# Devoir Surveillé 1 Programmation et Algorithmes

L1 MPCI

23 février 2024 - Durée: 3h

**Lorsque l'on vous demande d'écrire de décrire ou de donner un algorithme cela signifiera toujours en donner un pseudo-code, justifier de son exactitude et de sa complexité**

*On rappelle qu'aucun document ni équipement électrique ou électronique n'est autorisé.*

Le but de ce devoir est de résoudre de quatre façons différentes **le problème de trouver l'élément majoritaire**.

**Définition.** Étant donné un tableau $T$, un **élément majoritaire (de $T$)** est une valeur $x$ telle que strictement plus de la moitié des éléments de $T$ sont égaux à $x$.

Par exemple :
- le tableau $[2, 4, 5, 4, 5, 4, 5, 4, 4]$ admet 4 comme élément majoritaire,
- le tableau $[2, 2, 3, 6, 4, 3, 2, 2, 3, 3, 2, 2]$ n'a pas d'élément majoritaire.

**Les exercices :**
- sont au nombre de 4 ;
- sont indépendants ;
- ont un début plus facile que la fin.

POUR PRÉSERVER LA SANTÉ MENTALE DU CORRECTEUR RENDEZ UNE COPIE PAR EXERCICE.

# Exercice 1 – Encadrement du problème

## 1.1 Préliminaire

Combien d'éléments majoritaires peut avoir un tableau ? Prouvez-le rigoureusement.

## 1.2 Borne minimum

**Question 1.2.1** Montrez que quelle que soit la taille $n > 2$ du tableau et quel que soit $0 \leq i < n$, il existe un tableau $T$ d'élément majoritaire $e$ tel que $T[i] \neq e$.

**Question 1.2.2** Montrez que quelle que soit la taille $n \geq 1$ du tableau et quel que soit $0 \leq i < n$, il existe un tableau $T$ d'élément majoritaire $e$ tel que $T[i] = e$.

**Question 1.2.3** Montrez que quels que soient $n > 2$ et $0 \leq i < n$, il existe toujours deux tableaux d'entiers $T$ et $T'$ de taille $n$ tels que :

- $T[j] = T'[j]$ pour tout $j \neq i$,
- $T$ admet un élément majoritaire,
- $T'$ n'admet pas d'élément majoritaire.

**Question 1.2.4** Déduisez-en que la complexité du problème de l'élément majoritaire est en $\Omega(n)$, avec $n$ la taille du tableau (il n'existe pas d'algorithme permettant de résoudre le problème de l'élément majoritaire en strictement moins de $n$ opérations pour tout $n > N_0$).

## 1.3 Borne maximum

**Question 1.3.1** Donnez le pseudo-code et la preuve d'une fonction COMPTE de complexité $\mathcal{O}(n)$ telle que $\text{COMPTE}(T, x)$ est le nombre d'occurrences de $x$ dans $T$ (si $x \notin T, \text{COMPTE}(T, x) = 0$).

**Question 1.3.2** Utilisez la question précédente pour créer algorithme en $\mathcal{O}(n^2)$ pour résoudre le problème. Vous en donnerez le pseudo-code et la preuve.

**Question 1.3.3** Explicitez en quelques phrases le fonctionnement de l'algorithme de la question précédente si, en entrée, on lui donne les deux exemples du début de l'énoncé.

**Question 1.3.4** En déduire que la complexité du problème de l'élément majoritaire est en $\mathcal{O}(n^2)$

## Exercice 2 – Tris

On va maintenant utiliser le tri pour résoudre le problème.

### 2.1 Tris

**Question 2.1.1** Donnez le nom d'un algorithme de tri ayant une complexité de $\mathcal{O}(n\ln(n))$ avec $n$ la taille du tableau à trier. Vous expliciterez son fonctionnement en quelques phrases.

**Question 2.1.2** Justifiez en quelques phrases que le problème du tri est en $\Omega(n\ln(n))$.

**Question 2.1.3** Déduisez-en que le problème du tri est en $\Theta(n\ln(n))$.

### 2.2 Élément majoritaire

On suppose que l'on possède un algorithme permettant de trier un tableau de $n$ entiers en $\mathcal{O}(n\log(n))$ opérations.

**Question 2.2.1** Si $T$ est trié, démontrez l'existence d'un indice $i$ (que vous expliciterez) tel que, si $T$ admet un élément majoritaire $x$, alors $x = T[i]$.

**Question 2.2.2** Déduisez-en un algorithme itératif, dont vous donnerez le pseudo-code et la preuve, plus efficace que celui de l'exercice 1.3.2 pour résoudre le problème de l'élément majoritaire.

**Question 2.2.3** Explicitez en quelques phrases le fonctionnement de l'algorithme de la question précédente si, en entrée, on lui donne les deux exemples du début de l'énoncé.

