EPFL - Information, Calcul et Communication - Partie théorique

Exemples d'exercices: solutions

Exercice 1. a) oui

b) L'algorithme **mystère** sort oui si et seulement si le nombre x se trouve dans le tableau A.

c) (BONUS) Si x fait partie du tableau A, l'algorithme le trouvera forcément, pour la raison suivante: à tout moment, si l'algorithme a atteint la position (i, j) dans le tableau, alors tous les nombres du tableau au-dessus de la position (i, j) (i.e., ceux sur une ligne $k < i$) sont plus grands que x et tous ceux à gauche de la position (i, j) (i.e., ceux sur une colonne $\ell < j$) sont plus petits que x.

Vous pouvez vérifier que c'est manifestement vrai au départ de l'algorithme (i.e., en position $(1, 1)$), et aussi que si c'est vrai à un moment donné de l'exécution de l'algorithme en une position (i, j) donnée, alors ça le reste le moment d'après. En effet:

- si $A(i, j) < x$, alors tous les nombres en-dessous de la position (i, j) sont également inférieurs à x (car $A(k, j) \leq A(i, j) < x$ pour tout $k > i$), et dans ce cas, l'algorithme se décale vers la droite en position $(i, j + 1)$: il reste donc bien vrai que tous les nombres à gauche de la position $(i, j + 1)$ sont plus petits que x.

- si $A(i, j) > x$, alors tous les nombres à droite de la position (i, j) sont supérieurs à x (car $A(i, \ell) \geq A(i, j) > x$ pour tout $\ell > j$), et dans ce cas, l'algorithme se décale vers le bas en position $(i + 1, j)$: il reste donc bien vrai que tous les nombres au-dessus de la position $(i + 1, j)$ sont plus grands que x.

Ainsi l'algorithme parcourera tout le tableau, finissant éventuellement dans le coin en bas à droite (i.e., en position (n, m)) sans avoir trouvé la valeur x.

d) $\Theta(n)$

e) également $\Theta(n)$

f) Oui, c'est possible. Voici un algorithme de complexité temporelle $\Theta(\log_2(n))$ qui fonctionne, car les deux lignes du tableau $A(1)$ et $A(2)$ sont chacune triées dans l'ordre croissant:

algorithme

entrée : *tableau A de $2 \times n$ nombres entiers, nombre entier x*
sortie : *valeur binaire (oui/non)*

$s_1 \longleftarrow$ **recherche dichotomique**$(A(1), n, x)$
$s_2 \longleftarrow$ **recherche dichotomique**$(A(2), n, x)$
Si $s_1 =$ *oui ou* $s_2 =$ *oui*
  Sortir: *oui*
Sinon
  Sortir: *non*

Exercice 2. a) La sortie vaut 5.

b) La sortie ne change pas.

c) La complexité temporelle est un $\Theta(\log_2(n))$.

d) Voici une possibilité:

mystère

entrée : *deux listes L_1, L_2 de nombres entiers positifs, toutes deux de taille $n = 2^k$ avec $k \geq 0$, et toutes deux ordonnées dans l'ordre croissant*
sortie : *nombre entier positif*

```
a ⟵ 1
b ⟵ n
c ⟵ 1
d ⟵ n
Tant que b > a et d > c
    m1 ⟵ ⌊(a+b)/2⌋
    m2 ⟵ ⌊(c+d)/2⌋
    Si L1(m1) > L2(m2)
        b ⟵ m1
        c ⟵ m2 + 1
    Sinon
        a ⟵ m1 + 1
        d ⟵ m2
Sortir: (L1(a)+L2(c))/2
```

ou une encore une version plus simple:

mystère

entrée : *deux listes L_1, L_2 de nombres entiers positifs, toutes deux de taille $n = 2^k$ avec $k \geq 0$, et toutes deux ordonnées dans l'ordre croissant*
sortie : *nombre entier positif*

```
Tant que n > 1
    m ⟵ n/2
    Si L1(m) > L2(m)
        L1 ⟵ L1(1 : m)
        L2 ⟵ L2(m + 1 : n)
    Sinon
        L1 ⟵ L1(m + 1 : n)
        L2 ⟵ L2(1 : m)
    n ⟵ m
Sortir: (L1(1)+L2(1))/2
```

