

**DS1 - documents de cours, TD, TP autorisés**

Note : lorsque, dans une question, est demandée la complexité, c'est une fonction de la taille de la donnée qui est attendue. Lorsque c'est le comportement asymptotique, la réponse attendue est soit une notation  $\mathcal{O}$ , soit une notation  $\Theta$  soit une notation  $\Omega$  (le choix vous incombe).

Vous pourrez utiliser toute fonction vue en cours, TD, TP à la condition expresse de spécifier correctement les entrées et les sorties.

Le barème est donné à titre indicatif.

**Exercice 1 : Questions de cours [3,25 points] (réponses sans justification)**

**Q 1.1 [0.75 point]** Si je prouve qu'un algorithme est en  $\Omega(n^2)$ , peut-il être en  $\Theta(n)$  sur certains exemplaires? Répondre par oui ou non.

*Corrigé*

Non bien sûr. Puisque l'algorithme est en  $\Omega(n^2)$  il s'exécute dans le meilleur des cas en  $\Theta(n^2)$ , il ne peut donc pas y avoir d'instance qui s'exécutent en  $\Theta(n)$  puisque  $n < n^2$ .

**Q 1.2 [0.75 point]** Si je prouve qu'un algorithme est en  $\mathcal{O}(\sqrt{n})$ , peut-il être en  $\Theta(n)$  sur certains exemplaires? Répondre par oui ou non.

*Corrigé*

Non. Puisque l'algorithme est en  $\mathcal{O}(\sqrt{n})$  il s'exécute dans le pire des cas en  $\Theta(\sqrt{n})$ , il ne peut y avoir certaines instances qui s'exécutent en  $\Omega(n)$  puisque  $n > \sqrt{n}$ .

**Q 1.3 [0.75 point]** Un algorithme qui utilise la stratégie diviser pour régner s'exécutera d'autant plus vite qu'il divise le problème en un plus grand nombre de sous-problèmes. Répondre par vrai ou faux. *Corrigé*

Faux. Nous en avons vu un exemple avec la fusion à 4.

**Q 1.4 [1 point]** Le nombre de basculements de bits nécessaires pour incrémenter un compteur binaire  $n$  de  $k$  bits est (donner toutes les réponses correctes) :

- (a)  $\mathcal{O}(\log n)$    (b)  $\mathcal{O}(k)$    (c)  $\Theta(\log k)$    (d)  $\Theta(n)$    ?

*Corrigé*

- (a) puisque le nombre de bits pour représenter  $n$  est  $\log n$   
 (b) puisque  $k$  est le maximum de bits à basculer possible

**Exercice 2 : Tableaux unimodaux [2 points]**

On dit qu'un tableau est *unimodal* si il peut être découpé en une suite d'éléments croissants suivi d'une suite d'éléments décroissants.

Par exemple  $[1; 3; 4; 10; 14; 10; 9; 2]$  est unimodal puisque que  $1; 3; 4; 10; 14$  forme une suite croissante et  $10; 9; 2$  une suite décroissante.

On donne la fonction suivante qui retourne l'élément d'indice maximum (le plus à droite du tableau) tel que les éléments suivants forment une séquence décroissante.

```
(* CU: t contient au moins un element *)
let rec unimodal t =
  let n = Array.length t
  in
  if n = 1 then
    t.(0)
  else begin
    let m = ((n-1) / 2)
    in
    if t.(m) < t.(m+1) then
      unimodal (Array.sub t (m+1) (n-(m+1)))
    else
      unimodal (Array.sub t 0 (m+1))
    end
  end
```

```
# unimodal [|1;3;4;10;14;10;9;2|];;
- : int = 14
# unimodal [|13;1;11;2;9;8;7;2|];;
- : int = 9
```

**Q 2.1** [1 point] Etablir l'équation de récurrence  $c(n)$  qui calcule le nombre de comparaisons de valeurs du tableau  $t$ .

*Corrigé*

$$c(n) = \begin{cases} 0 & \text{si } n = 1 \\ 1 + c(\frac{n}{2}) & \text{sinon} \end{cases}$$

**Q 2.2** [1 point] Donner le comportement asymptotique de  $c(n)$ . Justifier. *Corrigé*

C'est une équation de partition.  $a = 1, b = 2, \log_b a = 0, f(n) = 1$ .  $f(n) = 1 = \Theta(n^0)$  donc on est dans le cas 2 du théorème général. Ainsi  $c(n) = \Theta(n^0 \log n) = \Theta(\log n)$ .

### Exercice 3 : Listes mélangées [4 points]

On considère des listes avec les définitions de type suivantes :

```
type liste = Vide | Cons of cellule
and cellule = {
  valeur : int;
  mutable suivant : liste;
}
```

On considère disposer d'une fonction `la_cellule` qui est un accesseur à la cellule en tête de liste, lève une exception `ListeVide` si la liste est vide.

