

ULIN501  
Examen 14 novembre 2006

---

Les documents sont autorisés. La note prendra en compte la clarté des explications. Il est conseillé d'effectuer les exercices dans l'ordre.

Exercice 1

1. Déterminer la complexité des algorithmes 1 2 3 et 4 suivants (par rapport au nombre d'itérations effectuées), où  $m$  et  $n$  sont deux entiers positifs.

---

Algorithm 1 algorithm 1

```
i := 1; j := 1;
while i ≤ m et j ≤ n do
  i := i + 1
  j := j + 1
end while
```

---

Algorithm 2 algorithm 2

```
i := 1; j := 1;
while i ≤ m ou j ≤ n do
  i := i + 1
  j := j + 1
end while
```

---

Algorithm 3 algorithm 3

```
i := 1; j := 1;
while j ≤ n do
  if i ≤ m then
    i := i + 1
  else
    j := j + 1
  end if
end while
```

□

---

Correction exercice 0

---

Algorithm 4 algorithm 4

```
i := 1; j := 1;
while j ≤ n do
  if i ≤ m then
    i := i + 1;
  else
    j := j + 1;
    i := 1;
  end if
end while
```

---

Le corps des différents algorithmes ne contient que des opérations en  $O(1)$ . En conséquence le nombre d'itérations effectuées par chacun des algorithmes, on obtient immédiatement :

1. l'algorithme  $A$  est en  $O(\min(m, n))$ . En effet, les itérations s'arrêteront lorsque  $i > m$  ou  $j > n$  et que à chaque itération  $i$  et  $j$  sont incrémentés de 1.
2. l'algorithme  $B$  est en  $O(\max(m, n))$ . En effet, les itérations s'arrêteront lorsque  $i > m$  et  $j > n$  et que à chaque itération  $i$  et  $j$  sont incrémentés de 1.
3. l'algorithme  $C$  est en  $O(m + n)$ , car tant que  $i \leq m$  la boucle passera dans le if ( $m$  fois). Et lorsque  $i > m$  la boucle passera  $n$  fois dans le else.
4. l'algorithme  $D$  est en  $O(mn)$  car à chaque  $i$  est réinitialisé à 1, la boucle passera  $m$  fois dans le if=; Or  $i$  est initialisé à 1 tant que  $j \leq n$  c'est à dire  $n$  fois.

---

Fin correction exercice 0

Exercice 2

1. Ecrire un algorithme *nombrediviseurs* efficace qui donne le nombre de diviseurs de  $n$  et évaluer sa complexité. En déduire un algorithme premier? pour savoir si un nombre est premier.

□

---

Correction exercice 0

NombreDiviseurs( $n$ ) //  $n$  est strictement positif

$NbDiv \leftarrow 0$

Pour  $i$  de 1 à  $\lfloor \sqrt{n} \rfloor$  si  $n \bmod i = 0$  alors  $NbDiv \leftarrow NbDiv + 2$   
si  $n = (\sqrt{n})^2$  alors  $NbDiv \leftarrow NbDiv - 1$

Renvoyer  $NbDiv$

Complexité dans  $\Theta(\lfloor \sqrt{n} \rfloor)$

**Premier(n)** // n est strictement positif, 1 n'est pas premier

**Renvoyer** *NombreDiviseurs(n)* = 2

---

**Fin correction exercice 0**

---

**Exercice 3**

Soit la fonction suivante :

On recherche un élément  $X$  dans un tableau  $T$  de  $n$  éléments, trié par ordre croissant. On suppose que  $X$  est présent dans  $T$  et qu'il peut-être dans la case  $i$  avec une probabilité  $1/n$ . Voici deux versions (algorithmes 5 et 6) de la recherche dichotomique.

---

**Algorithm 5** Version1

---

```
i := 1; j := n; m := (i + j)div2;
while i ≠ j do
  if  $X > T[m]$  then
    i := m + 1
  else
    j := m
  end if
  m := (i + j)div2;
end while
renvoyer i;
```

---

---

**Algorithm 6** Version2

---

```
i := 1; j := n; m := (i + j)div2;
while i ≠ j do
  if  $X = T[m]$  then
    renvoyer m
  else if  $X > T[m]$  then
    i := m + 1
  else
    j := m - 1
  end if
  m := (i + j)div2;
end while
renvoyer i;
```

---

1. Pour chacune d'elles donner le nombre de comparaisons entre éléments dans le meilleur des cas, le pire des cas et en moyenne.

2. Conclure

3. Indication : pour simplifier les calculs, on supposera que  $n = 2^k$  pour la version 1 et  $n = 2^k - 1$  pour la version 2.

□

---

**Correction exercice 0**

---

1. Pour la version 1

- Pire des cas : Nous devons résoudre le l'équation suivante :

$$\begin{aligned} \max(1) &= 1 \\ \max(n) &= 1 + \lceil \max(n/2) \rceil \end{aligned}$$

Nous obtenons  $\max(n) = \lceil \log_2 n \rceil + 1$ .

