

## Solution quizz et questions ouvertes MT-EL 01/11/2019

Remarque : l'ordre des réponses était différent selon les variantes. Donc ne faites pas attention à la lettre correspondant à la réponse correcte mais seulement à **la réponse correcte elle-même qui est surlignée en jaune**. Quelques explications brèves sont ajoutées pour les ordres de complexité.

Dans cet examen, L'opérateur **modulo** calcule le reste dans la division entière de l'opérande de gauche par l'opérande de droite et le caractère **/** désigne l'opérateur de la division entière sauf indication contraire. En l'absence d'autre précision la fonction **log** désigne la fonction logarithme de base 2.

Toutes les questions portant sur l'ordre de complexité travaillent sur des nombres correspondant au cadre classique pour lequel les opérateurs arithmétiques ont un coût constant quelle que soit l'amplitude des opérandes.

**Question 1:** Quelle est la représentation binaire en complément à deux sur 8 bits de  $-42_{10}$  ?

- A 10111110
- B **11010110**
- C 11010111
- D 10101010

**Question 2:** Soit les trois fonctions suivantes exprimant le nombre d'instructions élémentaires de problèmes différents en fonction de la taille  $N$  du problème :

$$f(N) = 1000 + N^2 + 100N^3$$

$$g(N) = 25N^3 + 5000N^2$$

$$h(N) = 1000000N^{0,5} + 5000N\log(N) + N^{1,5}$$

Quelle réponse est **Fausse** ?

- A L'ordre de complexité de  $f()$  est en  $O(N^3)$  mais pas en  $O(N^2)$
- B  $f()$  et  $g()$  ont le même ordre de complexité
- C L'ordre de complexité de  $h()$  est en  $O(N^{1,5})$  mais pas en  $O(N^{0,5})$
- D **L'ordre de complexité de  $h()$  est en  $O(N\log(N))$  mais pas en  $O(N^{0,5})$**  // car  $N\log N < N^{1,5}$

**Question 3:** Quelle est la complexité de **Algo\_sum** en fonction de  $N$  ?

Algo_sum
entrée : entier $N \geq 0$ sortie : entier $i$
$i \leftarrow 0$ $sum \leftarrow 0$  <b>Tant que</b> $sum < N$ $i \leftarrow i + 1$ $sum \leftarrow sum + i$
<b>Sortir:</b> $i$

- A  $O(N^2)$  mais pas  $O(N \log(N))$
- B  $O(\log(N))$  mais pas  $O(1)$
- C  $O(N)$  mais pas  $O(\log(N))$
- D  **$O(N^{0,5})$  mais pas  $O(\log(N))$**  // car la valeur de  $sum$  augmente comme le carré de  $i$

**Question 4: Nombre positif représentés en Virgule fixe**

Quel est le nombre binaire le plus proche du nombre décimal  $24,456_{10}$  ?

- A 10111.0101
- B 10111.1111
- C 11001.0111
- D 10111.0111

**Question 5 (Algo récursif : 2 questions):** Soit **Algo\_indice** un algorithme recevant en entrée trois entiers strictement positifs  $a, b, c$  et une liste non-vide  $L$  de taille  $N$  contenant des entiers positifs distincts triés dans l'ordre croissant.

Si on appelle **Algo\_indice** avec la valeur **1** pour  $a$  et la valeur  $N$  pour  $b$ , quelles sont les expressions ① et ② permettant à **Algo\_indice** de trouver l'indice de  $c$  dans la liste  $L$  ? Si  $c$  n'est pas dans la liste, l'algorithme renvoie la valeur  $-1$ .

<b>Algo_indice</b>
Entrées : $a, b, c$ trois entiers strictement positifs, et une liste $L$ non-vide de taille $N$ contenant des entiers distincts, strictement positifs et triés dans l'ordre croissant
Sortie : l'indice de $c$ dans la liste $L$ ou $-1$ si $c$ n'est pas dans la liste
<p><b>Si</b> <math>a &gt; b</math>              <b>Sortir</b> <math>-1</math>  <math>d \leftarrow a + (b-a)/2</math>  <b>Si</b> <math>L(d) = c</math>              <b>Sortir</b> <math>d</math>  <b>Si</b> <math>L(d) &gt; c</math>              <b>Sortir</b> ① ??              <b>Sortir</b> ② ??</p>

