INFO2050 - Programmation avancée

Projet 1: Algorithmes de tri

Jean-Michel Begon – Romain Mormont – Pascal Fontaine

02 octobre 2020

L'objectif du projet est d'implémenter, de comparer et d'analyser les algorithmes de tri suivants:

(a) l'algorithme InsertionSort (fourni);

(b) l'algorithme QuickSort (à implémenter);

(c) l'algorithme HeapSort (à implémenter);

(d) l'algorithme PlaceSort (voir Section 2, à implémenter);

(e) l'algorithme RecSort (voir Section 3, à implémenter).

1 Algorithmes vus au cours: analyse expérimentale

Dans cette partie du projet, vous allez étudier les performances de vos implémentations de QuickSort et HeapSort, ainsi que celle de InsertionSort.

Dans votre rapport, nous vous demandons de répondre aux questions suivantes:

1.a. calculez empiriquement le temps d'exécution moyen de ces trois algorithmes sur des tableaux de tailles croissantes (de 1.000, 10.000 et 100.000 éléments) lorsque ces tableaux sont aléatoires ou ordonnés de manière croissante. Reportez ces résultats dans une table au format donné ci-dessous[1]

Type de tableau	aléatoire			croissant		
Taille	10^3	10^4	10^5	10^3	10^4	10^5
InsertionSort						
QuickSort						
HeapSort						

1.b. commentez ces résultats. Pour chaque type de tableau:

- comparez l'évolution des temps de calcul en fonction de la taille du tableau aux complexités théoriques;
- commentez l'ordre relatif des différents algorithmes en reliant vos observations aux complexités théoriques.

Remarques:

- Les fonctions pour générer les tableaux vous sont fournies dans le fichier `Array.c`.
- Les temps reportés doivent être des temps moyens établis sur base de 10 expériences.

[1] Vous pouvez séparer cette table en deux tables, un par type de tableau.

2 PlaceSort

Lors de l'entretient technique pour être embauché chez *Sort Corp.*, l'examinateur vous demande d'implémenter l'algorithme PlaceSort qu'il vous décrit de la sorte.

Soit i, l'index d'un élément dans le tableau A. L'algorithme consiste à trouver la position finale de l'élément $A[i]$ en comptant les valeurs inférieures à cet élément. Si $A[i]$ n'est pas à la bonne place—appelons k sa position dans le tableau trié—, on échange $A[i]$ et $A[k]$ et on recommence en cherchant la position du nouvel élément à la position i. Une fois qu'il n'y a plus de permutation ($k = i$), l'algorithme continue avec $i = i + 1$. Initialement $i = 1$.

Pour réussir l'entretient d'embauche, vous devez:

2.a. donner le pseudo-code de PlaceSort.

2.b. étudier expérimentalement la complexité en temps de cet algorithme. Pour ce faire, suivez le même protocole qu'à la section 1 et ajoutez une ligne pour PlaceSort. Comment se compare PlaceSort avec les autres algorithmes?

2.c. étudier et justifier la complexité en temps de PlaceSort au pire et meilleur cas. Comparez avec les temps mesurés à la question précédente.

3 RecSort

Fort de votre connaissance poussée en algorithmes de tri, PlaceSort vous inspire un nouvel algorithme basé sur QuickSort et MergeSort. Vous décidez de le présenter à l'examinateur pour l'impressionner.

L'idée est de placer un pivot (par exemple le dernier élément) à la bonne position, de trier récursivement les sous-tableaux à gauche et à droite du pivot (comme dans QuickSort) et finalement de fusionner les deux sous-tableaux triés (comme dans MergeSort). En voici le pseudo-code.

```
Place(A, p, r, k)
1  m = 0
2  for i = p to r
3       if A[i] < A[k]
4            m = m + 1
5  l = m + p
6  Swap(A[k], A[l])
7  return l
```

```
RecSort(A, p, r)
1  if p < r
2       q = Place(A, p, r, r)
3       RecSort(A, p, q − 1)
4       RecSort(A, q + 1, r)
5       Merge(A, p, q, r)
```

La fonction Place(A, p, r, k) va placer l'élément $A[k]$ ($p \leq k \leq r$) à la position qu'il occuperait dans le sous-tableau $A[p\,..\,r]$ si celui était trié. Elle renvoie également cette position. La fonction Merge est identique à celle du cours, à la modification près que le pivot ne doit pas bouger de place. RecSort(A, p, r) ($1 \leq p, r \leq A.length$) trie le sous-tableau $A[p\,..\,r]$. En particulier RecSort$(A, 1, A.length)$ trie le tableau A.

Pour impressionner l'examinateur, vous devez

3.a. étudier expérimentalement la complexité en temps de cet algorithme. Pour ce faire, suivez le même protocole qu'à la section 1 et ajoutez une ligne pour RecSort. Comment se compare RecSort avec les autres algorithmes?

3.b. étudier et justifier la complexité en temps de RecSort au pire et meilleur cas. Comparez avec les temps mesurés à la question précédente;

3.c. conclure de l'intérêt pratique de PlaceSort et RecSort sur base des analyses théoriques et empiriques.

Deadline et soumission

Le projet est à réaliser ***en binôme*** pour le ***26 octobre 2020 à 23h59*** au plus tard. Le projet est à remettre via la plateforme de soumission de Montefiore: `http://submit.montefiore.ulg.ac.be/`.

Il doit être rendu sous la forme d'une archive `tar.gz` contenant:

(a) votre rapport (5 pages maximum) au format PDF. Soyez bref mais précis et respectez bien la numération des (sous-)questions;

(b) les fichiers `.c` suivants contenant l'implémentation de l'algorithme correspondant

- le fichier `QuickSort.c`;
- le fichier `HeapSort.c`;
- le fichier `PlaceSort.c`;
- le fichier `RecSort.c`.

Respectez bien les extensions de fichiers ainsi que les noms des fichier `*.c` (en ce compris la casse). Les fichiers suivants vous sont fournis:

- `Array.h` et `Array.c`: une petite bibliothèque pour générer différents types de tableaux.
- `Sort.h`: le header contenant le prototype de la fonction de tri.
- `InsertionSort.c`: implémentation de l'algorithme de InsertionSort donnée à titre d'exemple pour vos propres implémentations.
- `main.c`: un petit fichier de test.

Vos fichiers seront évalués sur les machines ms8xx avec la commande[2]:

```
gcc main.c Array.c InsertionSort.c --std=c99 --pedantic -Wall -Wextra -Wmissing-prototypes -DNDEBUG -lm -o test
```

Ceci implique que:

- Le projet doit être réalisé dans le standard C99.
- La présence de *warnings* impactera négativement la cote finale.
- Un projet qui ne compile pas avec cette commande sur ces machines recevra une cote nulle (pour la partie code du projet).

Un projet non rendu à temps recevra une cote globale nulle. En cas de plagiat[3] avéré, l'étudiant se verra affecter une cote nulle à l'ensemble des projets.

Les critères de correction sont précisés sur la page web des projets.

Bon travail !

[2] En substituant adéquatement `InsertionSort.c` par l'implémentation de l'algorithme que nous voulons tester.
[3] Des tests anti-plagiat seront réalisés.