, pour chaque comparaison, il effectuera 5 opérations élémentaires, de type affectation et indexage d'un tableau, qui est notre ensemble.

Enfin, nous nous sommes concentrés sur le pire cas, étant donné que c'est le seul cas où le nombre d'itérations est connu à l'avance.

Analyse du tri rapide :

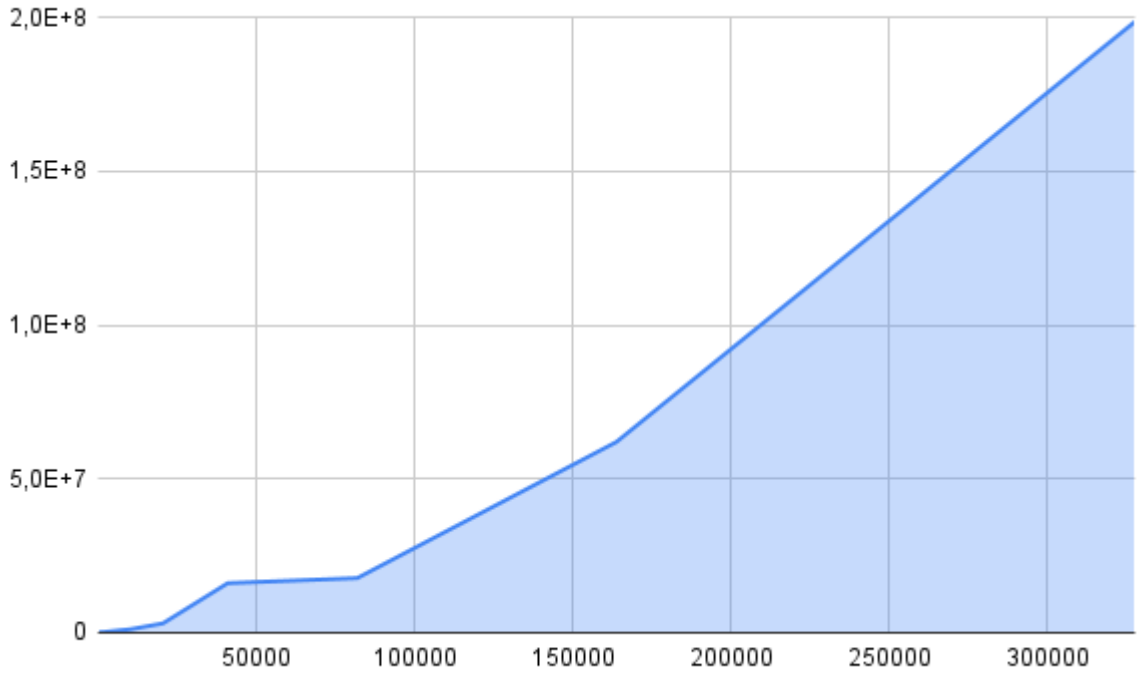
Description :

Le tri rapide (aussi appelé tri par pivot ou bien par partitions) est un algorithme de tri qui fonctionne en sélectionnant un élément pivot et en partitionnant les éléments autour de ce pivot.

Le but est de placer le pivot en position telle que tous les éléments à gauche soient inférieurs, et tous les éléments à droite soient supérieurs. Ce processus est répété récursivement pour les sous-ensemble gauche et droit, appelés partitions, jusqu'à ce que chaque sous-ensemble ne contienne plus qu'un seul élément, qui le rend trié de nature.

Analyse de complexité temporelle - time complexity :

Courbe d'évolution du temps en fonction de la taille de l'ensemble d'éléments :



Commentaires :

On voit que la courbe d'évolution du temps en fonction de la taille d'éléments est logarithmique ***[f(n) = n*log(n)]***, ce qui est conforme à la complexité théorique du tri rapide.

La complexité obtenue dans la courbe est la complexité moyenne (sur un ensemble de données évolutif).

Toutefois, la complexité au pire est aussi importante, et pour le tri rapide, le pire cas dépend du choix du pivot.

Analyse des opérations élémentaires :

Soit ***n,*** notre nombre d'éléments.

Suivant notre implémentation, où on prend le premier élément comme pivot, le pire cas c'est quand le tri est appliqué sur un tableau trié dans le sens décroissant et on veut avoir le sens croissant.

Par conséquent, le pivot est l'élément maximum de notre ensemble. Donc, à chaque appel récursif, à chaque partition / sous-ensemble, nous devons prendre tous les éléments à gauche et les mettre à droite, et vice versa. Ceci, nous force à effectuer un parcours linéaire complet pour chaque partition.

Étant donné que l'algorithme est du type ***"Divide & Conquer"*** , on aura ***log(n)*** partitions.

En conclusion, cela nous fait un nombre de parcours linéaire égale à ***n * log(n) .***

Aussi, durant chaque parcours, nous effectuerons une comparaison suivie par 4 opérations élémentaires du type: affectation, et indexage de notre tableau, qui est notre ensemble.

À la fin de chaque parcours, nous aurons aussi un ajustement, pour obtenir le prochain pivot qui consiste à faire 4 opérations élémentaires supplémentaires, de même nature.

Enfin, nous nous sommes concentrés sur le pire cas, étant donné que c'est le seul cas où le nombre d'itération et de partitions crées est connu à l'avance.

Analyse du tri par dénombrement :

Description :

Le tri par dénombrement est un tri qui fonctionne uniquement avec des clés de comparaison entières, il utilisera un tableau d'occurrences d'une taille égale à la valeur du plus grand élément du tableau d'entrée (le tableau d'entrée est notre ensemble de données à trier).