- A ① = Algo\_indice( $d-1, b, c, L$ ) et ② = Algo\_indice( $d+1, a, c, L$ )
- B ① = Algo\_indice( $a, d, c, L$ ) et ② = Algo\_indice( $d+1, b, c, L$ )
- C ① = Algo\_indice( $a, d-1, c, L$ ) et ② = Algo\_indice( $d+1, b, c, L$ )
- D ① = Algo\_indice( $d+1, b, c, L$ ) et ② = Algo\_indice( $a, d-1, c, L$ )

**Question 6:** Quelle est la complexité de l'algorithme **Algo\_indice** :

- A  $O(N^2)$  mais pas  $O(N \log(N))$
- B  $O(N \log(N))$  mais pas  $O(N)$
- C  $O(N)$  mais pas  $O(\log(N))$
- D  $O(\log(N))$  mais pas  $O(1)$  // c'est une recherche dichotomique

**Question 7:** Quelle est la complexité de l'algorithme suivant :

<b>Algo_mystere</b>
Entrées : un entier $N$ strictement positif et une liste $L$ de taille $N$ de nombres entiers
Sortie : la liste $L$ est modifiée
<p><b>Pour</b> <math>i</math> de <math>1</math> à <math>N-1</math>              <b>Pour</b> <math>j</math> de <math>N-1</math> à <math>i</math>                  <b>Si</b> <math>L(j) &gt; L(j+1)</math>                      échange(<math>L, j, j+1</math>) // algorithme à coût constant qui échange les valeurs d'indices <math>j</math> et <math>j+1</math></p>

- A  $O(N^2)$  mais pas  $O(N \log(N))$  // algorithme de tri bulle
- B  $O(N \log(N))$  mais pas  $O(N)$
- C  $O(2^N)$  mais pas  $O(N^2)$
- D  $O(N)$  mais pas  $O(\log(N))$

**Question 8: Virgule flottante sur 8 bits inspirée par le standard IEEE 754.**

Soit le motif binaire suivant : 1 1010 110

Le bit de poids fort est le bit de signe

Les 4 bits suivants donnent un entier positif appelé **exposant** dans la formule normalisée ci-dessous.

Les 3 derniers bits de poids faible constituent la **mantisse**

Voici la forme normalisée utilisée pour cette question :  $(-1)^{\text{signe}} \cdot 2^{(\text{exposant} - 7)} \cdot 1, \text{mantisse}$

On utilise la forme dénormalisée correspondante pour la valeur minimum de l'exposant.

Quelle est la valeur décimale représentée par le motif binaire indiqué plus haut ?

- A -1792
- B -12
- C -14
- D 11

**Question 9: Algorithme récursif**

<b>Algo_exec</b>
entrée : nombre réel $x > 0.0$ et $n$ un entier relatif sortie : valeur de la variable <b>resultat</b>
<p><b>Si</b> <math>n \leq 0</math>              <b>Sortir</b> <math>1/\text{Algo\_exec}(x, -n)</math> // l'opérateur / est la division de nombres réels</p> <p>resultat <math>\leftarrow 1</math>  <b>Pour</b> <math>i</math> de 1 à <math>n</math>              resultat <math>\leftarrow</math> resultat * <math>x</math></p> <p><b>Sortir</b> resultat</p>

Que peut-on dire de cet algorithme ?

- A il ne termine pas si  $n$  est strictement négatif
- B il calcule  $x^{|n|}$
- C il ne termine pas si  $n$  est nul
- D Sa complexité est en  $O(n^2)$  et pas en  $O(n \log(n))$

**Question 10: Algorithme pour rendre la monnaie<sup>1</sup> (2 questions)**

Soit l'algorithme **Algo\_A** qui prend en entrée le **prix** d'un article et le **versement** payé par le client et qui renvoie la monnaie sous forme d'une liste de valeurs entières parmi la liste (1, 2, 5, 10, 20, 50, 100, 200).

<b>Algo_A</b>
entrée : deux entiers strictement positifs, <b>prix</b> et <b>versement</b> sortie : aucune car le résultat est affiché
<p>reste <math>\leftarrow</math> versement - prix  <b>Si</b> reste &lt; 0              <b>Afficher</b> "Le versement est insuffisant : il manque de l'argent !"              <b>Sortir</b></p> <p><b>Si</b> reste = 0              <b>Afficher</b> "le compte est bon ; merci !"              <b>Sortir</b></p> <p><b>Afficher</b> "Voici la liste des valeurs correspondant à votre monnaie"  <b>Afficher</b> <b>Algo_B</b>(reste) // L'algorithme <b>Algo_B</b> renvoie une liste de valeurs (lire page suivante)</p>

<sup>1</sup> in english « algorithm to give change »

