Structure de données et algorithmes
Projet 1: Algorithmes de tri

Pierre Geurts – Jean-Michel Begon - Romain Mormont

22 février 2019

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 MergeSort (à implémenter).

(d) L'algorithme HeapSort (à implémenter).

(e) L'algorithme InPlaceSort (voir Section 2, à implémenter).

1 Algorithmes vus au cours: analyse expérimentale

Dans cette partie du projet, vous allez étudier les performances de vos implémentations de InsertionSort, QuickSort, MergeSort et HeapSort. Dans votre rapport, nous vous demandons de répondre aux questions suivantes:

1.a. Calculez empiriquement le temps d'exécution moyen de ces quatre algorithmes sur des tableaux de tailles croissantes (de 10.000, 100.000 et 1.000.000 éléments) lorsque ces tableaux sont aléatoires ou ordonnés de manière croissante ou décroissante. Reportez ces résultats dans une table au format donné ci-dessous[1].

Type de tableau	aléatoire			croissant			décroissant		
Taille	10^4	10^5	10^6	10^4	10^5	10^6	10^4	10^5	10^6
InsertionSort									
QuickSort									
MergeSort									
HeapSort									

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

- comparer 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 trois tables pour chaque type de tableau

2 Un nouvel algorithme de tri: InPlaceSort

Fort de vos nouvelles connaissances en algorithmique, vous avez décroché un poste chez *Sort & Ltd.*, entreprise spécialisée dans la création d'algorithmes de tri de tableaux. Votre premier projet consiste à produire une variante *en place* du tri par fusion.

Soit un tableau A et les indices p, q et r tels que

- $p \leq q < r$
- Les sous-tableaux $A[p\,..\,q]$ et $A[q+1\,..\,r]$ sont triés

Vous venez d'imaginer une fonction InPlaceMerge(A, p, q, r) qui fusionne *en place* les deux sous-tableaux triés de la manière suivante:
Soit $i_l = p$ et $i_r = q + 1$, deux indices placés au début des deux sous-tableaux:

- Tant que $i_l < q$, $i_r < r$ et l'élément à la position $i_r - 1$ est plus grand que celui à la position i_r:
  (i) Si l'élément à la position i_l est inférieur ou égal à l'élément à la position i_r, on incrémente i_l.
  (ii) Sinon, on place l'élément à la position i_r à la bonne place (c'est-à-dire à la position i_l) en décalant le sous-tableau $A[i_l\,..\,i_r - 1]$ d'une position vers la droite et on incrémente i_l.

Le reste de l'algorithme du tri par fusion est identique à celui du cours (en substituant la fonction Merge par InPlaceMerge)

Vous êtes sur le point de présenter l'algorithme à votre patron mais quelque chose vous perturbe. Vous décidez d'analyser l'algorithme plus en détail:

2.a. Repérez l'(les) erreur(s) dans le descriptif.

2.b. Donnez le pseudo-code de InPlaceMerge.

2.c. Est-ce que cet algorithme de tri est stable? Justifiez votre réponse.

2.d. Etudiez 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 InPlaceSort. Comment se compare InPlaceSort avec les autres algorithmes?

2.e. Etudiez et justifiez la complexité en temps et en espace de InPlaceSort pour le meilleur et le pire cas. Comparez avec les temps mesurés à la question précédente.

2.f. Concluez quant à l'intérêt pratique de ce nouvel algorithme de tri.

Les réponses à ces questions doivent être présentes dans votre rapport.

Deadline et soumission

Le projet est à réaliser ***en binôme*** pour le ***21 mars 2019 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) Un fichier `QuickSort.c`.

(c) Un fichier `MergeSort.c`.

(d) Un fichier `HeapSort.c`.

(e) Un fichier `InPlaceSort.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 des 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