LES ALGORITHMES DE TRIS - PERFORMANCE

SMAIL Mustapha

Université de Paris Nanterre

N° Étudiant: 40016261

mustapha.smail@parisnanterre.fr

Introduction

Un algorithme de tri permet d'organiser une collection d'objets selon une relation d'ordre total déterminée.

Les algorithmes de tri sont utilisés dans de très nombreuses situations. Ils constituent une brique de base utilisée pour de nombreux algorithmes avancés. La majorité des algorithmes de tri usuels procédant par comparaisons successives, et peuvent donc être définis indépendamment de l'ensemble auquel appartiennent les éléments et de la relation d'ordre associée. Un même algorithme peut être utilisé pour trier des entiers, des réels, des chaînes de caractères.

L’objectif de ce projet est de comparer les temps d’exécution des algorithmes de tris, en utilisant différents types de tableaux d’entiers, de différentes tailles.

Donnés

Dans le fichier **tableaux.c**, un algorithme permet de générer différents types de tableaux utilisés ensuite dans le projet.

Les types de tableaux utilisés sont:

- Des tableaux aléatoires
- Des tableaux triés par ordre croissant
- Des tableaux triés par ordre décroissant
- Des tableaux contenants une valeur constante

Les tailles des tableaux utilisés vont de 10 000 entiers à 100 000 entiers, avec un écart de 10 000.

Algorithmes

Ci dessous, les différents algorithmes de tris utilisés dans ce projet:

1. Tri à bulle - Bubble sort

Il consiste à comparer répétitivement les éléments consécutifs d'un tableau, et à les permuter lorsqu'ils sont mal triés.

Complexité en temps:

Pire cas : $O(n^2)$

Moyenne: $O(n^2)$

Meilleur cas: O(n)

2. Tri fusion

Pour une entrée donnée, l'algorithme la divise en deux parties de tailles similaires, trie chacune d'entre elles en utilisant le même algorithme, puis fusionne les deux parties triées

Complexité en temps:

Pire cas : O(n log n)

Moyenne: O(n log n)

Meilleur cas: O(n log n)

3. Tri rapide - Quick sort

Une valeur est choisie comme pivot et les éléments plus petits que le pivot sont dissociés, par échanges successifs, des éléments plus grands que le pivot ; chacun de ces deux sous-ensembles est ensuite trié de la même manière.

Complexité en temps:

Pire cas : $O(n^2)$

Moyenne: $O(n \log n)$

Meilleur cas: $O(n \log n)$

4. Tri par insertion

Le tri par insertion considère chaque élément du tableau et l'insère à la bonne place parmi les éléments déjà triés. Ainsi, au moment où on considère un élément, les éléments qui le précèdent sont déjà triés, tandis que les éléments qui le suivent ne sont pas encore triés.

Complexité en temps:

Pire cas : $O(n^2)$

Moyenne: $O(n^2)$

Meilleur cas: $O(n)$

5. Tri par sélection

le principe du tri par sélection est le suivant :

rechercher le plus petit élément du tableau, et l'échanger avec l'élément d'indice 0 ;

rechercher le second plus petit élément du tableau, et l'échanger avec l'élément d'indice 1 ;

continuer de cette façon jusqu'à ce que le tableau soit entièrement trié.

Complexité en temps:

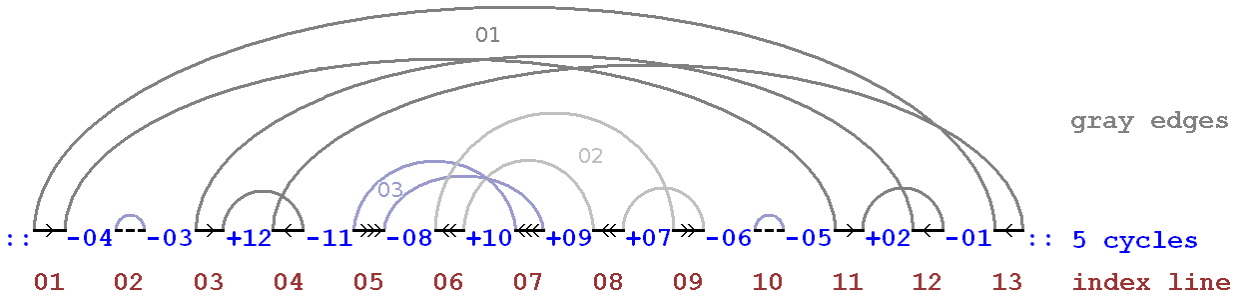
Pire cas : $O(n^2)$

Moyenne: $O(n^2)$

Meilleur cas: $O(n^2)$

6. Tri par cycle

Il est basé sur l'idée de permutation dans laquelle les permutations sont prises en compte dans les cycles, qui font individuellement pivoter et renvoient une sortie triée.



