Université Paris Nanterre

Licence MIAGE troisième année

Rapport de projet informatique

Expérience algorithmique - le tri

Réalisé par

Badr Agrad

Table des matières

1 Introduction

1.1 Contexte

Les algorithmes de tri constituent une composante fondamentale en informatique, visant à organiser des éléments dans un ordre spécifique, souvent croissant ou décroissant. Leur utilité réside dans la capacité à organiser des données de manière efficace et ordonnée, ce qui est essentiel dans de nombreuses applications informatiques.

Ces algorithmes permettent de résoudre divers problèmes, tels que la recherche rapide d'éléments, la facilitation de l'insertion et de la suppression dans des structures de données ordonnées, ou encore l'amélioration des performances pour des opérations de fusion ou de comparaison.

Il existe une variété d'algorithmes de tri, chacun ayant ses propres caractéristiques en termes de complexité temporelle, d'utilisation de la mémoire et d'adaptabilité aux différents types de données. Certains des algorithmes de tri les plus couramment utilisés incluent le tri par insertion, le tri par sélection, le tri à bulles, le tri rapide (QuickSort), le tri fusion (MergeSort), et bien d'autres.

La sélection de l'algorithme de tri adapté dépend souvent de la taille des données à trier, de la complexité des données et des exigences de performances spécifiques à l'application informatique concernée.

En somme, les algorithmes de tri jouent un rôle crucial dans la manipulation et l'organisation efficace des données en informatique, fournissant des méthodes pour ordonner les informations afin de faciliter les opérations et d'optimiser les performances des programmes.

1.2 Motivation

Le projet vise à explorer et implémenter divers algorithmes de tri dans le but d'évaluer leurs performances face à des ensembles de données variés. L'objectif principal est d'analyser et de comparer le temps d'exécution de ces algorithmes en fonction de différentes tailles et structures de données, telles que des ensembles de nombres premiers, des puissances de 2, ou d'autres jeux de données volumineux.

Cette démarche découle de la nécessité de comprendre et de mesurer les performances relatives des algorithmes de tri dans des contextes variés. En informatique, la sélection d'un algorithme de tri adapté à un ensemble de données spécifique peut avoir un impact significatif sur les performances globales d'une application. Par conséquent, cette évaluation comparative des algorithmes de tri permettra de déterminer quel algorithme est le plus efficace pour des types de données spécifiques et des volumes de données variables.

En étudiant et en évaluant les performances des algorithmes tels que le tri rapide (QuickSort), le tri fusion (MergeSort), le tri par insertion, ou d'autres méthodes de tri, ce projet vise à fournir des données tangibles sur la complexité temporelle et les performances pratiques de chaque algorithme. De plus, il offrira des informations précieuses pour guider le choix des développeurs et des ingénieurs lors de la sélection d'un algorithme de tri adapté à leurs besoins spécifiques.

En résumé, ce projet vise à fournir une analyse comparative détaillée des performances des différents algorithmes de tri, en les évaluant sur une variété de jeux de données, afin d'aider à déterminer leur efficacité respective dans différents scénarios informatiques.

2 Algorithmes de tri usuels

Il existe énormément de méthodes algorithmiques afin de trier des données. Voici quelques algorithmes usuels :

2.1 Bubble Sort

Le tri à bulles (Bubble Sort) est l'un des algorithmes de tri les plus simples. Son principe est de parcourir le tableau à trier plusieurs fois. À chaque passage, il compare les éléments adjacents et les échange s'ils ne sont pas dans le bon ordre. Ce processus est répété plusieurs fois (autant de fois qu'il y a d'éléments dans le tableau ou jusqu'à ce qu'aucun échange ne soit nécessaire). À chaque passage, les éléments les plus grands "remontent" vers la fin du tableau, comme des bulles remontant à la surface de l'eau d'où le nom "Bubble Sort".