Algo_B
entrée : un entier strictement positif <b>reste</b> sortie : liste <b>R</b> de valeurs entières
Liste $V \leftarrow (1, 2, 5, 10, 20, 50, 100, 200)$ // liste constante des 8 valeurs disponibles pour rendre // la monnaie ; ordre : $V(1)=1, V(2)=2, V(3)=5, \dots, V(8)=200$
$R \leftarrow$ Liste vide
<b>Tant que</b> $reste \neq 0$ $i \leftarrow 8$ valeur_choisie $\leftarrow$ Faux // initialisation d'une variable booléenne à Faux
<b>Tant que</b> $i > 0$ <b>et</b> valeur_choisie = Faux Si ①?? valeur_choisie $\leftarrow$ Vrai Sinon ②??
$R \leftarrow$ <b>Ajouter_au_debut_de_la_liste</b> ( $R, V(i)$ ) reste $\leftarrow$ reste - $V(i)$
<b>Sortir</b> $R$

L'algorithme **Ajouter\_au\_debut\_de\_la\_liste**( liste  $L$ , entier  $v$  ) ajoute la valeur  $v$  au début de la liste  $L$  et renvoie la nouvelle liste. On pose que cet algorithme a une *complexité constante en  $O(1)$* . Par exemple, si on a une liste  $L$  contenant les valeurs (60, 70, 80), alors les instructions suivantes ...

```
L ← Ajouter_au_debut_de_la_liste(L, 30)
Afficher(L)
```

... vont produire l'affichage des valeurs (30, 60, 70, 80).

Choisir la condition ① et l'instruction (2) qui permettent à **Algo\_B** d'être correct :

- |   |                            |                        |
|---|----------------------------|------------------------|
| A | ① $i < V(i)$               | ② $i \leftarrow i - 1$ |
| B | ① $V(i) \leq \text{reste}$ | ② $i \leftarrow i - 1$ |
| C | ① $V(i) \leq \text{reste}$ | ② $i \leftarrow i + 1$ |
| D | ① $V(i) < \text{reste}$    | ② $i \leftarrow i - 1$ |

**Question 11:** Quelle est alors la complexité de **Algo\_B** en fonction de la valeur entière **reste** ?

- A  $O(2^{\text{reste}})$  mais pas  $O(\text{reste}^2)$
- B  **$O(\text{reste})$  mais pas  $O(\log(\text{reste}))$**  // pour *reste* très grand il faut env. *reste/200* passages
- C  $O(\text{reste}^2)$  mais pas  $O(\text{reste})$
- D  $O(\log(\text{reste}))$  mais pas  $O(1)$

**Question 12 Théorie:** Soit le problème **PROB** appartenant à la classe **NP** ; quelle réponse est vraie ?

- A **PROB** n'a pas de solution
- B **PROB** n'est donc pas dans **P**
- C **PROB est décidable**
- D La vérification de toute solution de **PROB** prend un temps non-polynomial

Questions Ouvertes

**Question 1 (8 points):** recherche d'une suite d'éléments (appelée **Motif**) dans une liste.

Soit **L** une liste non-vide et non triée de taille **N** dont chaque élément est un caractère alphanumérique appartenant à l'alphabet majuscule, par exemple (S, G, U, Z, T).

**1.1) (2 pts)** Ecrire l'algorithme de *Recherche de Première Apparition* (**Algo\_RPA**) acceptant en entrée un entier **N**, une liste **L** non-vide et non-triée de taille **N**, une variable **var** de type caractère alphanumérique, et fournissant en sortie l'indice de la liste **L**, compris entre **1** et **N**, où se trouve la première apparition du caractère recherché. Si la valeur de **var** n'est pas trouvée dans la liste l'algorithme renvoie **0**.

Par exemple, pour la liste **L** de valeur (S, G, U, Z, T) où  $L(1)=S$ ,  $L(2)=G$ , etc... Si **var** a pour valeur U, alors l'appel **Algo\_RPA(5, L, var)** renvoie **3**. Si **var** a pour valeur H l'appel **Algo\_RPA(5, L, var)** renvoie **0**.

1.1.1) Donner le pseudocode ci-dessous:

<b>Algo_RPA</b>
entrée : entier <b>N</b> , liste <b>L</b> non-vide et non-triée de taille <b>N</b> , variable <b>var</b> de type caractère alphanumérique sortie : indice de première apparition de <b>var</b> dans <b>L</b> , ou <b>0</b> si la valeur de <b>var</b> n'est pas dans <b>L</b> .
<pre> <b>Pour i de 1 à N</b>   <b>Si L(i) = var</b>     <b>Sortir i</b>  <b>Sortir 0</b>           </pre>

1.1.2) Quel est son ordre de complexité en fonction de **N** : ...Linéaire en  $O(N)$  .....

