

## Diviser pour régner

### Exercice 1

Dans le Chapitre “Grandeur des fonctions” du livre “Introduction à l’algorithmique” de T. Cormen, C. Leiserson et R. Rivest, il est introduit la notion de  $o(g)$ .

1. Rappeler la définition de  $o(g)$ .
2. Soient  $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$  deux fonctions quelconques. Prouver ou infirmer les propositions suivantes :
  - (a) Si  $f \in o(g)$  alors  $f \in O(g)$ .
  - (b) Si  $f \in O(g)$  alors  $f \in o(g)$ .
  - (c) Si  $f \in o(g)$  alors  $f \notin \Omega(g)$ .
  - (d) Si  $f \notin \Omega(g)$  alors  $f \in o(g)$ .

### Exercice 2 (Rappels sur les logarithmes)

1. Montrer que  $a^{\log_b n} = n^{\log_b a}$  pour  $a, b, n > 0$ .
2. Montrer que  $\log_c m = \frac{\log_s m}{\log_s c}$  pour  $c, m, s > 0$ .

## Concevoir des solutions basées sur la méthode étudiée

### Exercice 3

On suppose que l’on dispose des trois algorithmes suivants.

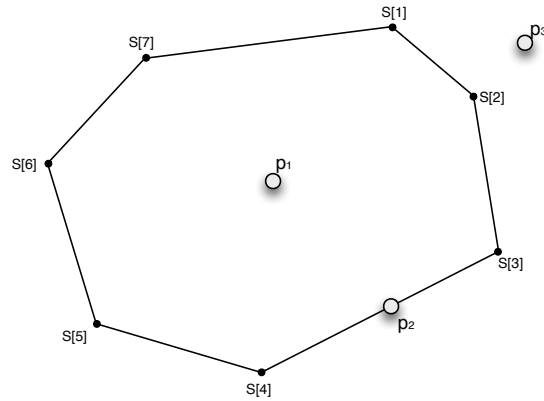
- Algorithme  $A$  résout des problèmes de taille  $n$  en les divisant en 5 sous-problèmes de taille  $\frac{n}{2}$ , en les résolvant chacun récursivement, puis en combinant les solutions en temps linéaire.
- Algorithme  $B$  résout des problèmes de taille  $n$  en les divisant en 2 sous-problèmes de taille  $n - 1$ , en les résolvant chacun récursivement, puis en combinant les solutions en temps constant.
- Algorithme  $C$  résout des problèmes de taille  $n$  en les divisant en 9 sous-problèmes de taille  $\frac{n}{3}$ , en les résolvant chacun récursivement, puis en combinant les solutions en temps en  $O(n^3)$ .

Quels sont les temps d’exécution (en notation grand-O) de chacun de ces algorithmes, et lequel choisiriez-vous ?

### Exercice 4 (Point dans un polygone)

On considère un polygone convexe (angles internes  $< 180^\circ$ ) à  $n$  sommets ( $n \geq 3$ ), illustré par la figure 1. Un point est désigné par ses coordonnées  $(x, y)$  et le polygone est défini par un tableau  $S$  de  $n$  points tel que  $S[n]S[1]$  soit un côté du polygone et pour tout  $1 \leq i < n$ ,  $S[i]S[i + 1]$  soit également un côté.

On veut déterminer si un point  $p$  donné du plan est inclus (au sens large, c’est-à-dire en acceptant que le point soit sur le périmètre) dans le polygone  $S[1 \dots n]$ . On souhaite donc résoudre le problème suivant :

**Problème POINTDANSPOLYGONE****Entrée :** un point  $p$  et un polygone  $S$ **Sortie :** la valeur de l'affirmation "Le point  $p$  est inclus dans le polygone  $S$ ."Figure 1: Un polygone convexe à 7 côtés.  $p_1$  et  $p_2$  y sont inclus,  $p_3$  ne l'est pas.

On suppose donnée la fonction  $même-côté(A, B, X, Y)$  qui retourne *vrai* si les points  $X$  et  $Y$  sont situés du même côté (au sens large) par rapport à la droite  $(AB)$  avec  $A$  et  $B$  distincts.

On supposera qu'un appel à la fonction  $même-côté$  a une complexité temporelle en  $O(1)$ .

1. Écrire une fonction  $dans-triangle(A, B, C, p)$  déterminant si le point  $p$  est inclus dans le triangle formé par les points  $A$ ,  $B$ , et  $C$ .
2. Écrire une fonction  $inclus1$  qui utilise la fonction  $dans-triangle(A, B, C, p)$  et qui procède par réduction simple, et dont l'appel  $inclus1(S, p)$  retourne la valeur de l'assertion "le point  $p$  est inclus dans le polygone  $S$ ".

Quel est l'ordre de grandeur de la complexité temporelle de votre algorithme pour  $inclus1$  ?

3. Écrire une fonction  $inclus2$  qui utilise la fonction  $dans-triangle(A, B, C, p)$  et qui procède par réduction logarithmique, dont l'appel  $inclus2(S, p)$  retourne la valeur de l'assertion "le point  $p$  est inclus dans le polygone  $S$ ".

Quel est l'ordre de grandeur de la complexité temporelle de votre algorithme pour  $inclus2$  ?

**Exercice 5 (Pavage avec des triominos)**

On appelle *table à trou* de dimension  $n > 0$  un quadrillage  $n \times n$  auquel on a retiré une case, comme illustré sur la figure 2.