Complexité en temps:

Pire cas : $O(n^2)$

Moyenne: $O(n^2)$

Meilleur cas: $O(n^2)$

7. Tri gnome

Le tri gnome est un algorithme de tri qui est similaire au tri par insertion en ce sens qu'il fonctionne avec un élément à la fois mais place l'élément au bon endroit par une série d'échanges, similaire à un tri à bulles. L'algorithme trouve le premier endroit où deux éléments adjacents sont dans le mauvais ordre et les permute.

Complexité en temps:

Pire cas : $O(n^2)$

Moyenne: $O(n)$

Meilleur cas: $O(n^2)$

8. Tri cocktail

La différence entre cet algorithme et le tri à bulles est qu'il exécute un tri dans chaque direction à chaque passe le long de la liste à trier

Complexité en temps:

Pire cas : $O(n^2)$

Moyenne: $O(n^2)$

Meilleur cas: $O(n)$

9. Tri crêpes

Il s'agit de trier une pile de crêpes afin que les crêpes soient empilées de la plus grande à la plus petite (au sens de leur diamètre).

Complexité en temps:

Pire cas : $O(n^2)$

Moyenne: $O(n^2)$

Meilleur cas: $O(n)$

Automatisation de la comparaison

Dans cette étape, j'ai réalisé un algorithme permettant de parcourir la liste des algorithmes de tris disponibles, de mesurer leur temps d'exécution, enregistrer les résultats dans un fichier csv, ensuite lire les données de ce dernier afin de générer un graphe en utilisant **gnuplot**.

Les méthodes utilisées sont les suivantes:

1. Struct functions:

Un tableau de struct functions, qui est une structure contenant le nom de l'algorithme de tri et un pointeur vers sa fonction.

2. Copie tableau

Permet de faire une copie d'un tableau donné en argument.

3. Temps

Renvoie le temps d'exécution d'un algorithme de tri passé en paramètre.

4. Comparer

Cette méthode prend en paramètre un nom de fichier (le fichier csv) et une fonction récupérée dans **tableaux.c,** permettant de générer un tableau d'un certain type.

On commence par ouvrir ce fichier en mode écriture+ (si le fichier n'existe pas, il sera créé, si le fichier contient des informations, elles seront supprimées)

On écrit ensuite, dans le première ligne, les noms des algorithmes de tris.

Un boucle while permet d'exécuter la comparaison des algorithmes sur le type du tableau choisi, avec une taille allant de 10 000 à 100 000, par écart de 10 000.

Les temps d'exécution sont écrits dans ce fichier.

