Université de Lille 1 SCIENCES ET TECHNOLOGIES

ASD

2016-2017
Licence SESI - Semestre 4

DS1 - documents de cours, TD, TP autorisés - durée 2h - Éléments 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 (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 barême est donné à titre indicatif.

Exercice 1 : Questions de cours [2 points] (réponses sans justification)

Q 1.1 Si je prouve qu'un algorithme est en $\mathcal{O}(n^2)$, peut-il être en $\mathcal{O}(n^3)$? Répondre par oui ou non.
Corrigé

Oui puisque $n^2 < n^3$.

Q 1.2 Si je prouve qu'un algorithme est en $\mathcal{O}(n \log n)$, peut-il être en $\Omega(n^2)$? Répondre par oui ou non.
Corrigé

Non. Puisque l'algorithme est en $\mathcal{O}(n \log n)$ il s'exécute dans le pire des cas en $\Theta(nlogn)$, le meilleur des cas ne pourra pas être en $\Omega(n^2)$ puisque $n \log n < n^2$.

Q 1.3 On peut résoudre l'équation de récurrence de l'algorithme suivant en utilisant le théorème général. Répondre par vrai ou faux.

```
def f(t):
  if t == []:
    return 0
  else:
    return 1 + f(t[1:])
```

Corrigé

Faux. L'algorithme n'utilise pas la stratégie diviser pour régner.

Q 1.4 N'importe quel algorithme de test de vacuité d'une liste simplement chaînée s'exécute en $\Omega(n)$. Répondre par vrai ou faux. Corrigé

Faux. Il existe au moins un algorithme permettant de tester la vacuité d'une liste en $\Theta(1)$.

Exercice 2 : Complexité [7 points]

Compétence évaluée : calculer des complexités, définir pire et meilleur des cas.

On donne l'algorithme suivant, qui prend en entrée un tableau `t` de n entiers. Le prédicat `cmp` permet de comparer deux entiers, `cmp(a,b)` retourne vrai si $a \geq b$, faux sinon.

```
n = len(t)
s = 0
i = n - 1
while i > 0:
    if i % 2 == 0:
        if cmp(t[i],i):
            for j in range(i//2, i//2 + n//2):
                s = s + t[i]
    i = i // 2
```

Q 2.1 Pourquoi le pire et le meilleur des cas sont-ils confondus pour le nombre d'appels à `cmp` ?

Corrigé

La comparaison est effectuée systématiquement, à chaque tour de boucle. C'est une boucle pour, le nombre de tous est donc constant, fixé à l'avance.

Q 2.2 Si le tableau a une longueur n tel que $n = 2^p + 1, p \geq 0$, quelle est la première valeur que prend i ?

Corrigé

$i = 2^p$

Q 2.3 Donner exactement le nombre $c(n)$ d'appels à `cmp` réalisées lorsque le tableau a une longueur n tel que $n = 2^p + 1, p \geq 0$.

Corrigé

$c(n) = c(2^p + 1) = p = \log_2(n-1)$ ou bien $c(p) = p$.

Q 2.4 Quel est le comportement asymptotique en fonction de n de l'algorithme pour le nombre d'appels à `cmp` ?

Corrigé

L'algorithme est en $\Theta(\log n)$.

Q 2.5 Décrire un pire des cas pour le nombre d'**additions** d'éléments de `t`.

Corrigé

Si t est tel que $t[i] = i \quad \forall i$: l'instruction `s = s + t[i]` est toujours exécutée.

Q 2.6 Donner exactement le nombre d'additions $a_p(n)$ d'éléments de `t` dans le pire des cas lorsque le tableau a une longueur n tel que $n = 2^p + 1, p \geq 0$ (pour exprimer $a_p(n)$, on pourra utiliser $c(n)$ si on n'a pas trouvé la réponse à la question 2.3).

Corrigé

$a_p(n) = c(n) \times \frac{n}{2} = \frac{n}{2} \log_2(n-1)$

Q 2.7 Décrire un meilleur des cas pour le nombre d'**additions** d'éléments de `t`.

Corrigé

Si t est tel que $t[i] = i - 1 \quad \forall i$: l'instruction `s = s + t[i]` n'est jamais exécutée.

Q 2.8 Donner exactement le nombre d'additions $a_m(n)$ d'éléments de `t` dans le meilleur des cas.

Corrigé

$a_m(n) = 0$

Q 2.9 En admettant que le comportement de l'agorithme est le même pour n quelconque que pour n de la forme $2^p + 1$, quel est le comportement asymptotique de l'algorithme pour le nombre d'additions $a(n)$ d'éléments de `t` ?