On dispose de pièces, appelées *triominos* dont la forme est obtenue à partir d'une table  $2 \times 2$  à laquelle on a retiré une case. Comme n'importe quelle case peut être retirée, il existe quatre triominos, illustrés par la Figure 3. On souhaite écrire un algorithme qui résout le problème suivant:

**Problème PAVAGETRIOMINOS****Entrée :** une table à trou  $T$ **Sortie :** un pavage de  $T$  avec des triominos

1. On note  $Pav(T)$  la propriété "la table à trou  $T$  peut être pavée". Montrer que la propriété  $Pav(T)$  est vraie pour toute table à trou  $T$  de dimension  $n = 2^r$  (où  $r \in \mathbb{N}$ ) (*Indication* : on procèdera par récurrence sur  $r$ ).

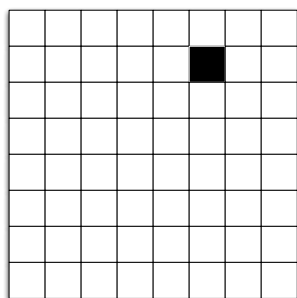
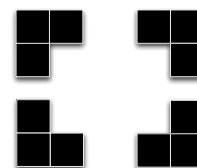
Figure 2: Une table à trou de taille  $8 \times 8$ 

Figure 3: Les quatre triominos

2. Proposer une méthode fondée sur le principe DR permettant de résoudre le problème PAVAGETRIOMINOS restreint à des entrées de dimension  $2^r$  (où  $r \in \mathbb{N}$ ).
3. Dans le cas général (la dimension des entrées du problème PAVAGETRIOMINOS est arbitraire, et plus nécessairement une puissance de 2), quelle propriété sur  $n$  est nécessaire pour que le problème admette une solution ? En déduire, lorsque le problème admet une solution, le nombre de triominos utilisés.

### Exercice 6 (Élément majoritaire)

Soit  $E$  une liste de  $n$  éléments rangés dans un tableau numéroté de 1 à  $n$ . On dispose d'une seule opération qui permet de tester si deux éléments sont égaux. On dit qu'un élément  $x \in E$  est *majoritaire* si l'ensemble  $E_x = \{y \in E \mid y = x\}$  est de cardinal strictement plus grand que  $n/2$ .

1. Montrer qu'un élément majoritaire est nécessairement unique (*Indication* : on pourra raisonner par l'absurde).

On suppose dans toute la suite que  $n$  est une puissance de 2, et on s'intéresse à la complexité dans le pire des cas.

2. Écrire une fonction  $Occurrences(x, E)$  qui calcule le nombre d'occurrences de  $x$  dans le tableau  $E$ .
3. En déduire un algorithme naïf pour vérifier si  $E$  possède un élément majoritaire. Quelle est la complexité de cet algorithme ?
4. On coupe  $E$  en deux tableaux  $E_1$  et  $E_2$  de tailles  $n/2$ . Montrer que tout élément majoritaire dans  $E$  est un élément majoritaire dans  $E_1$  ou dans  $E_2$ .
5. Écrire un algorithme récursif  $Majoritaire(E)$  qui renvoie le couple  $(x, c)$  si  $x$  est majoritaire dans le tableau  $E$  avec  $c$  occurrences et qui vaut  $(-, 0)$  s'il n'y a pas d'élément majoritaire dans  $E$ .
6. Donner la complexité de l'algorithme  $Majoritaire(E)$  dans le pire cas.

### Exercice 7 (Les tours de Hanoï)

Les tours de Hanoï est un jeu composé de  $n$  disques de diamètres distincts et de trois tiges verticales ( $A, B$ , et  $C$ ) posées sur un socle. Dans l'état initial, comme illustré dans la Figure 4 pour  $n = 4$ ,

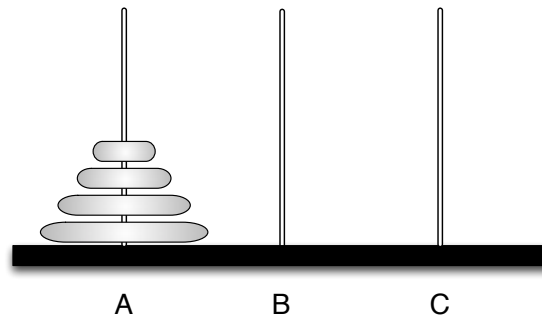


Figure 4: Les tours de Hanoï ( $n = 4$ , état initial).

les disques sont disposés autour de la même tige (que l'on nommera  $A$ ), empilés dans l'ordre décroissant de leur diamètre.

Le but du jeu est de déplacer l'ensemble des disques de la tige  $A$  à la tige  $C$  en respectant les contraintes suivantes:

- La seule opération autorisée est le déplacement d'un disque d'une tige vers une autre.
- Un disque ne doit jamais reposer sur un disque de diamètre inférieur.

1. Ecrire un algorithme récursif permettant de résoudre ce problème.
2. Donner sa complexité.

---

### Exercice 8

Un arbre binaire est dit *plein* si tous ses noeuds ont soit 0 soit 2 fils. Soit  $B_n$  le nombre d'arbres pleins avec  $n$  noeuds.

1. En dessinant tous les arbres pleins à 3, 5 ou 7 noeuds, déterminer les valeurs exactes de  $B_3$ ,  $B_5$ ,  $B_7$ . Pourquoi n'avons-nous pas considéré les nombres de noeuds pairs comme  $B_4$  ?
2. Pour un  $n$  quelconque, dériver une relation de récurrence pour  $B_n$ . En déduire une relation de récurrence pour  $C_n$  tel que  $C_n = B_{2n+1}$ .
3. Montrer que  $C_n$  est en  $\Omega(2^n)$ .