

TD d'Éléments d'Algorithmique n° 1
(Correction)

Introduction

Exercice 1. *Temps d'exécution.*

Quelle est la plus petite valeur strictement positive de n pour laquelle un algorithme dont le temps d'exécution est $100n^2$ s'exécute plus vite qu'un algorithme dont le temps d'exécution est 2^n sur la même machine ?

Correction : Tatonnement. La réponse est 15.

Exercice 2. *Ordres de grandeur.*

1. Comparer, pour les relations O , o , Ω et Θ , les fonctions $f(n) = \log^k(n)$ et $g(n) = n$ (pour toute constante $k > 0$). De même pour $2^{n/2}$ et 2^n .
2. Si $f \in O(n^a)$ et $g \in O(n^b)$, que peut-on dire de leur produit fg ?
3. Soient f et g deux fonctions à valeurs positives. A-t-on forcément $f \in O(g)$ ou $g \in O(f)$? Qu'en est-il si les deux fonctions sont strictement croissantes ?
4. Pour tout entier $k \geq 0$, montrer que $\sum_{i=1}^n i^k$ est dans $\Theta(n^{k+1})$.

Correction : Son corrigé

1. A écrire.
2. A écrire.
3. A écrire.
4. A écrire.

Exercice 3. *Complexité et opérations.*

1. Évaluer la complexité du bout de programme suivant :

Entrée : un entier n et une instruction en temps constant instr

Sortie : ...

pour i de 1 à n **faire**

pour j de 1 à i **faire**

pour k de 1 à j **faire**

Appliquer l'instruction en temps constant instr

2. On sait qu'un bout de programme a une complexité en $O(f(n))$ et qu'il est exécuté dans une boucle qui est itérée $g(n)$ fois. Quelle est la complexité totale de la boucle ?
3. On sait que deux bouts de programme ont une complexité en $O(f(n))$ et $O(g(n))$ respectivement. Ils sont exécutés séquentiellement l'un après l'autre. Quelle est la complexité totale ?

Correction :

1. A écrire.
2. A écrire.

3. A écrire.

Exercice 4. *Intervalles d'entiers.*

1. Écrivez une fonction `min` qui prend en argument un tableau d'entiers `T` et qui retourne le minimum du tableau.
2. Écrivez une fonction `max` qui prend en argument un tableau d'entiers `T` et qui retourne le maximum du tableau.
3. Écrivez une fonction `contientUnique` qui prend en argument un tableau d'entiers `T` et un entier `x`, et qui retourne 1 si `T` contient `x` une unique fois, et qui retourne 0 sinon.
4. Écrivez une fonction `verifieIntervalle` qui prend en argument un tableau d'entiers `T` et deux entiers `a` et `b` (avec $a \leq b$) et qui retourne 1 si `T` contient chaque entier entre `a` et `b` (`a` et `b` compris) une unique fois, et que `T` ne contient aucun autre entier, et qui vaut 0 sinon. Autrement dit, cette fonction vérifie que `T` correspond à l'intervalle $[a, b]$. Cette fonction utilisera `contientUnique`.
- ★ 5. Écrivez une fonction `verifieIntervalle2` qui prend en argument un tableau d'entiers `T` et qui retourne 1 si `T` correspond à un intervalle (c'est-à-dire s'il existe `a` et `b` tels quel `T` contient chaque entier entre `a` et `b`, une unique fois, mais aucun autre) et qui retourne 0 sinon. Cette fonction utilisera `verifieIntervalle`, `min` et `max`.

Correction :

1. trivial
2. trivial
3. boucle qui compte le nombre d'occurrences de `x` dans `T`. Retourne 1 si le nombre d'occurrences est exactement 1.
4. boucle de `i = a` à `b`, vérifie `contientUnique(T, x) = 1`
5. `verifieIntervalle(T, min(T), max(T))`.

Exercice 5. *Ordre alphabétique.*

Dans cet exercice, on considère qu'un mot (français) est un tableau de caractères, et que l'on peut connaître l'ordre (alphabétique) de deux caractères (en utilisant les opérateurs `<`, `<=`, `=`, `>=` et `>`). Par exemple, on a `'a' < 'b'`, `'b' = 'b'`, `'c' > 'b'`. En revanche, on ne peut pas comparer directement deux mots.

1. Écrivez une fonction `inf` qui prend en argument deux mots et qui retourne -1 si le premier mot est avant le second dans l'ordre alphabétique (ou lexicographique, c'est-à-dire l'ordre du dictionnaire), 0 si les deux mots sont les mêmes, et +1 sinon.
2. Écrivez une fonction `min` (utilisant `inf`) qui prend en argument un tableau de mots et qui retourne le plus petit mot pour l'ordre alphabétique.