Algorithm 1 Bubble Sort

Input: Tableau arr de taille n
Output: Tableau trié en ordre croissant
for $i \leftarrow 0$ **to** $n-1$ **do**
  for $j \leftarrow 0$ **to** $n-i-1$ **do**
    if $arr[j] > arr[j+1]$ **then**
      échanger $arr[j]$ et $arr[j+1]$
    end
  end
end

2.2 Insertion Sort

Le tri par insertion est un algorithme de tri simple et intuitif.Pour trier un tableau, l'algorithme commence par considérer le deuxième élément et le compare au premier. Si le deuxième élément est plus petit, il l'insère avant le premier élément pour que les deux soient dans le bon ordre. Ensuite, il prend le troisième élément et l'insère au bon endroit parmi les deux premiers, et ainsi de suite.

À chaque étape, l'algorithme insère un élément dans la section déjà triée du tableau, décalant les éléments plus grands vers la droite pour faire de la place. Il continue de cette manière jusqu'à ce que tous les éléments soient placés à leur bonne position dans la séquence triée.

Algorithm 2 Insertion Sort

Input: Tableau arr de taille n
Output: Tableau trié en ordre croissant
for $i \leftarrow 1$ **to** $n-1$ **do**
  $key \leftarrow arr[i]$ $j \leftarrow i-1$ **while** $j \geq 0$ ***et*** $arr[j] > key$ **do**
    $arr[j+1] \leftarrow arr[j]$ $j \leftarrow j-1$
  end
  $arr[j+1] \leftarrow key$
end

2.3 Gnome Sort

Le Gnome Sort opère en déplaçant un élément à sa bonne position comme le ferait un gnome pour placer une fleur dans une rangée de pots.

Il fonctionne en parcourant le tableau de données. Si l'élément courant est plus grand que l'élément précédent, il se déplace vers la gauche, sinon il avance d'une position vers la droite. Cela continue jusqu'à ce que l'élément soit à sa bonne position par rapport aux éléments précédents.

Algorithm 3 Gnome Sort

Input: Tableau arr de taille n
Output: Tableau trié en ordre croissant
$i \leftarrow 0$ **while** $i < n$ **do**
  if $i == 0$ ***ou*** $arr[i] \geq arr[i-1]$ **then**
    $i \leftarrow i + 1$
  else
    échanger $arr[i]$ et $arr[i-1]$ $i \leftarrow i - 1$
  end
end

3 Complexité des Algorithmes de tri

Les algorithmes de tri tels que le tri à bulles (Bubble Sort), le tri par sélection (Selection Sort) ou encore le Gnome Sort ont une complexité temporelle quadratique. Cela signifie que leur temps d'exécution augmente de manière quadratique à mesure que la taille des données à trier croît.

Par exemple, prenons le tri à bulles : pour trier un tableau de n éléments, l'algorithme doit effectuer approximativement n^2 comparaisons et échanges d'éléments. De même, le tri par sélection et le Gnome Sort effectuent également un nombre quadratique d'opérations pour trier des ensembles de données de grande taille.

Cette complexité quadratique devient rapidement inefficace pour des ensembles de données volumineux. Par conséquent, bien que ces algorithmes soient simples à comprendre et à implémenter, ils deviennent impraticables pour des applications manipulant de grandes quantités de données.

Lorsque la taille des données augmente, ces algorithmes de tri quadratiques deviennent moins performants en termes de temps d'exécution, et d'autres algorithmes de tri comme le tri rapide (QuickSort) ou le tri fusion (MergeSort) deviennent des choix plus appropriés. Ces derniers ont une complexité temporelle inférieure, ce qui les rend plus efficaces pour des ensembles de données de grande taille.

En conclusion, bien que les algorithmes de tri quadratiques soient simples, leur performance décline rapidement pour des ensembles de données de grande taille en raison de leur complexité temporelle quadratique.