Corrigé

L'algorithme est en $\Omega(1)$ et en $\mathcal{O}(n \log n)$.

Exercice 3 : Récursivité [3 points]

Compétence évaluée : établir l'équation de récurrence donnant la complexité d'un algorithme récursif, appliquer le théorème général.

On considère la fonction récursive suivante :

```
# CU: n est un entier positif ou nul
def f(n):
  if n < 1:
    return n
  else:
    m = 0
    for i in range (0,n+1):
      m = max(g(i),m)
    m = max(m,f(n//2))
    return m
```

où `g` est une fonction **non récursive** définie par ailleurs.

Dans l'exercice, on s'intéresse à la complexité en nombre d'opérations de calcul du maximum entre deux entiers. On notera $c_f(n)$ la fonction de complexité de `f`, et $c_g(n)$ la fonction de complexité de `g` (qui dans son code peut effectuer des calculs de maximum entre deux entiers).

Q 3.1 Donner l'équation de récurrence de $c_f(n)$.

Corrigé

$$c_f(n) = \left(\sum_{i=0}^{n}(c_g(i)+1)\right) + 1 + c_f(\frac{n}{2})$$

Q 3.2 On suppose que $c_g(n) = k_1 \times n$, k_1 une constante, quel sera le comportement asymptotique de `f` ? Justifier.

Corrigé

$a = 1, b = 2, \log_b a = \log_2 1 = 0$

$$f(n) = \left(\sum_{i=0}^{n}(c_g(i)+1)\right) + 1 = \left(\sum_{i=0}^{n}(k_1 i + 1)\right) + 1 = \frac{k_1}{2}n(n+1) + (n+1) + 1$$

$f(n) = \Theta(n^2) = \Omega(n^{0+\varepsilon})$ avec $\varepsilon = 1$ par exemple.

$$a \times f(\frac{n}{b}) = f(\frac{n}{2}) = \frac{k_1}{2}\frac{n}{2}(\frac{n}{2}+1) + (\frac{n}{2}+1) + 1 \leq \frac{1}{2}\frac{k_1}{2}n(n+1) + (n+1) + 1 = \frac{1}{2}f(n)$$

On vérifie donc la condition du 3ème cas du théorème général.
On en conclut que $c_f(n) = \Theta(f(n)) = \Theta(n^2)$

Exercice 4 : GnomeSort [10 points]

Compétence évaluée : comprendre un algorithme, évaluer sa complexité, analyser expérimentalement sa complexité.

On présente ici un nouveau tri, assez curieux puisqu'il réalise le tri d'un tableau grâce à une unique boucle.

On peut lire sur la page de son concepteur, Dick Grune :

> *The simplest sort algorithm is not Bubble Sort..., it is not Insertion Sort..., it's Gnome Sort!*
> *Gnome Sort is based on the technique used by the standard Dutch Garden Gnome (Du. : tuinkabouter). Here is how a garden gnome sorts a line of flower pots. Basically, he looks at the flower pot next to him and the previous one; if they are in the right order he steps one pot forward, otherwise he swaps them and steps one pot backwards. Boundary conditions : if there is no previous pot, he steps forwards; if there is no pot next to him, he is done.*

```
def gnomeSort(l):
    pos = 0
    while pos < len(l):
        if pos == 0 or cmp(l[pos],l[pos-1]):
            pos = pos + 1
        else:
            l[pos], l[pos-1] = l[pos-1], l[pos]
            pos = pos - 1
```

Le prédicat `cmp` permet de comparer deux entiers, `cmp(a,b)` retourne vrai si $a \geq b$, faux sinon.

Q 4.1 Est-ce un tri sur place ? Justifier.

Corrigé

Oui parce que seuls des échanges au sein de la liste initiale sont réalisés.

Q 4.2 Quelle est le comportement asymptotique en espace ? Justifier.

Corrigé

$\Theta(1)$ puisqu'on n'utilise pas d'espace mémoire annexe autre que la variable servant à l'échange de `l[pos]` et `l[pos-1]`.

Q 4.3 Déroulez l'algorithme pour `l1 = [1,2,3,4]` (vous indiquerez la valeur de `pos` et de `l` à la fin de chaque tour de boucle).

Corrigé

```
[1, 2, 3, 4] 1
[1, 2, 3, 4] 2
[1, 2, 3, 4] 3
[1, 2, 3, 4] 4
```

Q 4.4 Quel est le nombre d'opérations de comparaisons (d'appels à `cmp`) d'éléments du tableau réalisées dans ce cas ?

Corrigé

3

Q 4.5 Déroulez l'algorithme pour `l1 = [4,3,2,1]` (vous indiquerez la valeur de `pos` et de `l` à la fin de chaque tour de boucle).