Correction :

1. trivial
2. trivial

Exercice 6. *Représentations des parties de $\{1, \dots, n\}$, et opérations.*

En cours, vous avez vu une bijection entre les parties de l'ensemble $\{1, \dots, n\}$ et les suites (ou tableaux) de `n` bits (un bit étant un élément de $\{0, 1\}$). Une partie de $\{1, \dots, n\}$ peut donc être représentée d'au moins deux manières : par la liste de ses éléments, et sous la forme d'un tableau de `n` bits. L'objectif de cet exercice est de voir comment passer d'une représentation à l'autre, et d'entr'apercevoir que certaines représentations sont plus pratiques pour certaines tâches.

1. Écrivez un algorithme qui prend en entrée un entier $n > 0$ et une partie de $\{1, \dots, n\}$ sous la forme d'une *liste de ses éléments*, et qui retourne sa représentation sous la forme d'un *tableau de n bits*.
 2. Écrivez un algorithme qui effectue l'opération inverse, c'est-à-dire qu'il prend en entrée un entier $n > 0$ et une partie de $\{1, \dots, n\}$ sous la forme d'un *tableau de n bits* et qui retourne sa représentation sous la forme d'une *liste de ses éléments*.
 3. Écrivez un algorithme qui prend en entrée un entier $n > 0$ et une partie de $\{1, \dots, n\}$ sous la forme d'un *tableau de n bits*, et qui retourne le complément de cette partie (i.e., l'ensemble des entiers dans $\{1, \dots, n\}$ qui ne sont pas dans la partie), toujours sous la forme d'un *tableau de n bits*.
 4. Écrivez un algorithme similaire mais qui utilise la représentation sous la forme de liste des éléments des parties. Vous proposerez deux solutions : l'une directe et l'autre utilisant les algorithmes des questions précédentes.
- ★ 5. Comparez les deux solutions précédentes en terme de temps de calcul.

Correction :

1. **Entrée :** n , partie de $\{1, \dots, n\}$ sous la forme d'une liste de ses éléments

Sortie : un tableau T de bits représentant la même partie

$T \leftarrow$ tableau de n bits initialisés à 0

pour tout j dans L **faire**

$T[j] \leftarrow 1$

retourne T

2. **Entrée :** n , partie de $\{1, \dots, n\}$ sous la forme d'un tableau T de bits

Sortie : une liste L représentant la partie

$L \leftarrow$ liste vide

pour i de 1 à n **faire**

si $T[i - 1] = 1$ **alors**

Ajouter i à la fin de L

retourne L

3. **Entrée :** n , partie de $\{1, \dots, n\}$ sous la forme d'un tableau T de bits

Sortie : le complément de cette partie sous la même forme

$T' \leftarrow$ tableau de n bits initialisés à 0

pour i de 1 à n **faire**

$T'[i - 1] \leftarrow 1 - T[i - 1]$

retourne T'

4. • Première solution :

Entrée : n , partie de $\{1, \dots, n\}$ sous la forme d'une liste L

Sortie : le complément de cette partie sous la même forme

$L' \leftarrow$ liste vide

pour i de 1 à n **faire**

$flag \leftarrow 0$

pour tout j dans L **faire**

si $i = j$ **alors**

$flag \leftarrow 1$

si $flag$ vaut 0 **alors**

$L' \leftarrow i$

retourne L'

- Seconde solution : on convertit la liste des éléments en un tableau de bits (question 1), puis on calcule le complément de la partie sous cette représentation (question 3), et enfin on convertit le résultat obtenu sous la forme d'une liste d'éléments (question 2).

5. La seconde solution effectue $\Theta(n)$ opérations, tandis que la première effectue $\Theta(n^2)$. La seconde solution est donc plus efficace.

Exercice 7. Médiane.

La médiane d'un tableau T à n éléments distincts avec n impair, est l'élément x de T tel que exactement $(n - 1)/2$ éléments de T sont $< x$.

1. Écrivez une fonction `countInf` qui prend en argument un tableau d'entiers T et un entier x , et qui retourne le nombre d'éléments de T qui sont $< x$.
2. Écrivez un algorithme (utilisant cette fonction) qui trouve la médiane d'un tableau T à n éléments distincts avec n impair.
- ★ 3. Quelle est la complexité de cet algorithme ?

Correction :

1. trivial
2. boucle sur les éléments de T jusqu'à trouver la médiane.
3. $\Theta(n^2)$