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.

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

Exercice 2 :

Cocher les assertions exactes.

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

Exercice 3 :

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

Donner la représentation en produit de cycles disjoints de σ et la représentation linéaire de τ.

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

Donner deux décompositions différentes de σ 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, déterminer si `L` contient un singleton, *i.e.* si au moins un élément de `L` n'y apparaît qu'une 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 :

17	9	4	14	11	8	13	2

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

Appliquer l'algorithme de tri rapide (*QuickSort*) à `T` dans sa version simple (pas en place, avec `T[0]` comme pivot). Combien de comparaisons d'éléments sont effectuées (exactement) ?

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 une liste chaînée, 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 trié*, c'est-à-dire tel que, pour un certain indice i (inconnu), `T[`i`:] + T[:`i`]` est trié en ordre croissant. Décrire un algorithme aussi efficace que possible pour déterminer la position du minimum de `T`.

Quelle est sa complexité ?