4 Experience

4.1 Code

Les algorithmes de tri ont été codés en langage C afin d'optimiser au maximum la vitesse d'exécution. Nous les regrouperons tous dans un fichier appelé "sort.c". Voici la liste des algorithmes qui ont été implémentés au projet :

- **Bubble Sort :** Compare et échange les éléments adjacents s'ils sont dans le mauvais ordre, répétant ce processus jusqu'à ce que le tableau soit trié.
- **Bubble Sort Optimisé :** Une variante du Bubble Sort qui arrête le tri s'il n'y a aucun échange lors d'un parcours complet du tableau.
- **Insertion Sort :** Trie le tableau en insérant chaque élément à sa place dans la partie déjà triée du tableau.
- **Counting Sort :** Trie les éléments en comptant le nombre d'occurrences de chaque élément distinct.
- **Merge Sort :** Divise récursivement le tableau en sous-tableaux, les trie et les fusionne pour obtenir un tableau trié.
- **Selection Sort :** Recherche le minimum à chaque itération et le place à la bonne position.
- **Gnome Sort :** Compare les éléments adjacents, déplaçant l'élément vers la gauche ou la droite jusqu'à sa place correcte.
- **Odd-Even Sort :** Trie de manière itérative en comparant et en échangeant les éléments à des indices pairs et impairs.
- **Tree Sort :** Utilise une structure d'arbre pour trier les éléments du tableau.
- **Comb Sort :** Une variante du Bubble Sort qui compare les éléments avec un intervalle décroissant.

4.2 Structure du projet

Le projet est structuré de manière modulaire, divisé entre les fonctions de tri implémentées en langage C et regroupées dans les fichiers `sort.c` et `sort.h`, ainsi que les fonctionnalités de génération de jeux de données, de mesure du temps d'exécution, et d'analyse statistique, qui sont implémentées en Python dans le fichier `main.py`.

Les fonctions de tri telles que `bubble_sort`, `bubble_sort_opt`, `insertion_sort`, etc., partagent le même prototype : `exemple_sort(int *tab, int taille)`. Ces fonctions sont regroupées dans un fichier `sort.c`, tandis que leur déclaration se trouve dans `sort.h`.

Pour lier les fichiers C avec le script Python, une bibliothèque partagée (`libsort.so`) est générée avec la commande suivante :

```
gcc -shared -o libsort.so obj/sort.o obj/main.o -Iinclude
```

Dans le script Python, la bibliothèque partagée est chargée à l'aide du module `ctypes`, permettant d'appeler les fonctions de tri implémentées en C directement depuis Python. Le choix de Python s'avère pertinent pour l'analyse statistique des données obtenues.

Le script Python utilise également la bibliothèque `matplotlib` pour tracer des graphiques présentant les temps d'exécution des algorithmes de tri selon les différents types et quantités de données.

Voici l'arborescence de fichier du projet :

```
/ (racine)
├── Makefile
├── libsort.so
├── main.py
├── bin
│   └── main
├── include
│   └── sort.h
├── obj
│   ├── sort.o
│   └── main.o
└── src
    ├── sort.c
    └── main.c
```

5 Résultat obtenu et conclusions

Les résultats obtenus confirment les attentes concernant les algorithmes de tri à complexité quadratique tels que le Bubble Sort. Comme prévu, le temps d'exécution augmente au fur et à mesure que la taille des données augmente. Cette augmentation n'est pas linéaire, mais plutôt à un certain point, le temps d'exécution semble augmenter de manière quasi exponentielle.

En revanche, les algorithmes comme le Tree Sort et le Merge Sort montrent des résultats encourageants, avec des temps d'exécution restant en dessous de 0.1 secondes même pour des ensembles de données plus importants.

Le Counting Sort se démarque comme le meilleur en termes de rapidité, triant extrêmement rapidement toutes les données, à l'exception du cas "Random 0 to INT_MAX", qui présente un temps d'exécution élevé par rapport aux autres ensembles ne dépassant pas les 0.01 secondes ! En effet, le Counting Sort fonctionne particulièrement bien lorsque la gamme des valeurs à trier est relativement petite par rapport à la taille de l'ensemble de données. Il est souvent utilisé pour trier des entiers dans une plage spécifique, ce qui lui permet d'atteindre une complexité linéaire en temps. Cela explique pourquoi le temps de tri est aussi conséquent pour ce qui est des valeurs entre 0 et INT_MAX.

En conclusion, les performances des algorithmes de tri varient significativement en fonction de la méthode utilisée. Le choix d'un algorithme dépendra des caractéristiques spécifiques des données à trier et des exigences de performance.