**1.2) (3 pts)** On considère maintenant une autre liste **LM** (comme *Liste Motif*) non-vide et non-triée de taille **M** contenant aussi des lettres majuscules.

Ecrire l'algorithme de *Recherche de Première Apparition du Motif* (**Algo\_RPAM**) acceptant en entrée un entier **N**, une liste **L** non-vide et non-triée de taille **N**, un entier **M** tel que  $M \leq N$ , une liste **LM** non-vide et non-triée de taille **M**, et renvoie en sortie un booléen à **Vrai** dès que la liste **LM** est trouvée dans **L** et Faux sinon. Attention : la liste **LM** doit être trouvée en un seul bloc avec tous ses caractères consécutifs dans **L**.

## Solution quizz et questions ouvertes MT-EL 01/11/2019

Par exemple, pour la liste **L** de valeur (S, G, U, U, Z, T) où  $L(1)=S$ ,  $L(2)=G$ , etc...

Si **LM** = (U, Z) alors l'algorithme renvoie **Vrai** car ce motif est contenu dans **L**.

Si **LM** = (G, Z) l'algorithme renvoie **Faux** car les caractères G et Z ne sont pas consécutifs dans **L**.

Si **LM** = (Z, T, B) l'algorithme renvoie **Faux** car le dernier caractère de **LM** n'est pas dans **L**.

1.2.1) Donner le pseudocode ci-dessous.

<b>Algo_RPAM</b>
entrée : entier <b>N</b> , liste <b>L</b> non-vide et non-triée de taille <b>N</b> entier $M \leq N$ , liste <b>LM</b> non-vide et non-triée de taille <b>M</b> sortie : valeur booléenne <b>Vrai</b> dès que la liste <b>LM</b> est trouvée dans la liste <b>L</b> et <b>Faux</b> sinon.
<pre><b>Pour i de 1 à N-M+1</b>      <b>j ← 1</b>      <b>Tant que L(i + j-1) = LM(j)</b>          <b>Si j = M</b>             <b>Sortir Vrai</b>         <b>Sinon</b>             <b>j ← j + 1</b>  <b>Sortir Faux</b></pre>

1.2.2) Quel est son ordre de complexité en fonction de N dans le cas où M vaut N/2 :

Les N/2 passages dans la boucle Pour produisent des passages dans la boucle Tant-que : d'abord N/2 puis N/2 -1 puis N/2 -2 etc jusqu'à 1. Une telle somme est en  $O(N^2)$

### Solution quizz et questions ouvertes MT-EL 01/11/2019

**1.3) (3 pts)** cette question travaille avec les même entrées que la question précédente. Ecrire l'algorithme de *Décompte du Nombre d'Apparitions du Motif (Algo\_DNAM)* qui renvoie le nombre de fois que la liste **LM** apparaît en bloc et de manière distincte (sans superposition) dans la liste **L**. L'algorithme renvoie **0** si **LM** n'apparaît aucune fois dans **L**.

Par exemple, pour la liste **L** de valeur (S, G, U, Z, T, G, U, Z, T, T, T) où  $L(1)=S$ ,  $L(2)=G$ , etc...

Si **LM** = (G, U, Z) alors l'algorithme renvoie **2** car ce motif apparaît 2 fois dans **L**.

Si **LM** = (G, Z) l'algorithme renvoie **0** car **LM** n'apparaît pas en un seul bloc dans **L**.

Si **LM** = (T, T) l'algorithme renvoie **1** car ce motif n'apparaît qu'une fois sans superposition dans **L**.

Donner le pseudocode ci-dessous.

Algo_DNAM
entrée : entier <b>N</b> , liste <b>L</b> non-vide et non-triée de taille <b>N</b> entier $M \leq N$ , liste <b>LM</b> non-vide et non-triée de taille <b>M</b> sortie : valeur entière du nombre de fois que <b>LM</b> apparaît en bloc et de manière distincte dans <b>L</b> .
<pre>count ← 0  i ← 1 Tant que i ≤ N-M+1      j ← 1     check ← Vrai     Tant que check=Vrai and L(i+j-1)=LM(j)          Si j = M             count ← count + 1             i ← i + M - 1             check ← Faux          Sinon             j ← j+1      i ← i + 1  Sortir count</pre>

**Question 2: Algorithme de Luhn (5 points)**

Cette question est inspirée par l'algorithme de Luhn qui permet de vérifier la validité d'un numéro de carte de crédit. Pour cela on suppose que le numéro de carte est représenté par un nombre entier positif  $n$ .

