NOM : PRENOM : GROUPE :

IUT de Montpellier **Partiel**
M3103 Algorithmique avancée **Durée : 2h**

Consignes :

— **commencez par mettre votre nom, prénom, groupe, en haut de *toutes* les pages du sujet**

— **répondez directement sur le sujet**

— **sauf pour l'exercice de complexité, il est interdit d'utiliser des boucles**

Exercice 1. Exercice de récursivité : fusion (tri) de deux tableaux triés

On propose d'écrire une fonction renvoyant un tableau d'entiers triés par ordre croissant correspondant à la « fusion » de deux tableaux triés :

```
int [] fusionTabTries (int [] t1, int [] t2) {
// Pre-requis : t1 et t2 sont triés par ordre croissant
int [] tabRes = new int [t1.length+t2.length];
fusionTabTriesAux (t1, 0, t2, 0, tabRes, 0);
return tabRes;
}
```

Question 1.1.

Ecrire l'algorithme récursif :

```
void fusionTabTriesAux (int [] t1, int i1, int [] t2, int i2, int []
tabRes, int iRes) {
// Pre-requis : tableaux t1 et t2 triés à partir de i1 et i2
// respectivement,
// 0 ≤ i1 ≤ t1.length, 0 ≤ i2 ≤ t2.length, 0 ≤ iRes ≤ tRes.length,
// tabRes.length - iRes = (t1.length - i1) + (t2.length - i2)
//
// Action : remplit tabRes à partir de l'indice iRes, avec les
// valeurs des sous-tableaux t1[i1..(t1.length-1)] et
// t2[i2..(t2.length-1)] triées par ordre croissant
```

NOM : PRENOM : GROUPE :

Exercice 2. Exercice de récursivité : suite de Fibonacci

On appelle suite de Fibonacci la suite (u_n) définie pour tout entier $n \geq 1$ par :

- $u_1 = u_2 = 1$,
- $\forall n \geq 3\ u_n = u_{n-1} + u_{n-2}$

Question 2.1.

Ecrire récursivement l'algorithme :

```
int fibo (int n)
// pre-requis : n ≥ 1
// resultat : retourne u_n
```

Question 2.2.

Ecrire récursivement l'algorithme :

```
int nbAddFibo (int n)
// pre-requis : n ≥ 1
// resultat : retourne le nombre total d'additions effectuées lors de
l'exécution de fibo(n)
```

Question 2.3.

Ecrire l'algorithme récursif fibo2Aux suivant :

```
  int[] fibo2Aux (int n)
//pre-requis : n ≥ 1
// resultat : retourne un tableau t de longueur 2, avec t[0] = u_n et t[1]
= u_{n+1}
// stratégie : contient un unique appel récursif
```

Question 2.4.
Réécrire l'algorithme fibo (renommé en fibo2) sous la forme d'un algorithme non récursif utilisant fibo2aux.

Exercice 3. Exercice sur diviser pour régner : minimum local

Étant donné un tableau d'entiers t de longueur non nulle, on dit qu'une case $t[l]$ de t est un *minimum local* de t si sa valeur est inférieure ou égale aux valeurs de ses voisines ($t[l-1]$ si $l > 0$ et $t[l+1]$ si $l < t.length - 1$). En particulier, si t est de longueur 1 alors $t[0]$ est forcément un minimum local de t. Par exemple, dans le tableau $[2, 3, 4, 2, 2, 1, 7]$ les cases d'indices 0, 3 et 5 sont des minimums locaux.

On adapte la définition précédente à un sous tableau de la façon suivante. Etant donné un tableau t, deux indices i, j tels que $0 \leq i \leq j < t.length$ et un indice $l \in [i, j]$, on dit que la case $t[l]$ est un minimum local **de t[i..j]** si sa valeur est inférieure ou égale aux valeurs de ses voisines dans $t[i..j]$ ($t[l-1]$ si i $l > i$ et $t[l+1]$ si $l < j$). Par exemple, pour $t = [2, 3, 4, 2, 2, 1, 7]$, $i = 1$ et $j = 4$, les cases d'indices l=1, 3 et 4 sont des minimums locaux de $t[i..j]$. On remarque donc que par exemple $l = 1$ est un minimum local de $t[1..4]$, mais pas de t.

Réfléchissez à la remarque suivante : Pour tout t et tout i, j tels que $0 \leq i \leq j < t.length$, le sous tableau $t[i..j]$ a au moins un minimum local.

Question 3.1.

On considère un tableau t, et trois indices i, m et j tels que $0 \leq i < m < j < t.length$. Montrer que

— si $t[m-1] < t[m]$ alors tout minimum local de $t[i..(m-1)]$ est un minimum local de $t[i..j]$

— si $t[m+1] < t[m]$ alors tout minimum local de $t[(m+1)..j]$ est un minimum local de $t[i..j]$

.

Question 3.2.
Déduire de la propriété précédente une fonction récursive

```
int minLocalAux(int[] tab, int i, int j)
// pre-requis: tab.length > 0
// 0 <= i <= j <tab.length
// résultat: l'indice d'un minimum local de tab[i..j]
```

Cette fonction doit implémenter un algorithme de type *diviser pour régner* de telle sorte qu'au plus un appel récursif soit fait à chaque appel de la fonction et que la différence $(j - i)$ soit au moins deux fois plus petite dans l'appel récursif que dans l'appel original.

Question 3.3.
En utilisant la fonction `minLocalAux` écrite précédemment, écrire une fonction

```
int minLocal(int[] tab)
// pre-requis: tab.length > 0
// resultat: l'indice d'un minimum local de tab
```

Question 3.4 (cette question ne vaut pas de points)
Quelle est la complexité de la fonction `minLocal` en fonction de la taille n du tableau en entrée ?

Exercice 4. Exercice de complexité : tri linéaire

On considère un tableau t de n cases, que l'on souhaite trier par ordre croissant. La majorité des tris que vous connaissez sont "en n^2" (plus précisément, il existe une constante c telle que pour tout tableau t de $n \geq 1$ cases, $m \leq cn^2$ où m est le nombre d'opérations du tri). L'objectif de cet exercice est de montrer comment on peut améliorer cette complexité et trier en temps linéaire, c'est à dire "en n", lorsque l'on a l'hypothèse supplémentaire que tous les entiers du tableau sont entre 0 et 50.

Question 4.1.

Ecrire (non récursivement, avec boucles autorisées) l'algorithme suivant, et prouver sa complexité :

```
void tri (int[] t)
// pre-requis : pour tout i dans [0, t.length − 1], 0 ≤ t[i] ≤ 50
// resultat : trie t par ordre croissant
// contrainte : la complexité doit être linéaire, c'est à dire
montrer qu'il existe une constante c telle que pour tout tableau t
de n ≥ 1 cases, m ≤ cn où m est le nombre d'opérations de tri(t)).
```

Indication : imaginez d'abord que t ne contienne que des 0 et des 1, puis essayez de généraliser.