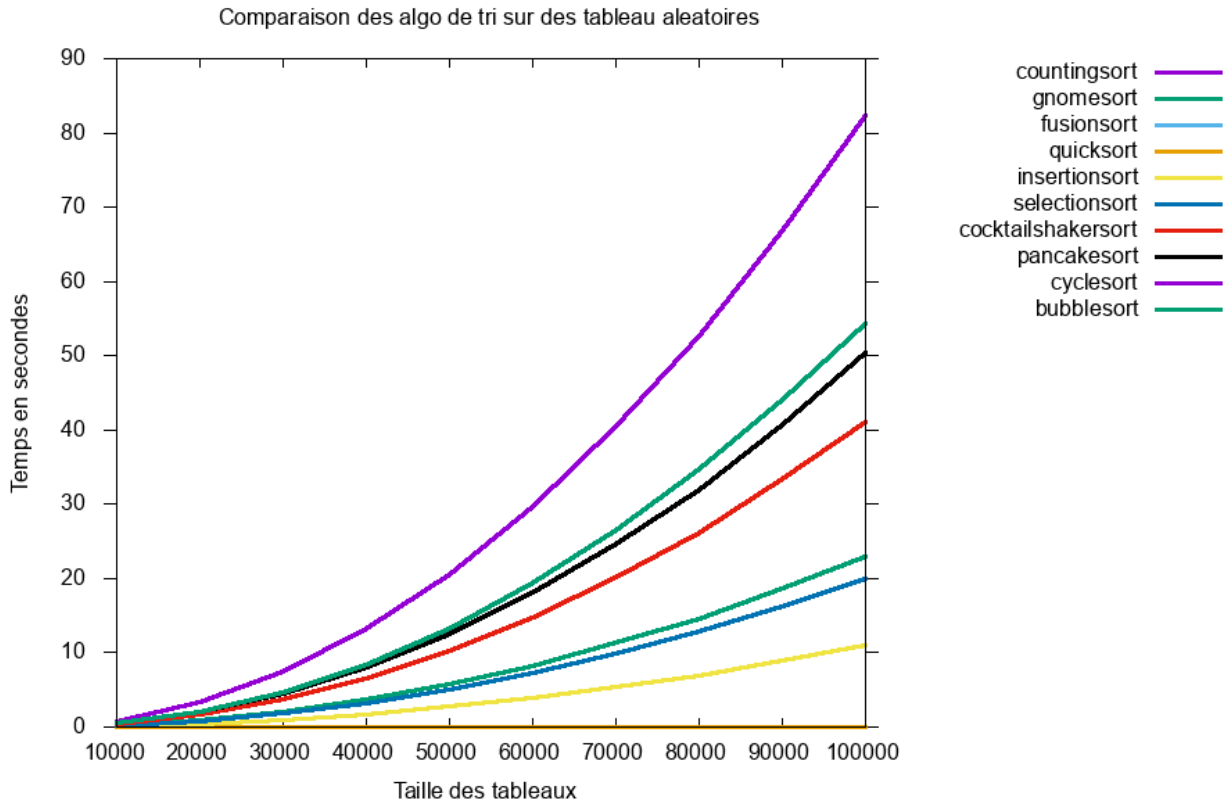
5. Générer graphes

Une méthode qui prend en arguments, un nom fichier, et un nom d'image.

Cette fonction exécute dans le terminal des commandes **gnuplot**, permettant de générer les graphiques reliés pour chaque fichier donné.

Les graphiques

1. Graphiques des tableaux aléatoires



Remarques

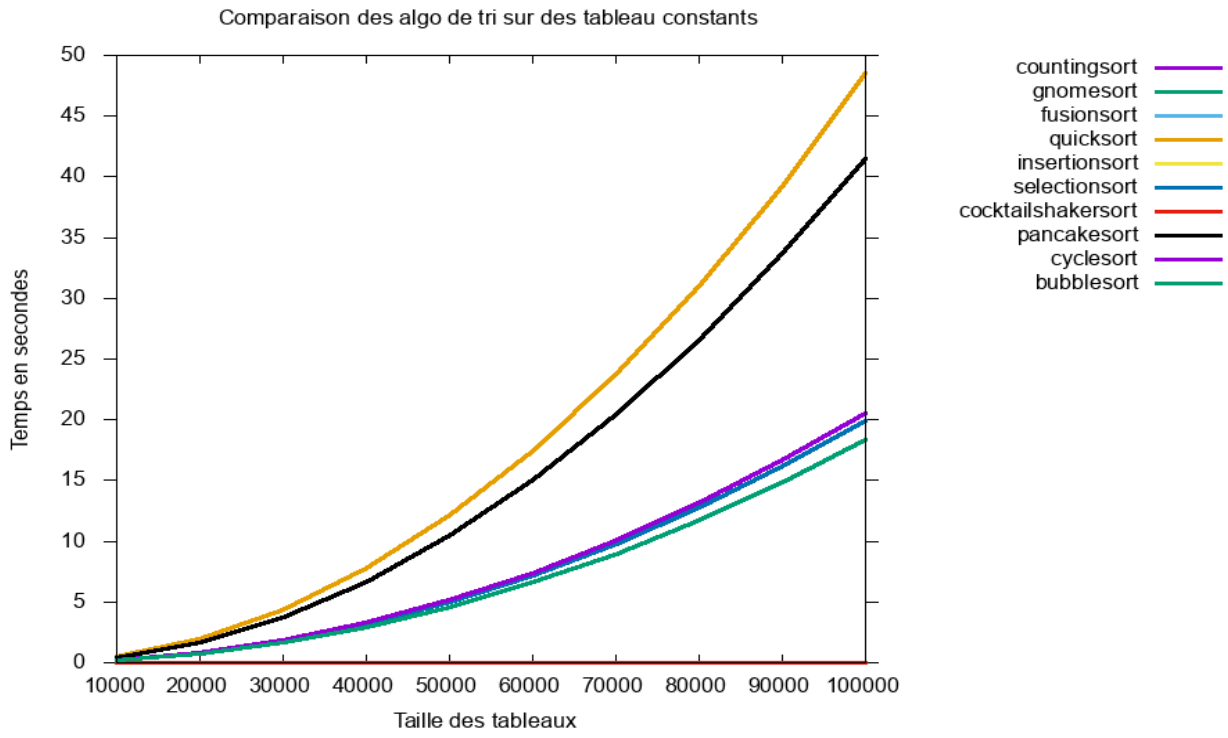
L'algorithme ayant le moins de performances est le tri par cycle.