- Meilleur des cas : Nous devons résoudre le l'équation suivante :

$$\begin{aligned} \min(1) &= 1 \\ \min(n) &= 1 + \lfloor \min(n/2) \rfloor \end{aligned}$$

Nous obtenons  $\min(n) = \lfloor \log_2 n \rfloor + 1$ .

- Moyenne des cas : ainsi la moyenne  $1 + \lfloor \log_2 n \rfloor \leq \text{moy}(n) \leq \lceil \log_2 n \rceil + 1$ . Dans le cas où  $n = 2^k$  alors nous avons  $\max_c(n) = \min_c(n) = \text{moy}_c(n) = \log_2 n$ .

2. Pour la version 2

- Pire des cas :  $\max(n) = 2 \lfloor \log_2 n \rfloor + 1$

- Meilleur des cas : une seule comparaison  $\min(n) = 1$

- Moyenne des cas :

On calcul le nombre de comparaisons si on cherche tous les éléments du tableau un par un et on divise par  $n$ .

- Pour 1 élément, on fait 1 comparaison,

- Pour 2 éléments, on fait 3 comparaisons,

- Pour 4 éléments, on fait 5 comparaisons,

- Pour  $2^i$  éléments, on fait  $2i + 1$  comparaisons,

- Pour  $2^k - 2$  éléments, on fait  $2k - 3$  comparaisons,

- Pour  $2^k - 1$  éléments, on fait  $2k - 2$  comparaisons (la dernière itération n'a pas lieu)

Si on somme on obtient :  $\sum_{i=0}^{k-1} 2^i(2i + 1)$  (à cette somme on doit retirer  $2^{k-1}$  i.e un terme en  $O(1)$ ).

En posant  $f(x) = \sum_{i=0}^{k-1} x^i$  avec  $k - 1 = (x^k - 1)/(x - 1)$ , on remarque que la somme précédente est égale à  $4f'(2) + f(2)$  comme  $f(2) = n$  et  $4f'(2) = 2 \log 2(n + 1) + O(n)$

On obtient  $\text{Moy}_c(n) = 2 \log 2(n + 1) + O(1)$ . La moyenne est donc à une constante près égale à  $\max_c(n)$ .

Conclusion : la version 1 est meilleure.

---

**Fin correction exercice 0**

---

**Exercice 4** Soient  $a$  et  $b$  deux entiers vérifiant  $a \geq b$ . Lorsque  $b = 0$ , le pgcd de  $a$  et  $b$  est égal à  $a$ . Lorsque  $b > 0$ , on a  $\text{pgcd}(a, b) = \text{pgcd}(b, a \bmod b)$ .

1. Ecrire un algorithme récursif qui calcule le pgcd de  $a$  et  $b$ .
2. Calculer  $\text{pgcd}(34, 21)$  et  $\text{pgcd}(36, 24)$  en utilisant l'algorithme précédent.
3. Montrer cet algorithme calcule  $\text{pgcd}(a, b)$  en moins de  $2 * \log_2 a$  appels récursifs, lorsque  $a > 1$ .

□

---

**Correction exercice 0**

---

1. L'algorithme :

$\text{pgcd}(a, b)$

si  $b = 0$  retourner  $a$

sinon retourner  $\text{pgcd}(b, a \bmod b)$

2.  $\text{pgcd}(34, 21) = \text{pgcd}(21, 13) = \text{pgcd}(13, 8) = \text{pgcd}(8, 5) = \text{pgcd}(5, 3) = \text{pgcd}(3, 2) = \text{pgcd}(2, 1) = \text{pgcd}(1, 0) = 1$   
 $\text{pgcd}(36, 12) = \text{pgcd}(24, 12) = \text{pgcd}(12, 0) = 12$

3. Soit  $f(a)$  le nombre maximum d'appels récursifs qu'effectue  $\text{pgcd}(a, b)$ . On veut montrer que  $f(a) \leq 2 \log_2(a)$  lorsque  $a > 1$ .

On initialise l'induction en vérifiant que lorsque  $a = 2$ , le nombre d'appels est au plus de 1 (0 si  $b = 0$  et 1 si  $b = 1$  ou 2). Si  $a = 3$ , le nombre d'appels est au plus 2.

Pour l'étape inductive, il suffit de montrer qu'après deux appels récursifs, la valeur de  $a$  a été divisée au moins par 2.

En effet,  $\text{pgcd}(a, b) = \text{pgcd}(b, a \bmod b) = \text{pgcd}(a \bmod b, b \bmod (a \bmod b))$ , donc on veut vérifier que  $a/2 \geq (a \bmod b)$ . C'est vrai si  $b \leq a/2$ , car  $a \bmod b$  est inférieur à  $b$ . C'est aussi vrai si  $b > a/2$  car  $a \bmod b$  est inférieur ou égal à  $a - b$ .

Pour résumer,  $f(i) \leq 2 \log_2(i)$  pour  $i = 2, 3$  et  $f(i) \leq f(i/2) + 2$ .

Ainsi si l'on suppose que  $f(b) \leq 2 \log_2(b)$  pour tout  $b < a$ , on a  $f(a) \leq 2 + f(a/2) \leq 2 + 2 \log$

---

**Fin correction exercice 0**

---