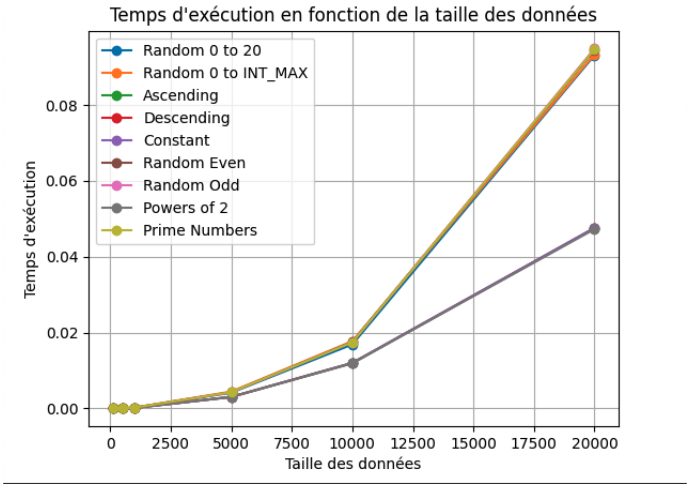

FIGURE 1 – Évolution du temps d'exécution pour le Bubble Sort

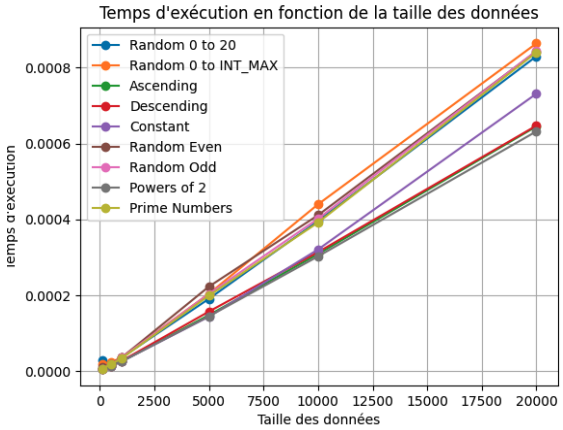

FIGURE 2 – Évolution du temps d'exécution pour le Merge Sort

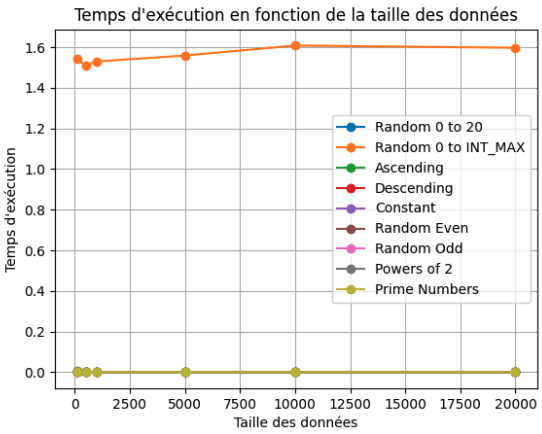

FIGURE 3 – Évolution du temps d'exécution pour le Counting Sort

6 Manuel d'utilisation

Une MakeFile a été créé afin de pouvoir compiler l'ensemble des fichiers. Ainsi, il vous suffit d'entrer la commande `make` dans le terminal pour compiler le projet. Ensuite, il vous faudra entrer la commande

```
gcc -shared -o libsort.so obj/sort.o obj/main.o -Iinclude
```

afin de générer la bibliothèque partagée entre C et python. Enfin, il faudra entrer la commande `python3 main.py example_sort` , avec en argument le nom de l'un des tri qui a été implémenté dans le projet, dans le terminal.

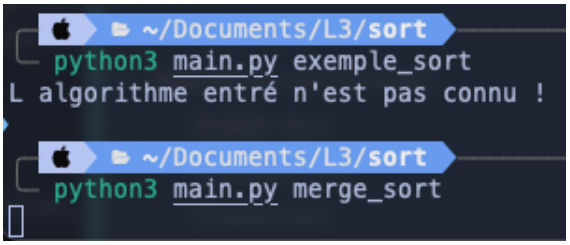

FIGURE 4 – Execution du projet