université PARIS DIDEROT PARIS 7

**L2 Informatique**

Nom :

Prénom :

# EA4 – Éléments d'algorithmique
## Partiel – 2 mars 2016
### Durée : 1h45

***Document autorisé : une feuille A4 manuscrite***
***Appareils électroniques éteints et rangés***

*Le sujet est (trop) long, il en sera tenu compte dans la notation. Les exercices sont indépendants et ne sont absolument pas classés par ordre de difficulté.*
*Sauf mention contraire, on s'intéresse à la complexité* ***dans le pire des cas****.*

**Exercice 1 :**

Compléter le tableau ci-dessous avec les ordres de grandeur des complexités en temps des différents algorithmes de tri étudiés en cours, en fonction du nombre $n$ d'éléments à trier.

| | meilleur cas | en moyenne | pire cas |
|---|---|---|---|
| tri par sélection | | | |
| tri par insertion | | | |
| tri rapide | | | |
| tri par fusion | | | |

**Exercice 2 :**

Cocher les assertions exactes.

| | | $f \in \Theta(g)$ | $f \notin \Theta(g)$ | $f \in O(g)$ | $f \in \Omega(g)$ |
|---|---|---|---|---|---|
| $f(n) = (n(n+1))^2$ | $g(n) = n^4$ | ☐ | ☐ | ☐ | ☐ |
| $f(n) = (n(n+1))^2$ | $g(n) = 4n$ | ☐ | ☐ | ☐ | ☐ |
| $f(n) = \log n$ | $g(n) = \sqrt{n}$ | ☐ | ☐ | ☐ | ☐ |
| $f(n) = \log(n^4)$ | $g(n) = \log(4n)$ | ☐ | ☐ | ☐ | ☐ |
| $f(n) = \log(n^4)$ | $g(n) = (\log n)^4$ | ☐ | ☐ | ☐ | ☐ |
| $f(n) = n^4 + \sqrt{n}$ | $g(n) = n^4\sqrt{n}$ | ☐ | ☐ | ☐ | ☐ |
| $f(n) = \log(n^4 + \sqrt{n})$ | $g(n) = \log(n^4\sqrt{n})$ | ☐ | ☐ | ☐ | ☐ |
| $f(n) = n^4$ | $g(n) = 4^n$ | ☐ | ☐ | ☐ | ☐ |
| $f(n) = 4^{(n-2)}$ | $g(n) = 4^{(n+1)}$ | ☐ | ☐ | ☐ | ☐ |
| $f(n) = 4^{(3n)}$ | $g(n) = 3^{(4n)}$ | ☐ | ☐ | ☐ | ☐ |

**Exercice 3 :**

On considère les permutations $\sigma = 4\ 5\ 1\ 3\ 7\ 6\ 8\ 2$ (notation linéaire) et $\tau = (1\ 3\ 5)\ (2)\ (4\ 8)\ (6\ 7)$ (notation cyclique).

Donner la représentation en produit de cycles disjoints de $\sigma$ et la représentation linéaire de $\tau$.

Calculer les inverses de $\sigma$ et $\tau$, ainsi que les produits $\sigma\tau$ et $\tau\sigma$ (sous la forme que vous préférez).

Donner deux décompositions différentes de $\sigma$ en produit de transpositions.

**Exercice 4 :**

Démontrer que $\log(n!) \in \Theta(n \log n)$.

**Exercice 5 :**

On s'intéresse au problème suivant : étant donné une liste `L` de $n$ entiers, vérifier que `L` est sans doublon, *i.e.* qu'aucun élément de `L` n'y apparaît plusieurs fois.

Décrire un algorithme naïf permettant de résoudre ce problème.

Indiquer un choix d'opération(s) élémentaire(s) pertinent pour évaluer sa complexité (en temps).

Quel est l'ordre de grandeur de cette complexité ? Justifier.

Comment résoudre ce problème avec une complexité strictement meilleure ? Laquelle ?

**Exercice 6 :**

Soit `T` le tableau suivant :

| 4 | 14 | 9 | 13 | 21 | 8 | 11 | 2 |
|---|---|---|---|---|---|---|---|

Décrire le déroulement de l'algorithme de tri fusion (*MergeSort*) appliqué à `T`. Combien de comparaisons d'éléments sont effectuées (exactement) ?

Mêmes questions pour l'algorithme de tri rapide (*QuickSort*) dans sa version la plus simple (qui ne trie pas en place, avec `T[0]` comme pivot).

**Exercice 7 :**

On considère l'algorithme suivant :

```
from random import randint
def mystere(n) :
  res = [0] * n                # res = [0, 0, ..., 0]
  aux = list(range(1, n+1))    # aux = [1, 2, ..., n]
  for i in range(n) :
    r = randint(0, n-i-1)      # renvoie un entier aléatoire compris entre 0 et n-i-1
    res[i] = aux.pop(r)        # supprime aux[r] de aux et renvoie l'élément supprimé
  return res
```

Combien d'exécutions *différentes* `mystere(n)` peut-il avoir ?

Quel est l'ensemble $\mathcal{R}(n)$ des valeurs de retour possibles de `mystere(n)` ?

Étant donné un élément $R \in \mathcal{R}(n)$, combien d'exécutions différentes produisent le résultat $R$ ?

En supposant que la liste `aux` est représentée par un tableau, décrire un algorithme correspondant à l'appel `aux.pop(r)`. Quelle est sa complexité (en temps) ?

En admettant que l'appel `randint(0, k)` s'exécute en temps indépendant de $k$, en déduire la complexité de `mystere(n)`.

Rappeler l'algorithme vu en cours permettant de résoudre le même problème que `mystere` de manière plus efficace.

Quelle est sa complexité ? Justifier.

**Exercice 8 :**

On considère un tableau `T` de $n$ entiers distincts *circulairement décroissant*, c'est-à-dire tel que, pour un certain indice $i$ (inconnu), `T[`$i$`:] + T[:`$i$`]` est trié en ordre décroissant. Décrire un algorithme aussi efficace que possible pour déterminer la position du minimum de `T`.

Quelle est sa complexité ?