

Nom :
Prénom :

Examen d'algorithmique

EPITA ING1 2017 S1; A. DURET-LUTZ

Durée : 1h30

Janvier 2015

Consignes

- Cette épreuve se déroule **sans document** et **sans calculatrice**.
- Répondez sur le sujet dans les cadres prévus à cet effet.
- Soignez votre écriture, et ne donnez pas plus de détails que ceux nécessaires à vous justifier.
- Il y a cinq pages d'énoncé, assurez-vous de l'avoir en entier.
- Le barème, indicatif, correspond à une note sur 24.

1 Dénombrement (6 pts)

Donnez vos réponses en fonction de N . (On souhaite des formules précises, pas des classes de complexité.)

1. (2 pts) Combien de fois le programme ci-dessous affiche-t-il "x" ?

```
for (int i = 2 * N; i >= 0; --i)
  for (int j = 0; j < i; ++j)
    puts("x");
```

Réponse :

2. (2 pts) Et celui-ci ?

```
for (int i = 3; i < N; ++i)
  for (int j = 1; j < i; ++j)
    puts("x");
```

Réponse :

3. (2 pts) Et celui-ci ?

```
for (int i = 0; i <= N; ++i)
{
    puts("x");
    for (int j = 0; j < i; ++j)
        puts("x");
}
```

Réponse :

2 Gaussons-nous ! (4 pts)

L'algorithme du *pivot de Gauss*, aussi appelé *élimination de Gauss-Jordan*, peut s'écrire comme suit. Notez qu'il n'est pas nécessaire de savoir à quoi sert cet algorithme pour répondre aux questions.

```
1 // input : A[1..n][1..m], a matrix of n rows and m columns
2 for k ← 1 to m do :
3     // Find pivot for column k
4     p ← k
5     for i ← k + 1 to m :
6         if abs(A[i][k]) > abs(A[p,k]) :
7             p ← i
8     if A[p][k] = 0 :
9         error "Matrix is singular"
10    // swap rows k and p
11    A[k] ↔ A[p]
12    // Update all elements below and the pivot
13    for i ← k + 1 to m do :
14        for j ← k to n do :
15            A[i][j] ← A[i][j] - A[k][j] × (A[i][k] / A[k][k])
16        // fill lower triangular matrix with zeroes
17        A[i][k] ← 0
```

Dans les deux questions qui suivent, on suppose que la matrice est carrée ($n = m$) et inversible (la ligne 9 n'est jamais exécutée).

1. (2 pts) Donnez une formule (simplifiée) indiquant exactement combien de multiplications scalaires sont effectuées par cet algorithme (ligne 15) en fonction de n .

Réponse :

2. (2 pts) Quelle est la complexité de cet algorithme en fonction de n (soyez précis dans votre choix de $\Theta(\dots)$ ou $O(\dots)$).

Réponse :

3 Recursion (8 pts)

On considère la fonction suivante qui retourne une liste contenant tous les anagrammes de la chaîne s :

```

1 ANAGRAMS(s) :
2   if s == "" :
3     return [s]
4   res ← [] /* liste vide */
5   for w in ANAGRAMS(s[1 :]) :
6     for pos in {0, 1, ..., |w|} :
7       res.insert(w[: pos] + s[0] + w[pos :])
8   return res

```

$T(0)$	$T(n), n > 0$
$\Theta(1)$	
$\Theta(1)$	

Si $w = "abcde"$, la notation $w[: 3]$ désigne le préfixe de w ne contenant que les lettres 0, 1 et 2, soit $w[: 3] = "abc"$; tandis que $w[3 :]$ désigne le suffixe de w à partir de la lettre 3, ici $w[3 :] = "de"$. En particulier, $w[: 0] = w[5 :] = ""$.

A titre d'exemple, ANAGRAM("foo") retourne ["foo", "fo", "of", "fo", "of", "of"]. Les doublons viennent du fait que la chaîne "foo" contient deux 'o' qui peuvent être permutés : l'algorithme ne fait aucun effort pour éviter cela.

Dans tout cet exercice, on note n la taille de la chaîne s passée à ANAGRAM.

1. (2 pts) Pour une chaîne de taille n , quel est le nombre d'anagrammes retournés par ANAGRAMS ? (Donnez une formule précise, en fonction de n .)

Réponse :

2. (2 pts) Si ANAGRAMS est appelé avec une chaîne de taille $n > 0$, combien de fois la ligne 7 est-elle exécutée lors de cet appel (c'est-à-dire sans compter les exécutions de la ligne 7 lors des appels récursifs effectués ligne 5) ?

Réponse :

3. (2 pts) Justifiez, en annotant les lignes de algorithme par leurs complexités respectives, que la complexité $T(n)$ de ANAGRAMS satisfait l'équation suivante :

$$T(n) = \begin{cases} \Theta(1) & \text{si } n = 0 \\ \Theta(n) \times n! + T(n-1) & \text{si } n > 0 \end{cases}$$

4. (2 pts) Quelle est la solution de l'équation ci-dessus ? (Notez que $i! \times i = i! \times (i+1) - i! = (i+1)! - i!$. Fascinant, non ?)

Réponse :

C'est pas fini...

4 Complexité récursive (6 pts)

Théorème général. Pour une récurrence du type $T(n) = aT(n/b + O(1)) + f(n)$ avec $a \geq 1$, $b > 1$:

- si $f(n) = O(n^{(\log_b a) - \varepsilon})$ pour un $\varepsilon > 0$, alors $T(n) = \Theta(n^{\log_b a})$;
- si $f(n) = \Theta(n^{\log_b a})$, alors $T(n) = \Theta(n^{\log_b a} \log n)$;
- si $f(n) = \Omega(n^{(\log_b a) + \varepsilon})$ pour un $\varepsilon > 0$, **et de plus** $af(n/b) \leq cf(n)$ pour un $c < 1$ et toutes les grandes valeurs de n , alors $T(n) = \Theta(f(n))$.

Pour chacune des définitions récursive de complexité qui suivent, donnez la classe de complexité à laquelle elles appartiennent.

1. $T(n) = 3T(n/3) + \Theta(1)$

Réponse :

2. $T(n) = 2T(n/3) + \Theta(n)$

Réponse :

3. $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(1)$

Réponse :

The End