# Structure de données et algorithmes
## Projet 1: Algorithmes de tri

Pierre Geurts – Jean-Michel Begon - Romain Mormont

1 mars 2017

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 standard (à implémenter).

(c) L'algorithme QuickSort avec un pivot aléatoire (à implémenter).

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

(e) L'algorithme GSort (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 standard ($i$), QuickSort avec un pivot aléatoire ($ii$) et HeapSort. Dans votre rapport, nous vous demandons de répondre aux questions suivantes:

**1.a.** Soit $N$ le nombre de données à trier dans un tableau. Calculez empiriquement le temps d'exécution moyen de ces quatre algorithmes pour différentes valeurs de $N$ (10, 100, 1.000, 10.000, 100.000 et 1.000.000) lorsque les tableaux sont générés de manière complètement aléatoires. Reportez ces résultats dans une table au format donné ci-dessous.

| $N$ | InsertionSort | QuickSort ($i$) | QuickSort ($ii$) | HeapSort |
|---|---|---|---|---|
| 10 | | | | |
| 100 | | | | |
| 1.000 | | | | |
| 10.000 | | | | |
| 100.000 | | | | |
| 1.000.000 | | | | |

**1.b.** Commentez ces résultats en comparant les algorithmes :

- les uns par rapport aux autres
- par rapport à leur complexité théorique

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.

## 2 Un nouvel algorithme de tri: *GSort*

Fort de vos nouvelles connaissances en algorithmique, vous avez décroché un poste chez *Sort & Co.*, entreprise spécialisée dans la création d'algorithmes de tri. Un jour, un de vos collègues arrive en trombe annonçant qu'il a inventé un nouvel algorithme de tri, qu'il a appelé *GSort*, et dont voici le pseudo-code[1]:

```
GSort(A, p, r)
1  if p ≥ r
2       return
3  GSort(A, p, r − 1)
4  if A[r − 1] > A[r]
5       s = A[r − 1]
6       A[r − 1] = A[r]
7       A[r] = s
8       GSort(A, p, r − 1)
```

En tant que nouveau, vous êtes désigné pour faire une analyse fouillée de cet algorithme, afin de décider de son intérêt commercial ou non pour votre société:

**2.a.** Expliquez intuitivement pourquoi l'algorithme est correct.

**2.b.** Est-ce que cet algorithme de tri est stable ? Est-il en place ? Justifiez vos réponses.

**2.c.** Etudiez et justifiez la complexité en temps au meilleur cas de *GSort* et expliquez à quoi correspond ce meilleur cas.

**2.d.** Votre patron, qui a de l'expérience, prétend que cet algorithme est dans le pire cas $\Theta(2^N)$. Expliquez à quoi correspond le pire cas et vérifiez **empiriquement** l'intuition de votre patron. Expliquez l'expérience que vous avez réalisée et en quoi elle confirme ou infirme une complexité $\Theta(2^N)$ dans le pire cas.

**2.e.** Si votre patron avait raison, justifiez son intuition. S'il avait tort, étudiez et justifiez la complexité réelle dans le pire cas.

**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 **individuellement** pour le **vendredi 24 mars 2017** 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` contenant l'implémentation de QuickSort standard.

(c) Un fichier `RandomizedQuickSort.c` contenant l'implémentation de QuickSort avec pivot aléatoire.

(d) Un fichier `HeapSort.c` contenant l'implémentation de l'algorithme du même nom.

[1] Les paramètres $p$ et $r$ sont respectivement les indices de début et de fin (inclus) du sous-tableau à trier. Pour trier l'entiereté d'un tableau $A$, l'appel serait GSort($A$, 1, $A.length$).

(e) Un fichier `GSort.c` contenant l'implémentation de l'algorithme du même nom.

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 des fonctions à implémenter.
- `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
-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 avéré, l'étudiant se verra affecter une cote nulle à l'ensemble des projets.

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

**Bon travail !**

[2]En replaçant évidemment `InsertionSort.c` par l'implémentation de l'algorithme que nous voulons tester.