université PARIS-SACLAY

Correction interro n°1

Cours d'algorithmique

—Licence CFA / Troisième Année—

Les documents manuscrits, sujets de travaux pratiques et dirigés ainsi que les supports de cours sont autorisés. Tous les autres documents tels que livres, calculatrices, téléphones portables et ordinateurs sont interdits.

Les exercices sont indépendants. Si l'on ne sait pas justifier une question, on peut néanmoins utiliser la réponse dans la suite.

Durée : 45 min

▸ Exercice 1. (Tableaux ZigZag)

On dit qu'un tableau `Tab` de taille `n` est en ZigZag si

$$\texttt{Tab[0]} < \texttt{Tab[1]} > \texttt{Tab[2]} < \texttt{Tab[3]} > \texttt{Tab[4]} < \cdots$$

Par exemple, $[3, 6, 1, 4, 2, 5]$ est en ZigZag mais pas $[1, 6, 7, 4, 2, 5]$. Écrire un algorithme qui prend en paramètre un tableau `Tab` de taille `n` et qui retourne `true` si le tableau est en zigzag et `false` sinon. Quelle est la complexité de cet algorithme ? Justifier brièvement votre réponse.

✂ ----------

```
bool ZigZag(int n, int Tab[]) {
  for (int i=0; i < n; i++) {
    if (i % 2 == 0) {
       if (Tab[i] >= Tab[i+1]) return false
    }
    else {
       if (Tab[i] <= Tab[i+1]) return false
    }
  }
  return true
}
```

La boucle est faites au plus n fois, le code à l'intérieur de la boucle est en $O(1)$ donc la fonction est en $O(n)$.

Solution alternative

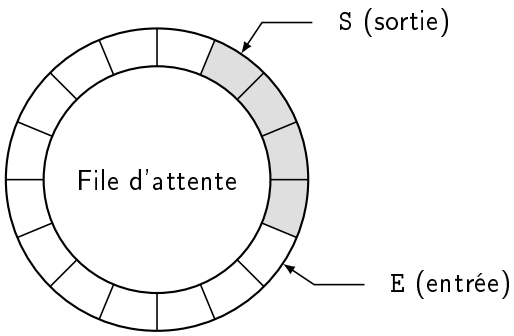
```
bool ZigZag(int n, int Tab[]) {
  for (int i=0; i+2 <= n; i=i+2) {
    if (Tab[i] >= Tab[i+1] or Tab[i+1] <= Tab[i+2]) return false
  }
  return (n % 2 == 1 or Tab[n-2] >= Tab[n-1])
}
```

La boucle est faite au plus $n/2$ fois, le code à l'intérieur de la boucle est en $O(1)$ donc la fonction est en $O(n)$.

---------- ✂

▸ **Exercice 2. (File d'attente à double entrée)**

On reprend la structure de file d'attente codée par un tableau circulaire vue en TD. On utilisera les quatre attributs `capacité`, `E`, `S`, `Tab` selon l'illustration ci-dessous, où les cases grisées sont occupées :



Dans le fonctionnement normal vu en TD, les éléments entrent à l'indice `E` et sortent à l'indice `S`. On demande d'écrire une fonction `addS` qui permet de mettre dans la structure un élément du côté de la sortie (indice `S`) sans changer les autres éléments. Ainsi le nouvel élément sera le premier à sortir. Ça permet de prendre en compte le cas d'une personne prioritaire qui passe devant tout le monde dans la file d'attente. On peut réutiliser les fonctions du TD et définir une nouvelle fonction auxiliaire si besoin.

✂ ----------

```
int precedent(int i) {
  if (i==0) return capacite-1
  else return i-1;
}

void addS(int e) {
  if (precedent(S) == E) raise ("File Pleine")
  S = precedent(S)
  Tab[S] = e
}
```

---------- ✂

▸ **Exercice 3. (Complexité)**

1. Montrer que si $f \in O(g)$, alors $g \in \Omega(f)$.

✂ -

$f \in O(g)$ signifie que l'on peut trouver un entier n_0 et un réel $\alpha > 0$ tels que pour tout $n \geq n_0$ on a $f(n) \leq \alpha g(n)$. En gardant la valeur de n_0, si $n \geq n_0$, on peut écrire en divisant les deux membres de l'inégalité par α (qui est non nul et positif), la nouvelle inégalité

$$\frac{1}{\alpha} f(n) \leq \frac{\alpha}{\alpha} g(n) = g(n) \,.$$

Si l'on pose $\beta := \frac{1}{\alpha}$ qui est bien strictement positif, on a donc bien montré que pour tout $n \geq n_0$ on a $g(n) \geq \beta f(n)$. Ce qui s'écrit $g \in \Omega(f)$.

- ✂

2. En déduire que si $f \in O(g)$ et $g \in O(f)$, alors $f \in \Theta(g)$.

✂ -

si $f \in O(g)$ et $g \in O(f)$, d'après la question précédente, on a $f \in O(g)$ et $f \in \Omega(g)$. C'est bien la définition de $f \in \Theta(g)$.

- ✂

3. On considère les fonctions suivantes :

$$n + 4n^2, \quad \log(n), \quad 3 + 2^n, \quad 4n + 1, \quad 7, \quad 5n^2 + 1, \quad n \log(n), \quad 3 - \frac{1}{n} \,.$$

Donnez l'expression la plus simple possible de la complexité de chacune de ces fonctions.

✂ -

$$n + 4n^2 \in \Theta(n^2), \quad \log(n) \in \Theta(\log(n)), \quad 3 + 2^n \in \Theta(2^n), \quad 4n + 1 \in \Theta(n),$$
$$7 \in \Theta(1), \quad 5n^2 + 1 \in \Theta(n^2), \quad n \log(n) \in \Theta(n \log(n)), \quad 3 - \frac{1}{n} \in \Theta(1), .$$

- ✂

4. Classez ces fonctions du point de vue de la complexité, de la plus faible (programme rapide) à la plus forte (programme lent), en indiquant éventuellement les ex-aequo, c'est-à-dire les fonctions de complexités équivalentes.
   - 7 et $3 - 1/n$ en $\Theta(1)$
   - $\log(n)$ en $\Theta(\log(n))$
   - $4n + 1$ en $\Theta(n)$
   - $nlog(n)$ en $\Theta(n \log(n))$
   - $n + 4n^2$ et $5n^2 + 1$ en $\Theta(n^2)$
   - $3 + 2^n$ en $\Theta(2^n)$