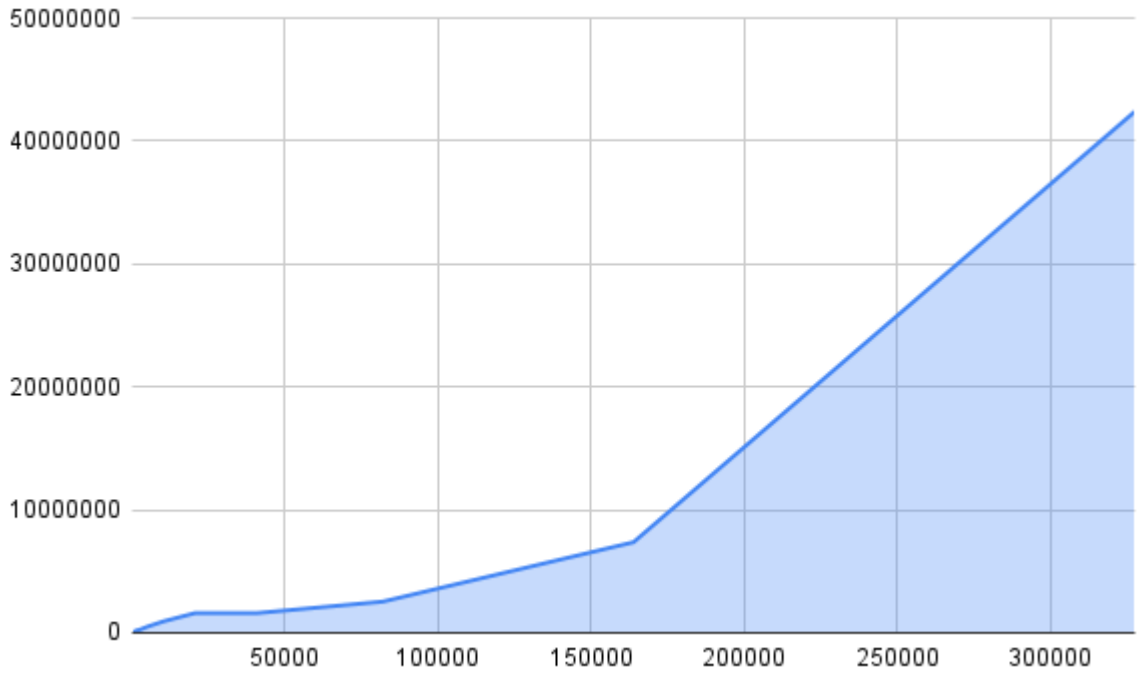
Ensuite, l'objectif va être de parcourir le tableau d'entrée et de compter le nombre de fois où chaque élément apparaît en incrémentant la valeur du tableau d'occurrences stockée à l'indice correspondant à l'élément en question.

Par la suite, avec le tableau d'occurrences rempli, on va faire la somme cumulative, c'est-à-dire qu'on va additionner le nombre d'occurrences de chaque élément avec le nombre d'occurrences de l'élément précédent. Ainsi, cela va nous permettre de déterminer l'indice de départ de chaque élément dans le tableau trié. Pour ce qui suit, nous ferons référence à ce concept par le terme "complémentation / complémenter" du tableau d'occurrences

Pour finir, on va parcourir une dernière fois le tableau d'occurrences et pour chaque élément, on va prendre sa valeur puis la somme cumulative pour trouver sa position dans le tableau trié avant de l'insérer.

Analyse de complexité temporelle - time complexity :

Courbe d'évolution du temps en fonction de la taille de l'ensemble d'éléments :



Commentaires :

On voit que la courbe d'évolution du temps en fonction de la taille d'éléments est linéaire, c'est-à-dire qu'elle est proportionnelle à la taille du tableau d'entrée et d'occurrences. Elle est de la forme ***[f(n) = n + k]***, ce qui est conforme à la complexité théorique du tri par dénombrement.

La complexité obtenue dans la courbe est la complexité moyenne (sur un ensemble de données évolutif).

De plus, quelques avantages de cet algorithme sont la stabilité et le maintien de la complexité linéaire.

Analyse des opérations élémentaires :

Soit ***n*** notre nombre d'éléments,

Nous devons faire ***n - 1*** de comparaisons pour trouver le max, et cela à l'aide d'un parcours linéaire sur notre ensemble de données.

Soit ***k*** la valeur du max.

Nous devons créer notre tableau d'occurrences de taille ***k.*** Ensuite, nous allons parcourir ce tableau deux fois afin de l'initialiser et puis le complémenter.

Puis, nous allons créer notre tableau résultat de taille ***n.***

Enfin, nous devons parcourir notre tableau d'entrée dans le sens inverse, qui nous permettra de rendre le tri stable. Pour chaque itération, nous allons effectuer une lecture depuis le tableau d'occurrences suivie d'une écriture vers le tableau résultat.

Par conséquent, nous avons :

- deux parcours linéaires sur notre tableau d'entrée, qui fait une complexité de l'ordre de ***O(2n) ~ O(n).***
- deux parcours linéaires sur notre tableau d'occurrences, qui fait une complexité de l'ordre de ***O(2k) ~ O(k).***

Etant donné que les parcours ne sont pas imbriqués, les complexités s'additionnent. On obtient une complexité totale de ***O(n+k),*** ce qui confirme encore une fois la complexité théorique de cet algorithme.

Conclusion:

En conclusion, les algorithmes de tri sont des outils importants pour la gestion de données, et il est important de comprendre les différentes approches et leur complexité pour choisir le meilleur algorithme pour chaque situation. Le tri rapide, le tri par dénombrement et le tri à bulles sont tous des algorithmes valables, chacun ayant ses propres avantages et inconvénients. En fin de compte, le choix dépendra des besoins spécifiques de chaque cas d'utilisation.