En effet, plus la taille du tableau augmente, plus son temps d'exécution augmente de façon considérable.

Ceci est dû au fait que dans le tri cyclique, il y a toujours le parcours du sous-tableau entier à partir de la position courante afin de compter le nombre d'éléments inférieurs à l'élément courant.

Il doit donc s'exécuter dans un temps quadratique.

2. Graphiques des tableaux constants



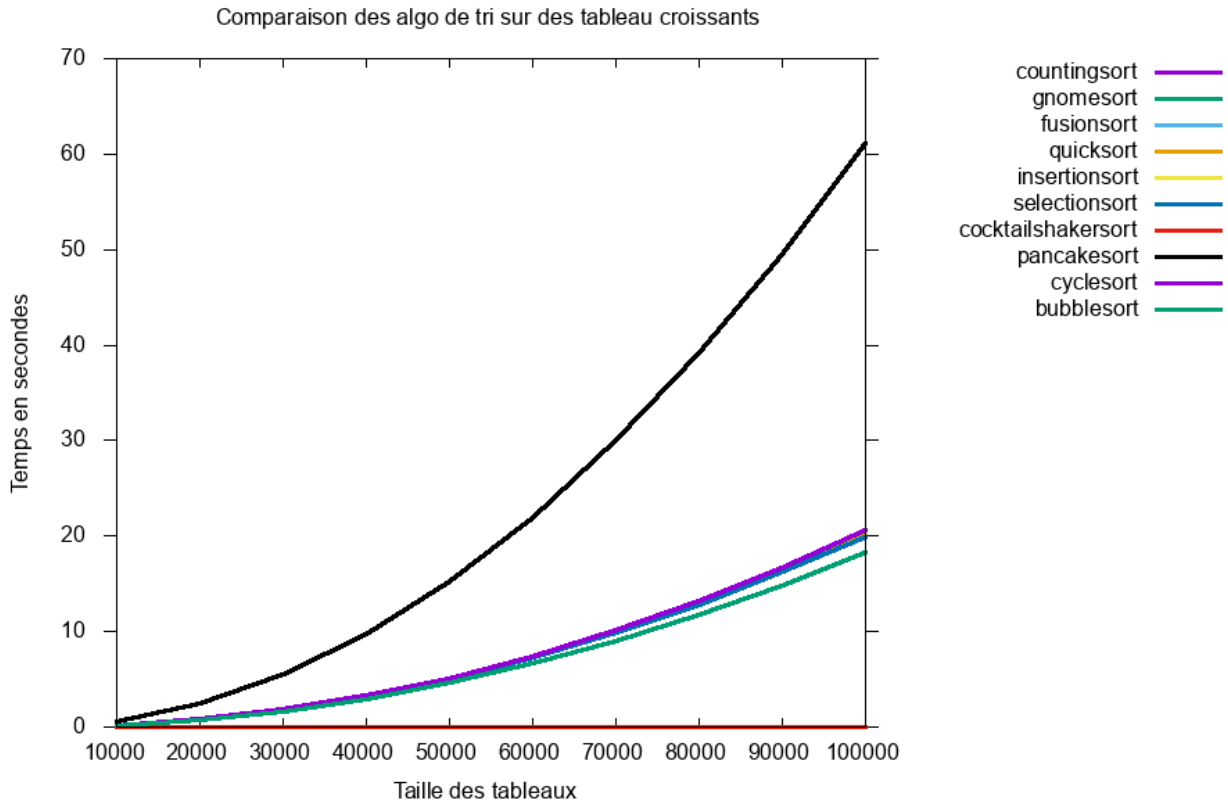
Remarques

L'algorithme ayant le moins de performances est le tri rapide.

En effet, plus la taille du tableau augmente, plus son temps d'exécution augmente de façon considérable.

Ceci est dû au choix du pivot, dans ce projet le début (gauche), ce qui cause donc un temps d'exécution considérable quand le tableau est déjà trié par ordre croissant, ce qui n'est pas le cas quand c'est un ordre décroissant.

