Nom: **Prénom:**

Université Lille1 Sciences et Technologies

Info 204 - ASD

2010–2011
Licence STS - Semestre 4

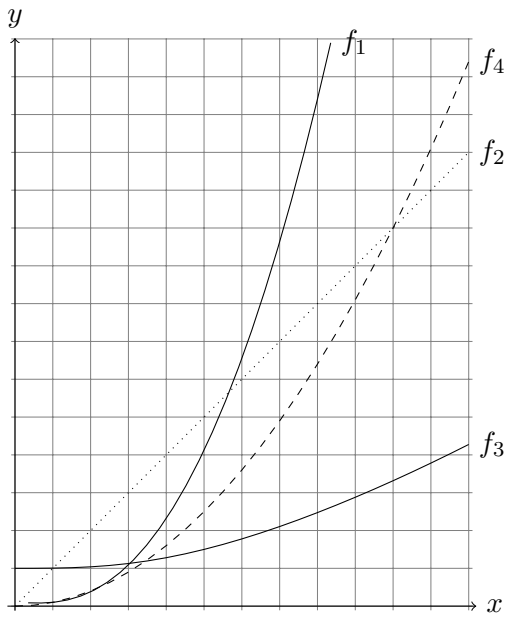
DS1 - documents de cours, TD, TP autorisés - Elements de correction

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 Θ soit une notation Ω.

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

Exercice 1 : Notations Ω, $\mathcal{O}$, Θ

On présente ci-dessous quatre fonctions dont on souhaite déterminer le comportement asymptotique des unes par rapport aux autres.



$f_1 =$	Ω	(f_2)	$f_2 =$	$\mathcal{O}$	(f_1)
$f_1 =$	Ω	(f_3)	$f_2 =$	Ω	(f_3)
$f_1 =$	Ω	(f_4)	$f_2 =$	$\mathcal{O}$	(f_4)
$f_3 =$	$\mathcal{O}$	(f_1)	$f_4 =$	$\mathcal{O}$	(f_1)
$f_3 =$	$\mathcal{O}$	(f_2)	$f_4 =$	Ω	(f_2)
$f_3 =$	$\mathcal{O}$	(f_4)	$f_4 =$	Ω	(f_3)

Q 1.1 Indiquer dans le tableau ci-dessus (directement sur la feuille), pour chaque couple de fonctions, la relation qui les lient.

Aucune des fonctions n'est comprise entre deux fonctions identiques à une constante multiplicative près donc la notation Θ ne pouvait pas être utilisée

Exercice 2 : Trouver un algorithme d'une complexité donnée

On souhaite rechercher la valeur ayant le nombre maximal d'occurrences dans un tableau quelconque. L'étudiant Dupont propose un algorithme $\mathcal{A}$ dont la complexité est en $\Theta(n^2)$. L'étudiant Durand propose un algorithme $\mathcal{B}$ dont la complexité est en $\mathcal{O}(n \log n)$. Les deux algorithmes sont corrects.

Q 2.1 Donner le code OCaml de l'algorithme $\mathcal{A}$.

Pour chaque élément du tableau, compter le nombre d'occurrences dans le tableau.

Q 2.2 Justifier que l'algorithme est de la complexité annoncée.
Soit n recherches d'occurrences, chaque recherche demandant n comparaisons.

Q 2.3 Donner le code OCaml de l'algorithme $\mathcal{B}$.
Trier le tableau, puis compter le nombre d'occurrences de chaque valeur : se fait en un seul passage sur un tableau trié, tant que la valeur est la même incrémenter un compteur, si la valeur change, conserver le max et remettre le compteur à 1.

Q 2.4 Justifier que l'algorithme est de la complexité annoncée.
Tri en $\mathcal{O}(n \log n)$, décompte en $\Theta(n)$. Soit au total $\mathcal{O}(n \log n)$.

Exercice 3 : Diviser pour régner

On souhaite procéder à la recherche du maximum dans un tableau quelconque. La consigne est de réaliser un algorithme *récursif* utilisant le principe "diviser pour régner" pour réaliser cette recherche.

On supposera traiter des tableaux d'entiers positifs ou nuls et que le tableau contient au moins un entier.

Q 3.1 Décrire en français le principe d'un algorithme permettant la recherche de la valeur maximale d'un tableau utilisant ce principe.
On divise le tableau en deux. On lance la recherche dans les deux sous-tableaux. On conserve la maximum.

