INFO0250 - Programmation avancée

Projet 1 : Algorithmes de tri

Groupe 1: Martin RANDAXHE, Cyril RUSSE

Table des matières

1 Introduction

2 Algorithmes vus au cours

Dans cette section, nous allons analyser les différents algorithmes ayant été vus lors du cours théorique et implémenté dans le cadre de ce projet. Pour ce faire, nous allons présenter les résultats des calculs empiriques des temps d'exécution des 3 algorithmes concernés. Par la suite, nous comparerons l'évolution de ceux-ci par rapport aux complexités théoriques en fonction de la taille du tableau. Et finalement y apporter nos analyses vis à vis de l'ordre relatif de ces différents algorithmes.

2.1 Résultats empiriques de temps d'exécution

Voici, présenté dans la table ci-dessous, les valeurs de temps d'exécution des 3 algorithmes de tri vus au cours : InsertionSort, QuickSort et HeapSort.(Algorithmes respectivement implémentés dans les fichiers InsertionSort.c, QuickSort.c et HeapSort.C) Ces valeurs sont basées sur une moyenne de 10 temps d'execution pour des tableaux de taille 10^3, 10^4 et 10^5.

Type de tableau	aléatoire			croissant		
Taille	10^3	10^4	10^5	10^3	10^4	10^5
InsertionSort	0.001939	0.06914	6.65226	0.000017	0.000146	0.001397
QuickSort	0.000377	0.001536	0.044997	0.011423	0.33125	31.01124
HeapSort	0.000291	0.003594	0.009951	0.00008	0.000463	0.003292
PlaceSort	0.009042	0.479786	41.534067	0.00707	0.230783	22.825008

FIGURE 1 – Résultats empiriques de temps d'exécution

2.2 Analyse des tests en fonction des complexités théoriques des différents algorithmes

1. InsertionSort :
   — croissant : Nous retrouvons entre 10^3 et 10^4 un facteur 8,59 et entre 10^4 et 10^5 un facteur 9.57. En moyenne, cela nous fait 9.08$\simeq$ 10. La complexité $\Theta(n)$ est vérifiée.
   — aléatoire : Nous retrouvons entre 10^3 et 10^4 un facteur 35.66 et entre 10^4 et 10^5 un facteur 96.21. En moyenne, cela nous fait 65.93. Nous nous rapprochons bel et bien du facteur 10^2 attendu, d'autant plus notable entre les tableaux $10^4 et 10^5$. La complexité $\Theta(n^2)$ est vérifiée.
2. QuickSort :
   — croissant : Nous retrouvons entre 10^3 et 10^4 un facteur 93.62 et entre 10^4 et 10^5 un facteur 29. En moyenne, cela nous fait 61.31. Nous nous rapprochons bel et bien du facteur 10^2 attendu, d'autant plus notable entre les tableaux 10^4 et 10^5. La complexité $\Theta(n^2)$, étant donné que nous sommes dans le pire cas pour le QuickSort qui va toujours avoir sa valeur de pivot qui ne permettra pas de "diviser pour mieux régner", est vérifiée.
   — aléatoire : Nous retrouvons entre 10^3 et 10^4 un facteur 4.07 et entre 10^4 et 10^5 un facteur 29.29. En moyenne, cela nous fait 16.68. Nous nous rapprochons bel et bien du facteur $10 log(10)$ attendu, d'autant plus notable entre les tableaux 10^4 et 10^5. La complexité $\Theta(n * log(n))$ est vérifiée.
3. HeapSort :
   — croissant : Nous retrouvons entre 10^3 et 10^4 un facteur 5.79 et entre 10^4 et 10^5 un facteur 7.11. En moyenne, cela nous fait 6.45. Théoriquement la complexité du heapsort

est $\Theta(n * log(n))$, nous nous en rapprochons dans ce cas-ci, les résultats sont mêmes meilleurs que ceux attendus.

— aléatoire : Nous retrouvons entre 10^3 et 10^4 un facteur 12.35 et entre 10^4 et 10^5 un facteur 2.77. En moyenne, cela nous fait 7.56. La complexité $\Theta(n*log(n))$ est vérifiée. Cet algorithme est de loin le plus performant quelque soit le cas, il respecte sa complexité théorique de $\Theta(n * log(n))$ quelque soit le cas et est même plus efficace que les autres algorithmes ayant cette même complexité.

2.3

3 PlaceSort

3.1 Pseudo-code

```
        PLACE(A)
            i=1
            while(i<=A.length)
                m=1
                for j=1 to A.length
                    if A[i]>A[j]
                        m = m + 1
                if m!=i
                    if A[i]==A[m]
                        while (A[i]==A[m])
                            m = m + 1
                        swap(A[i], A[m])
                        i = i + 1
                    else
                        swap(A[i], A[m])
                else
                    i = i + 1
```