Corrigé

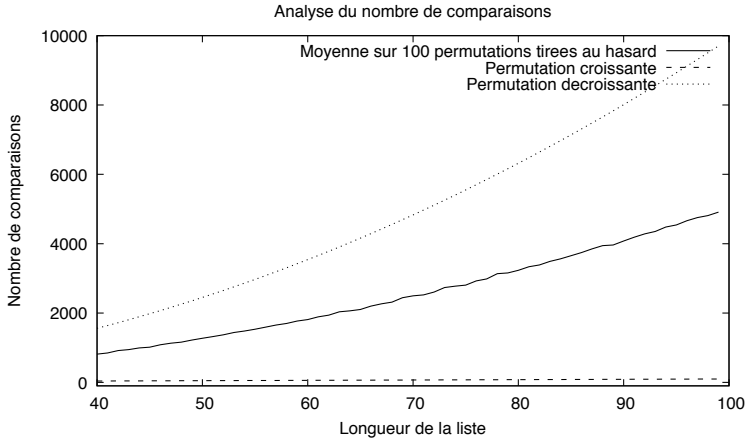
```
[4, 3, 2, 1] 1
[3, 4, 2, 1] 0
[3, 4, 2, 1] 1
[3, 4, 2, 1] 2
[3, 2, 4, 1] 1
[2, 3, 4, 1] 0
[2, 3, 4, 1] 1
[2, 3, 4, 1] 2
[2, 3, 4, 1] 3
[2, 3, 1, 4] 2
[2, 1, 3, 4] 1
[1, 2, 3, 4] 0
[1, 2, 3, 4] 1
[1, 2, 3, 4] 2
[1, 2, 3, 4] 3
[1, 2, 3, 4] 4
```

Q 4.6 Quel est le nombre d'opérations de comparaisons (d'appels à `cmp`) d'éléments réalisées dans ce cas ?

Corrigé

12

Le tracé du nombre ~~d'accès~~ de comparaisons sur différents types et tailles de permutations [1] donne ceci :



Q 4.7 Qu'en déduisez-vous ?

Corrigé

- Il existe un pire et un meilleur des cas.
- Le meilleur des cas semble être pour les permutations croissantes, en $\Theta(1)$.
- Le pire des cas semble être pour les permutations décroissantes, en $\Theta(n^2)$.
- L'algorithme semble avoir une complexité en moyenne en $\Theta(n^2)$.

Q 4.8 Décrire un meilleur des cas.

Corrigé

Un meilleur des cas est de trier une liste déjà triée.

1. Une permutation de longueur n est une séquence d'entiers composée des éléments 1 à n, différents deux à deux.

Q 4.9 Quelle est la complexité asymptotique en nombre de comparaisons (d'appels à `cmp`) dans le meilleur des cas ? Justifier.

Corrigé

Dans ce cas, c'est toujours l'instrucion `pos = pos + 1` qui est réalisée, la boucle est donc effectuée n fois, et la comparaison à chaque fois sauf à la première itération lorsque `pos` vaut 0. On est donc en $\Theta(n)$.

Q 4.10 Décrire un pire des cas.

Corrigé

Un pire des cas est de trier une liste dont les éléments sont dans l'ordre décroissant.

Q 4.11 Quelle est la complexité asymptotique en nombre de comparaisons (d'appels à `cmp`) dans le pire des cas ? Justifier clairement.

Corrigé

Ici c'est plus délicat. Chaque fois qu'on trouve une paire de positions `(pos,pos-1)` telle que les éléments sont dans l'ordre décroissant, on va faire descendre l'élément le plus petit jusqu'au début du tableau. Puis remonter les indices jusqu'à se repositionner sur une paire d'éléments décroissants.
La descente des éléments nécessite `pos` comparaisons (instructions du `else`, la remontée d'indice nécessite `pos` comparaisons (instruction du `then`), et pas `pos`+1 puisque lorsque `pos` vaut 0 il n'y a pas de comparaison.
Pour la paire en position (1,0), on va réaliser une comparaison, puis 1 échange, puis une comparaison pour se positionner sur la paire (2,1). Pour la paire en position (2,1), on va réaliser 2 comparaisons (et 2 échanges), puis deux comparaisons pour se positionner sur la paire (3,2). Et ainsi de suite.
Les paires `(pos,pos-1)` sont envisagées à partir de `pos=0` et jusqu'à `pos=n-1`.

$$c(n) = \sum_{i=1}^{n-1} i + i = 2 \times \sum_{i=1}^{n-1} i = 2 \times \frac{n(n-1)}{2} = n^2 - n = \Theta(n^2)$$