On souhaite réaliser une opération de mélange de deux listes  $\ell_1$  et  $\ell_2$  qui consiste à construire une liste telle que le premier élément sera le premier élément de  $\ell_1$ , le second le premier élément de  $\ell_2$ , le troisième le deuxième de  $\ell_1$ , le quatrième le deuxième de  $\ell_2$ , et ainsi de suite.

Pour simplifier, on ne mélangera que des listes possédant le même nombre d'éléments.

Dans l'exercice on se donne comme contrainte que l'espace mémoire utilisé sera constant.

**Q 3.1** [0.5 point] Représenter le chaînage de deux listes à trois éléments puis le chaînage de la liste résultant de leur mélange.

**Q 3.2** [2 points] Ecrire la **procédure** de mélange qui prend en entrée les listes  $\ell_1$  et  $\ell_2$  et modifie  $\ell_1$  pour qu'elle soit le mélange des deux. *Corrigé*

```

let melanger l1 l2 =
  let p1 = ref l1
  and p2 = ref l2
  and p = ref l1
  and i = ref 0
  in
  try
    while true do
      if !i mod 2 == 0 then begin
        p := (la_cellule !p1).suivant;
        (la_cellule !p1).suivant <- !p2;
        p1 := !p;
      end else begin
        p := (la_cellule !p2).suivant;
        (la_cellule !p2).suivant <- !p1;
        p2 := !p;
      end;
      i := !i + 1;
    done;
  with
  | ListeVide -> ()
;;

```

Ou une version récursive :

```

let rec melanger l1 l2 =
  try
    let l2s = (la_cellule l2).suivant
    and l1s = (la_cellule l1).suivant
    in
    (la_cellule l2).suivant <- (la_cellule l1).suivant;
    (la_cellule l1).suivant <- l2s;
    melanger l1s l2s;
  with
  | ListeVide -> ()
;;

```

où l'appel récursif peut être réalisé avant ou après la modification des suivants. Si c'est fait avant, on peut se passer des deux variables.

**Q 3.3** [0.5 point] Justifier que la complexité en espace est en  $\Theta(1)$ .

**Q 3.4** [1 point] Donner le nombre d'affectations de `liste` en fonction de  $n$ , le nombre d'éléments de  $\ell_1$  et  $\ell_2$ .

*Corrigé*

(version itérative) Chaque cellule de  $\ell_1$  voit son suivant modifié. Chaque cellule de  $\ell_2$  sauf la dernière voit son suivant modifié. L'exception se lève lorsqu'on accède au suivant de la dernière cellule de  $\ell_1$ . On a donc  $3(n - 1)$  affectations auxquelles on ajoute les 3 affectations initiales, soit  $3n$  en tout.  
 (version récursive) On a 4 affectations à chaque appel. Il y a  $\frac{n}{2}$  appels. Soit  $2n$  en tout.

#### **Exercice 4 : Trouver le $k$ -ième élément le plus petit**[12.5 points]

Nous nous intéressons ici à un problème classique en informatique, appelé le problème de sélection, qui consiste en la recherche du  $k$ -ième élément le plus petit parmi  $n$  éléments.

**Problème :** soit  $t$  un tableau de  $n$  entiers tous différents indicé de 0 à  $n - 1$ , trouver l'élément tel qu'il y a  $k - 1$  éléments strictement plus petits et  $n - k$  éléments strictement plus grands.

Par exemple, si le tableau contient [1; 7; 9; 2; 3; 5] alors le 4-ième plus petit élément est 5, puisque il y a 3 éléments plus petits.

Lorsque  $k = \lfloor \frac{n}{2} \rfloor$ , la recherche du  $k$ -ième élément le plus petit consiste en la recherche de la médiane.

Pour simplifier l'écriture, on pourra noter  $t(i..j)$  la tranche du tableau allant de  $i$  à  $j$  inclus.

Pour les différentes stratégies proposées vous aurez à écrire une fonction `selection` qui prend en entrée un tableau  $t$  ou une tranche de tableau et un entier  $k$ . On supposera que le  $k$  passé en paramètre est valable.

## Stratégie 1

Nous proposons une première stratégie qui consiste à trier le tableau.

**Q 4.1** [1.5 points] Proposer une fonction en pseudocode qui résoud le problème en utilisant cette stratégie.

*Corrigé*

```
fonction SELECTIONNER( $t, k$ )
    soit  $t'$  le tableau trié grâce au tri fusion
    retourner  $t'(k - 1)$ 
fin fonction
```

**Q 4.2** [1 point] Donner le comportement asymptotique en nombre de comparaisons en fonction de  $n$ . Justifier.

*Corrigé*

Le tri s'effectue en  $\Theta(n \log n)$  et l'extraction de l'élément en position  $k$  se fait en  $\Theta(1)$ .

## Stratégie 2

Une autre stratégie n'utilise pas de tri mais consiste à calculer successivement le minimum, le second minimum, etc. On se donne comme contrainte supplémentaire que l'algorithme doit s'effectuer dans un espace mémoire en  $\Theta(1)$ .