2.1) (2 points) Dans une étape préparatoire on demande d'écrire l'algorithme utilitaire **Prendre\_chiffre( $n$ ,  $rank$ )** qui reçoit en entrée un entier positif  $n$  et un second entier positif  $rank$  compris entre 0 et la partie entière de  $\log_{10}(n)$  et qui renvoie le chiffre de  $n$  associé à la puissance  $rank$  de 10.

Exemple: pour  $n$  valant 678 les valeurs autorisées du paramètre  $rank$  sont 0, 1 et 2 comme illustré ci-contre :

- Prendre\_chiffre(678, 0)** renvoie 8
- Prendre\_chiffre(678, 1)** renvoie 7
- Prendre\_chiffre(678, 2)** renvoie 6

$10^2$	$10^1$	$10^0$
<b>6</b>	<b>7</b>	<b>8</b>

Utiliser la division entière / et l'opérateur **modulo** pour obtenir le résultat demandé.

**Prendre\_chiffre**  
 entrée : entier positif  $n$ , entier  $rank$  compris entre 0 et la partie entière de  $\log_{10}(n)$   
 sortie : la valeur entière du chiffre de  $n$  associé à la puissance  $rank$  de 10

```

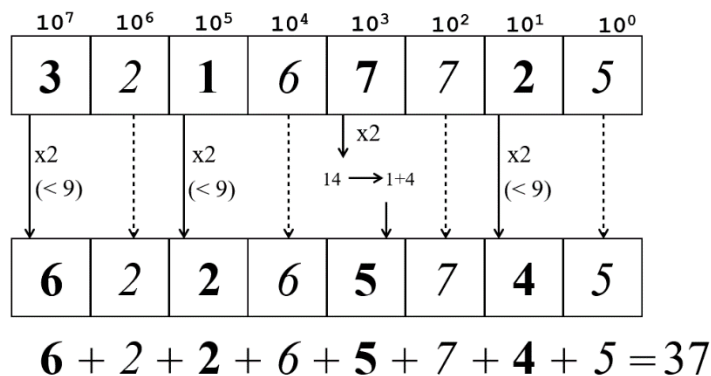
Pour i de 1 à rank par pas de 1
  n ← n/10

Sortir n modulo 10

// une solution utilisant l'opérateur d'élevation à
// la puissance (ex : 10rank) est aussi acceptée.
    
```

2.2 ) (3 points) L'algorithme de Luhn construit une somme **sum** à partir des chiffres de  $n$  de la manière suivante. Si le chiffre correspond à une puissance paire de 10, il est ajouté à **sum** sans modification. Par contre, si le chiffre correspond à une puissance impaire de 10 alors ce chiffre est multiplié par 2. Si le résultat de la multiplication ne dépasse pas 9 on l'ajoute à **sum**. Si par contre il dépasse 9, on calcule la somme des chiffres du résultat ; cette somme est ajoutée à **sum**. L'algorithme renvoie **Vrai** si **sum** est un multiple de 10 et renvoie **Faux** sinon.

La figure suivante illustre cet algorithme sur le nombre  $n$  valant 32167725.



La valeur obtenue pour **sum** n'étant pas multiple de 10 l'algorithme doit renvoyer le booléen **Faux**.



## Solution quizz et questions ouvertes MT-EL 01/11/2019

Utiliser l'algorithme **Prendre\_chiffre** pour écrire le pseudocode de l'algorithme de Luhn appliqué à un entier **n** (remarque : on ne se préoccupe pas de savoir si **n** est suffisamment grand pour être un numéro de carte de crédit ; on veut seulement vérifier si **sum** est un multiple de 10 ou pas).

Algo_Luhn
entrée : entier positif <b>n</b> sortie : valeur booléenne Vrai si <b>sum</b> est un multiple de 10 et Faux sinon
<pre>sum ← 0 max ← partie_entière(log10(n))  Pour i de 0 à max      val ← prendre_chiffre(n,i)      Si (i modulo 2) = 0         sum ← sum + val     Sinon         val ← 2*val         Si val &lt; 10             sum ← sum + val          Sinon // une seule dizaine + les unités             sum ← sum + 1 + (val modulo 10)          // Aussi ok si on utilise prendre_chiffre 0 et 1 sur val  Si (sum modulo 10) = 0     Sortir Vrai  Sortir Faux</pre>