Exercice 3. a) Un algorithme possible est le suivant:

algo

entrée : *liste L de nombres entiers positifs, de taille n, nombre entier positif x*
sortie : *oui/non*

```
y ⟵ max(L(1), L(2))
z ⟵ min(L(1), L(2))
Pour i allant de 3 à n
    Si L(i) ≥ y
        z ⟵ y
        y ⟵ L(i)
    Sinon, si L(i) ≥ z
        z ⟵ L(i)
Si y + z ≥ x (Note: y et z sont les deux plus grands nombres de la liste L)
    Sortir : oui
Sinon
    Sortir : non
```

b) Oui. Si on nous propose un sous-ensemble S comme solution, il suffit alors d'effectuer la somme $\sum_{j\in S} L(j)$ et de comparer celle-ci à la valeur x (ce qui se fait en temps polynomial, même en $\Theta(n)$).

c) Oui, il fait partie de la classe P: il suffit de parcourir une seule fois la liste L (complexité $\Theta(n)$) et de ne retenir que les nombres positifs de celle-ci. Si leur somme est plus grande ou égale à x, la réponse est oui; sinon, la réponse est non.

Exercice 4. a) Deux possibilités. La première en $\Theta(n)$:

algorithme

entrée : *Listes L_1, L_2 ordonnées dans l'ordre croissant, chacune de taille n*
sortie : *oui/non*

```
i ⟵ 1
j ⟵ 1
Tant que i ≤ n et j ≤ n
    Si L1(i) = L2(j)
        Sortir : oui
    Si L1(i) > L2(j)
        j ⟵ j + 1
    Sinon
        i ⟵ i + 1
Sortir : non
```

La seconde en $\Theta(n \log_2(n))$:

algorithme

entrée : *Listes L_1, L_2 ordonnées dans l'ordre croissant, chacune de taille n*
sortie : *oui/non*

Pour *i allant de 1 à n*
  $s \longleftarrow$ **recherche dichotomique**$(L_2, n, L_1(i))$
  Si *$s =$ oui*
    Sortir : *oui*

Sortir : *non*

b) Oui, c'est possible. Il faut appliquer la seconde solution ci-dessus:

algorithme

entrée : *Listes L_1, L_2 ordonnées dans l'ordre croissant, de tailles n et 2^n, respectivement*
sortie : *oui/non*

Pour *i allant de 1 à n*
  $s \longleftarrow$ **recherche dichotomique**$(L_2, 2^n, L_1(i))$
  Si *$s =$ oui*
    Sortir : *oui*

Sortir : *non*

La complexité temporelle de cet algorithme est en $\Theta(n^2)$.

Exercice 5. a) 6

b) 5

c) Le nombre de bits nécessaires pour représenter le nombre entier positif n; de manière équivalente: $\lceil \log_2(n+1) \rceil$ ou encore $\lfloor \log_2(n) \rfloor + 1$.

d) $\Theta(\log_2(n))$ [Dans le pire des cas, n est est divisé par 2 après 2 étapes de l'algorithme.]

Exercice 6. a) la sortie vaut 1 dans ce cas.

b) A chaque étape, l'algorithme **bidule** ou **machin** effectue $\Theta(1)$ opérations et appelle l'autre algorithme avec une valeur de n décrémentée de 1, jusqu'à la condition de terminaison où $n = 1$. La complexité est donc $\Theta(n)$.

c)

machin_machin

entrée : *nombre entier positif n*
sortie : *nombre entier positif s*

Si *$n = 1$ ou 2*
  Sortir: *n*

Sortir : $2*$**machin_machin**$(n-2)$

d) Si n est pair, $s = 2^{n/2}$. Si n est impair, $s = 2^{(n-1)/2}$. En général: $s = 2^{\lfloor n/2 \rfloor}$.