## Exercice 3 – Diviser pour régner

### 3.1 Principe

Expliciter le principe algorithmique de *diviser pour régner*

### 3.2 La taille du tableau est une puissance de 2

On suppose dans cet exercice que $n$ est une puissance de 2. On appelle $T_1$ le tableau constitué des $n/2$ premiers éléments de $T$ (en Python, $T_1 = T[0 : n/2]$), et $T_2$ le tableau constitué des $n/2$ derniers (en Python, $T_2 = T[n/2 : n]$).

**Question 3.2.1** Montrez que si $T$ admet un élément majoritaire $x$, alors $x$ est un élément majoritaire de $T_1$ ou $T_2$.

**Question 3.2.2** Donnez un algorithme qui détermine, quand $x$ est un élément majoritaire de $T_1$ (ou $T_2$), si $x$ est un élément majoritaire de $T$.

**Question 3.2.3** Déduisez-en un algorithme récursif, plus efficace que celui de l'exercice 1.3.2, pour résoudre le problème.

### 3.3 La taille du tableau est quelconque

**Question 3.3.1** Reprenez cet exercice sans supposer que $n$ est une puissance de 2. Que faut-il changer pour que l'algorithme continue de fonctionner ?

**Question 3.3.2** Explicitez en quelques phrases le fonctionnement de l'algorithme de la question précédente si, en entrée, on lui donne les deux exemples du début de l'énoncé.

## Exercice 4 – Piles

Une pile est un type de données dont on peut créer un objet en $\mathcal{O}(1)$ opérations et possédant deux méthodes :

- `empile(élément, pile)` qui ajoute un élément différent de `None` à la structure en $\mathcal{O}(1)$ opérations (on considère que tenter d'empiler `None` ne va pas provoquer d'erreurs et ne va rien ajouter à la structure).
- `dépile(pile)` qui **rend** et supprime de la structure le dernier élément à avoir été empilé en $\mathcal{O}(1)$ opérations. Si la pile est vide cette fonction va rendre `None`

L'algorithme suivant va par exemple afficher à l'écran `3`, `1`, `2`, `4` puis `None` :

```
crée une pile vide de nom P
empile 4 dans P
empile 3 dans P
empile None dans P
x = dépile(P)
affiche à l'écran la variable x
empile 2 dans P
empile 1 dans P
x = dépile(P)
affiche à l'écran la variable x
x = dépile(P)
affiche à l'écran la variable x
x = dépile(P)
affiche à l'écran la variable x
x = dépile(P)
affiche à l'écran la variable x
```

La pile est une structure permettant de garder en mémoire des choses à faire (en empilant) et de toujours effectuer la tâche la plus récente non réalisée (en dépilant).

### 4.1 Structure de pile

Proposez une implémentation avec des listes de la structure de pile en python. Vous implémenterez les fonctions :

- `crée_pile_vide() -> list`
- `empile(x: int or None, pile: list`
- `dépile(pile:list) -> int or None`

### 4.2 Pile et élément majoritaire

On considère l'algorithme Majorité ci après.

**Question 4.2.1** Que vaut $y$ à la ligne 6 par rapport à $P_1$ ?

**Question 4.2.2** Montrez qu'à chaque itération de la boucle for, $x$ est empilé soit dans $P_1$, soit dans $P_2$.

**Question 4.2.3** Donnez le contenu des deux piles $P_1$ et $P_2$ **à la fin de l'algorithme** si, en entrée, on lui donne les deux exemples du début de l'énoncé.

**Question 4.2.4** Quelle est la complexité de l'algorithme ?

**Question 4.2.5** Montrez que tous les éléments de la pile $P_2$ sont identiques à la fin de l'algorithme.

**Question 4.2.6** Montrez qu'à la fin de l'algorithme, deux éléments consécutifs de la pile $P_1$ sont toujours différents.

**Question 4.2.7** Déduisez-en qu'un élément majoritaire, s'il en existe, ne peut être que le dernier élément stocké dans $P_2$ ou le dernier élément stocké dans $P_1$.

```
   Entrées : Un tableau d'entiers T
 1 début
 2 |  Soient P_1 et P_2 deux piles vides.
 3 |  pour chaque x de T faire
 4 |  |  y = dépile(P_1)
 5 |  |  empile(y, P_1)
 6 |  |  si y == None ou y ≠ x alors
 7 |  |  |  empile(x, P_1)
 8 |  |  sinon
 9 |  |  |  z = dépile(P_2)
10 |  |  |  si z == None ou z == x alors
11 |  |  |  |  empile(z, P_2)
12 |  |  |  |  empile(x, P_2)
13 |  |  |  sinon
14 |  |  |  |  empile(z, P_1)
15 |  |  |  |  empile(x, P_1)
16 |  |  |  fin
17 |  |  fin
18 |  fin
19 fin
```

**Algorithme 1 :** algorithme MAJORITÉ

### 4.3 Conclutions

**Question 4.3.1** Déduisez de la partie précédente un algorithme linéaire en la taille du tableau en entrée pour résoudre le problème de l'élément majoritaire

**Question 4.3.2** Quelle est la complexité du problème de l'élément majoritaire ?