3. Graphiques des tableaux croissants



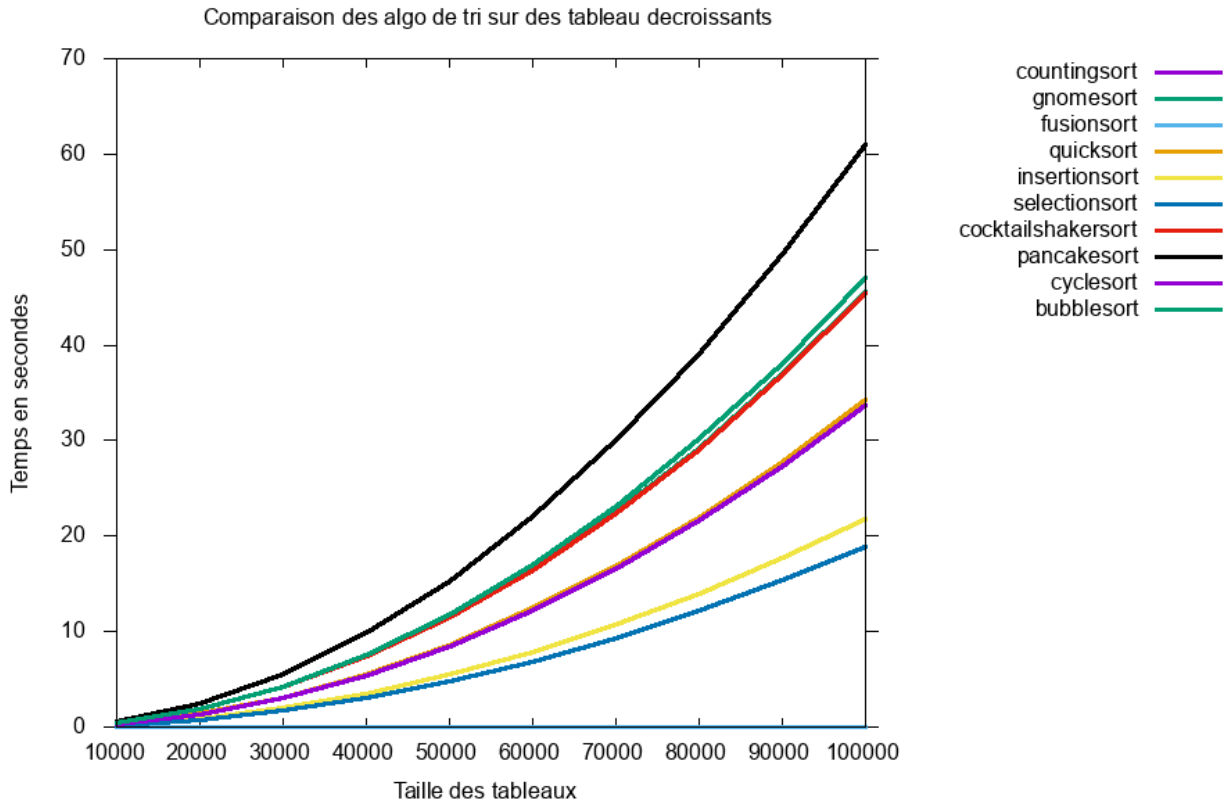
Remarques

L'algorithme ayant le moins de performance est le tri crêpes.

En effet, plus la taille du tableau augmente, plus son temps d'exécution augmente de façon considérable.

Ceci est dû aux valeurs présentes dans le tableau, car plus la taille augmente plus les valeurs augmentent (choix de conception), et ceci entraîne donc une alternance des grands et petits nombres, et donc le nombre de retournement de crêpes est beaucoup plus conséquent.

4. Graphiques des tableaux décroissants



Remarques

mêmes remarques que les tableaux constants.

Conclusion

Il n’existe pas forcément un algorithme performant dans toute condition. A travers ce projet, on peut déduire que le choix de l'algorithme de tri à utiliser dépend de plusieurs facteurs, que ce soit la taille, le type du tableaux ou le type de valeurs.

Ceci dit, le tri par comptage revient souvent comme étant le plus performant pour les tableaux d’entiers.

Références

https://fr.wikipedia.org/wiki/Algorithme_de_tri

https://learntutorials.net/fr/algorithm/topic/7252/tri-du-cycle

https://www.delftstack.com/tutorial/algorithm/pancake-sort/

https://www.javatpoint.com/cycle-sort

https://www.geeksforgeeks.org/when-does-the-worst-case-of-quicksort-occur/