Q 3.2 Ecrire en OCaml cet algorithme.

```
#let rec max_tab t =
#    let n = Array.length t
#    in
#    if n = 1 then
#        t.(0)
#    else
#        let m = n / 2 in
#        let m1 = max_tab (Array.sub t 0 m)
#        and m2 = max_tab (Array.sub t m (n-m))
#        in
#        max m1 m2
```

Q 3.3 Donner sous forme d'une équation de récurrence la complexité en nombre d'appels récursifs. Quel est le comportement asymptotique en temps ?

$$c(n) = \begin{cases} 1 & \text{si } n < 1 \\ 1 + c(n/2) + c(n - n/2) & \text{sinon} \end{cases}$$

Comportement asymptotique : on assimile l'équation à $c(n) = 1 + 2c(n/2)$ en supposant que le tableau est de longueur une puissance de 2. $a = 2$, $b = 2$, $f(n) = 1$, $n^{\log_b a} = n$ et donc $f(n) = \mathcal{O}(n^{1-\varepsilon})$ avec n'importe quel ε positif inférieur à 1. Et donc $c(n) = \Theta(n)$.

Q 3.4 Donner la complexité en espace. Quel est le comportement asymptotique en espace ?
On utilise dans la correction donnée ici deux variables à chaque appel stockant les maximums intermédiaires et Array.sub produit deux tableaux mais pas simultanément, il faut donc n'en compter qu'un.

$$e(n) = \begin{cases} 0 & \text{si } n < 1 \\ 2 + n/2 + 2e(n/2) & \text{sinon} \end{cases}$$

Cette fois, $f(n) = n/2 + 2 = \Theta(n^1)$ et donc la complexité en espace est en $\Theta(n \log n)$.

Exercice 4 : Programmation dynamique

On s'interesse ici au problème de la plus longue séquence strictement croissante dans un tableau quelconque. Il s'agit de trouver la longueur de la plus longue suite d'éléments du tableau tel que tous les éléments sont strictement croissants.

Par exemple, sur le tableau 0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15 une plus longue sous-séquence croissante est 0, 2, 6, 9, 13, 15. Cette sous-séquence est de longueur six et on voit qu'il n'existe pas de sous-séquence de longueur sept. Le plus longue sous-séquence croissante n'est pas nécessairement unique, par exemple 0, 4, 6, 9, 11, 15 en est une autre.

On propose l'algorithme suivant. Soit t le tableau contenant les nombres. Notons l un tableau initialisé à 1 dans toutes les cases tel que $l.(i)$ soit la longueur de la plus longue sous-séquence croissante qui se termine par $t.(i)$. Pour chaque position i du tableau, si il existe $j < i$ tel que $t.(j) < t.(i)$ et que $l.(i) < l.(j) + 1$ alors $l.(i)$ devient égal à $l.(j) + 1$ erreur du i en j corrigée.

Q 4.1 Dérouler l'algorithme sur le tableau t `[| 1; 0; 2; 4; 2 |]`. Vous indiquerez (directement sur le sujet) l'état du tableau l à chaque étape i, $i = 2$ correspond à l'état du tableau l après avoir traité l'élément $t.(2)$:

	0	1	2	3	4
$i = 0$	1	1	1	1	1
$i = 1$	1	1	1	1	1
$i = 2$	1	1	2	1	1
$i = 3$	1	1	2	3	1
$i = 4$	1	1	2	3	2

Q 4.2 Ecrire en OCaml un programme qui calcule le tableau l à partir d'un tabeleau d'entiers t.
Voici l'algorithme avec la recherche du maximum et l'impression de l.

```
#let lis_dyn t =
#    let n = Array.length t in
#    let l = Array.make n 1
#    and m = ref 1 in
#    for i = 0 to n-1 do
#        for j = 0 to i-1 do
#            if t.(j) < t.(i) && l.(j) + 1 > l.(i) then
#            l.(i) <- l.(j) + 1
#        done;
#        Printf.printf "i = %d l= " i; for k = 0 to n-1 do Printf.printf "%2d " l.(k) done;
#        Printf.printf "\n";
#        if !m < l.(i) then m := l.(i)
#    done;
#    !m
```

Q 4.3 Comment obtenir à partir de l la longeur de la séquence croissante la plus longue ?
C'est la valeur maximale dans l.

Q 4.4 Donner la complexité en nombre de comparaisons d'éléments de t de l'algorithme.

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

Q 4.5 **Bonus** Proposer un algorithme pour imprimer les valeurs du tableau t qui forment la plus longue sous-séquence croissante.

A partir de l'indice p correspondant à la valeur maximale de l, imprimer $t.(p)$ puis remonter les indcices inférieurs q, dès que $l.(p) = l.(q) + 1$ imprimer $t.(q)$ et faire $p := q$.