**Q 4.3** [2 points] Proposer une fonction en pseudocode qui résoud le problème en utilisant cette stratégie.

*Corrigé*

```
fonction SELECTIONNER( $t, k$ )
    pour  $i = 0$  à  $k - 1$  faire
        soit  $imin$  l'indice du minimum dans  $t(i..n - 1)$ 
        échanger  $t(imin)$  et  $t(i)$ 
    fin pour
    retourner  $t(k - 1)$ 
fin fonction
```

**Q 4.4** [1 point] Y a-t-il un pire et un meilleur des cas? Justifier.

*Corrigé*

Non, car la recherche du minimum nécessite de comparer tous les éléments.

**Q 4.5** [1 point] Donner le nombre exact de comparaisons en fonction de  $k$  et  $n$ .

*Corrigé*

Le nombre de comparaisons dans la tranche de tableau  $t(i..j)$  nécessite  $j - i$  comparaisons.  
Le nombre total de comparaisons est donc :

$$\sum_{i=0}^{k-1} (n - i) = \sum_{i=0}^{k-1} n - \sum_{i=0}^{k-1} i = nk - \frac{(k-1)k}{2}$$

### Stratégie 3

Une troisième stratégie consiste à partitionner successivement le tableau à la façon du tri rapide. Cependant, ici, nous savons d'avance dans laquelle des deux tranches va se trouver l'élément recherché par rapport à la taille de celles-ci.

On suppose disposer d'une fonction  $\text{PARTITIONNER}(t(i..j))$  qui partitionne la tranche  $t(i..j)$  avec comme pivot  $t(i)$  et retourne la position du pivot après partitionnement. Cette fonction s'exécute en  $j - i$  comparaisons.

Une trace d'exécution possible<sup>1</sup> sur l'exemple de tableau donné plus haut pour la recherche du 5-ème élément est :

```
SELECTIONNER( $t(0..5)$ ,  $k = 5$ )
   $s = \text{PARTITIONNER}(t(0..5)) = 0$ ,  $t = [1;3;5;9;7;2]$ 
  SELECTIONNER( $t(1..5)$ ,  $k = 4$ )
     $s = \text{PARTITIONNER}(t(1..5)) = 2$ ,  $t = [1;2;3;9;7;5]$ 
    SELECTIONNER( $t(3..5)$ ,  $k = 2$ ),
       $s = \text{PARTITIONNER}(t(3..5)) = 5$ ,  $t = [1;2;3;7;5;9]$ 
      SELECTIONNER( $t(3..4)$ ,  $k = 2$ ),
         $s = \text{PARTITIONNER}(t(3..4)) = 4$ ,  $t = [1;2;3;5;7;9]$ 
```

et on trouve donc que le 5-ème élément est 7 (à la fin  $k = 2$  et il reste 2 éléments dans la tranche).

**Q 4.6** [3 points] Proposer une fonction récursive en pseudocode qui résout le problème avec cette stratégie.

*Corrigé*

```
fonction SELECTIONNER( $t(\ell..r)$ ,  $k$ )
  soit  $s$  l'indice de PARTITIONNER( $t(\ell..r)$ )
  si  $s - \ell + 1 = k$  alors
    retourner  $t(s)$ 
  sinon
    si  $s - \ell + 1 > k$  alors
      retourner SELECTIONNER( $t(\ell..s - 1)$ ,  $k$ )
    sinon
      retourner SELECTIONNER( $t(s + 1..r)$ ,  $k - s$ )
  fin si
fin si
fin fonction
```

**Q 4.7** [0.5 point] Décrire un meilleur des cas.

*Corrigé*

$t(0)$  est le  $k$ -ième plus petit élément de  $t$ . Il suffit d'une partition.

**Q 4.8** [0.5 point] Donner  $c_m(n)$  qui compte le nombre de comparaisons réalisées dans ce cas.

*Corrigé*

$$c_m(n) = c_{\text{partition}}(n) = n - 1$$

**Q 4.9** [1 point] Décrire un pire des cas.

*Corrigé*

La partition découpe le tableau en une tranche de taille 0 et une autre de taille  $(n - 1)$ . C'est-à-dire lorsque  $s = l$  à chaque tour, ou bien  $s = r$ . C'est le cas d'un tableau trié.

1. cela dépend de la manière dont la partition est faite

**Q 4.10** [1 point] Donner le comportement asymptotique de  $c_p(n, k)$  qui compte le nombre de comparaisons réalisées dans ce cas. Justifier.

*Corrigé*

La partition sera réduite de 1 élément à chaque étape. Et ce jusqu'à arriver à un tableau de taille  $n - k$ .

$$\sum_{i=n-k}^n c_{\text{partition}}(i) = \sum_{i=n-k}^n i - 1 = \Theta(n^2)$$

---

On rappelle que :

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}$$