

PW COMPLEXITY
Mr. Boutouil
Practical work

Analyse et Implémentation des Algorithmes de Tri

By Bounader Med Rafik & Haraoui Kouceila



1. Introduction

Dans le cadre du module de complexité algorithmique, ce projet a pour objectif d'étudier, implémenter et comparer différents algorithmes de tri. Le tri est une opération fondamentale en informatique, utilisée dans de nombreux domaines tels que la recherche, l'optimisation et la gestion de bases de données.

Le travail demandé consiste à écrire un programme permettant à l'utilisateur d'introduire les éléments d'un tableau, d'appliquer plusieurs méthodes de tri, d'afficher l'état du tableau après chaque étape, et de mesurer le temps d'exécution de chaque algorithme.

2. Objectifs du projet

Les objectifs principaux de ce projet sont :

- Comprendre le fonctionnement interne des algorithmes de tri classiques.
- Visualiser l'évolution du tableau à chaque étape du tri.
- Analyser et comparer les performances des algorithmes en termes de temps d'exécution.
- Relier les résultats pratiques aux notions théoriques de complexité algorithmique.

3. Environnement et outils utilisés

Le programme a été développé en langage C, choisi pour sa proximité avec le matériel et la précision qu'il offre dans la mesure du temps d'exécution.

Les outils utilisés sont :

- Un compilateur C (GCC).
- Les bibliothèques standards du langage C, notamment `stdio.h`, `stdlib.h` et `time.h` pour la gestion des entrées/sorties, de la mémoire et du temps.

4. Présentation générale du programme

Le programme est structuré autour de plusieurs fonctions, chacune correspondant à un algorithme de tri. Une fonction principale (`main`) se charge de :

- Lire la taille du tableau et ses éléments.
- Afficher le tableau initial.
- Appeler successivement les différentes méthodes de tri.
- Afficher le tableau après chaque étape de tri.
- Calculer et afficher le temps d'exécution de chaque algorithme.

Cette structure modulaire facilite la compréhension du code et permet de comparer facilement les différentes méthodes.

5. Algorithmes de tri implémentés

5.1 Tri à bulles (Bubble Sort)

Le tri à bulles est un algorithme simple basé sur des comparaisons successives d'éléments adjacents. À chaque passage, les plus grands éléments « remontent » vers la fin du tableau.

- Principe : comparer deux à deux les éléments voisins et les échanger s'ils sont mal ordonnés.
- Complexité temporelle :
 - Meilleur cas : $O(n)$ (tableau déjà trié)
 - Cas moyen et pire cas : $O(n^2)$

Cet algorithme est facile à comprendre mais peu efficace pour les grands tableaux.

5.2 Tri par sélection (Selection Sort)

Le tri par sélection consiste à rechercher, à chaque itération, le plus petit élément du tableau non trié et à le placer à sa position définitive.

- Principe : sélectionner le minimum et l'échanger avec l'élément courant.
- Complexité temporelle :
 - Tous les cas : $O(n^2)$

Même si le nombre d'échanges est réduit, le nombre de comparaisons reste élevé.

5.3 Tri par insertion (Insertion Sort)

Le tri par insertion fonctionne de manière similaire au tri de cartes dans la main. Il construit progressivement une partie triée du tableau.

- Principe : insérer chaque nouvel élément à sa position correcte dans la partie déjà triée.
- Complexité temporelle :
 - Meilleur cas : $O(n)$
 - Cas moyen et pire cas : $O(n^2)$

Cet algorithme est efficace pour les petits tableaux ou les tableaux presque triés.

5.4 Tri rapide (Quick Sort)

Le tri rapide est un algorithme de type « diviser pour régner ». Il choisit un pivot, partitionne le tableau, puis applique récursivement le tri sur les sous-tableaux.

- Principe : partitionnement autour d'un pivot.
- Complexité temporelle :
 - Meilleur et cas moyen : $O(n \log n)$
 - Pire cas : $O(n^2)$

Malgré son pire cas théorique, Quick Sort est très performant en pratique.

6. Mesure du temps d'exécution

Le temps d'exécution est mesuré à l'aide de la fonction `clock()` de la bibliothèque `time.h`. Le temps est calculé comme la différence entre l'instant de début et de fin de l'exécution de chaque algorithme.

Il est important de noter que ces mesures dépendent de plusieurs facteurs (machine, compilateur, charge du système). Ainsi, les temps obtenus donnent une indication comparative mais ne représentent pas une mesure absolue.

7. Analyse et comparaison des résultats

Les résultats montrent que :

- Les algorithmes simples (Bubble Sort, Selection Sort) sont nettement plus lents lorsque la taille du tableau augmente.
- Insertion Sort est plus performant sur des tableaux presque triés.
- Quick Sort offre les meilleurs temps d'exécution dans la majorité des cas.

Ces observations sont cohérentes avec les analyses théoriques de complexité.

8. Conclusion

Ce projet a permis de mettre en pratique les concepts de complexité algorithmique et de mieux comprendre le comportement réel des algorithmes de tri. L'affichage du tableau à chaque étape facilite la visualisation du processus de tri, tandis que la mesure du temps d'exécution permet une comparaison concrète des performances.

En conclusion, le choix d'un algorithme de tri dépend fortement de la taille des données et de leur organisation initiale. Les algorithmes avancés comme Quicksort sont généralement préférables pour les grands volumes de données.

9. Perspectives

Comme amélioration future, il serait possible :

- D'ajouter d'autres algorithmes de tri (Merge Sort, Heap Sort).
- De comparer également la complexité mémoire.
- D'implémenter une interface graphique pour mieux visualiser les étapes du tri.

Github Repo ↗