Correction

EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communications
Cours ICC aux sections MA et PH
Chappelier J.-C.

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

INFORMATIQUE, CALCUL & COMMUNICATIONS

Sections MA & PH

Correction de l'examen intermédiaire I

26 octobre 2018

SUJET 1

Instructions :

- Vous disposez d'une heure quinze minutes pour faire cet examen (15h15 - 16h30).
- L'examen est composé de 2 parties : un questionnaire à choix multiples, à 12 points, prévu sur 45 minutes, et une partie à questions ouvertes, à 8 points, prévue sur 30 minutes.
  Mais vous êtes libres de gérer votre temps comme bon vous semble.
- **AUCUN DOCUMENT N'EST AUTORISÉ, NI AUCUN MATÉRIEL ÉLECTRONIQUE**.
- Vous devez **écrire à l'encre noire ou bleu foncée**, pas au crayon, ni en une autre couleur.
- Pour la première partie (questions à choix multiples), chaque question n'a qu'une seule réponse correcte parmi les quatre propositions. Il n'y a pas de point négatif.
  Indiquez vos réponses en bas de **cette** page en écrivant *clairement* pour chaque question une lettre majuscule parmi A, B, C et D. (Vous êtes autorisés à dégrafer cette page)
  Aucune autre réponse ne sera considérée, et en cas de rature, ou de toute ambiguïté de réponse, nous compterons la réponse comme fausse.
- Pour la seconde partie, répondez directement sur la donnée, à la place libre prévue à cet effet. Aucune feuille supplémentaire ne sera considérée.
- Toutes les questions comptent pour la note finale.

Notations : dans cet examen, le premier élément d'une liste L est noté $L[1]$.

Réponses aux quiz :

Reportez ici *en majuscule* la lettre de la réponse choisie pour chaque question, sans aucune rature.

1	2	3	4	5	6	7	8	9	10	11	12
A	B	C	A	B	C	C	D	D	B	B	A

PARTIE QUIZ

Question 1) On considère ici uniquement des schémas binaires sur 8 bits représentant des nombres entiers *positifs*. A quelle valeur décimale correspond le schéma binaire sur 8 bits de l'addition de `10010001` et `11111100` ?

✔**A]** 141 **B]** -115 **C]** 397 **D]** 198

Question 2) Si l'on interprète les schémas binaires suivants comme des nombres entiers *signés*, quel schéma correspond à la plus petite valeur ?

A] `11100000` ✔**B]** `10000011` **C]** `01110000` **D]** `00000111`

Question 3) En représentation non signée sur 8 bits, à quel nombre entier positif correspond le schéma binaire `10110101` ?

A] 75 **B]** 173 ✔**C]** 181 **D]** 74

Question 4) Toujours pour le même schéma binaire `10110101`, à quel nombre entier *signé* cela correspondrait-il ?

✔**A]** -75 **B]** 53 **C]** -181 **D]** -74

Question 5) Quelle est la complexité de l'algorithme suivant :

algo5

entrée : *Entier positif* n
sortie : *valeur*

Si $n \leq 2$
  Sortir : 3
Sinon
  Sortir : $4 \log\big(\textbf{algo5}(n-2)\big) + 1$

A] $\mathcal{O}(\log(n))$
✔**B]** $\mathcal{O}(n)$ mais pas $\mathcal{O}(\log(n))$
C] $\mathcal{O}(2^n)$ mais pas $\mathcal{O}(n^2)$
D] $\mathcal{O}(n^2)$ mais pas $\mathcal{O}(n)$

Question 6) Quelle est la complexité d'une recherche dans une liste quelconque (pas forcément ordonnée) de n éléments, si l'on utilise le meilleur algorithme de tri et la recherche par dichotomie ?

A] $\mathcal{O}(n)$ mais pas $\mathcal{O}(\log(n))$
B] $\mathcal{O}(\log(n))$
✔**C]** $\mathcal{O}(n \log(n))$ mais pas $\mathcal{O}(n)$
D] $\mathcal{O}\big(n\,(\log(n))^2\big)$ mais pas $\mathcal{O}(n \log(n))$

suite au dos ☞

Question 7) On considère la machine de Turing dont la table de transition est :

	0	1	ε
1	$(2, \varepsilon, +)$	$(3, \varepsilon, +)$	$(4, \varepsilon, -)$
2	$(2, 0, +)$	$(2, 1, +)$	$(4, 0, +)$
3	$(3, 1, +)$	$(3, 0, +)$	$(4, 1, +)$

Quel est l'état de la bande lorsque la machine s'arrête, si elle a démarré dans l'état 1 avec sa tête de lecture positionnée comme suit :

$\cdots\varepsilon$	1	1	0	1	0	1	1	0	$\varepsilon\cdots$
	↑								

A] $\cdots\varepsilon\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ \varepsilon\cdots$

B] $\cdots\varepsilon\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ \varepsilon\cdots$

✔**C]** $\cdots\varepsilon\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ \varepsilon\cdots$

D] $\cdots\varepsilon\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ \varepsilon\cdots$

Question 8) Sachant que « 3-SAT » est le nom d'un problème de décision célèbre pour les informaticiens, connu pour être dans NP, laquelle des propositions suivantes est vraie ?

A] « 3-SAT » n'a pas de solution.

B] « 3-SAT » n'est donc forcément pas dans P.

C] On connaît des algorithmes efficaces pour résoudre toute instance de « 3-SAT ».

✔**D]** Toute solution de « 3-SAT » est facilement vérifiable.

Question 9) Supposons que l'on sache qu'un problème de décision nommé « Tank » n'est pas dans NP. Laquelle des affirmations suivantes est vraie ?

A] « Tank » est dans P.

B] « Tank » est indécidable.

C] « Tank » est décidable et vérifier qu'une solution du problème « Tank » en est effectivement une prend un temps *au plus* polynomial par rapport à la taille de cette solution.

✔**D]** Soit « Tank » est indécidable, soit vérifier qu'une solution du problème « Tank » en est effectivement une prend un temps *plus que* polynomial par rapport à la taille de cette solution.

Question 10) Supposons que l'on connaisse un algorithme de complexité $4\,n\log(n) + 3\,n + 2$ permettant de résoudre un problème de décision appelé « Cypher » portant sur des données de taille n. Que peut-on en déduire ?

A] « Cypher » n'est pas dans NP.

✔**B]** « Cypher » est dans P.

C] « Cypher » est dans NP mais pas dans P.

D] rien de tout ça.

Question 11) Quelle est la sortie de l'algorithme suivant sur l'entrée $L = (-3, 5, 12, -4, 3, 8, -1, -6, 4)$:

algo11

entrée : L *liste de valeurs*
sortie : *? ? ?*

```
a ⟵ taille(L)
R ⟵ ∅        // Liste vide
Si a ≥ 3
    Pour i de 1 à a − 2
        Pour j de i + 1 à a − 1
            Pour k de j + 1 à a
                Si L[i] + L[j] + L[k] = 0
                    R ⟵ (i, j, k)
Sortir : R
```

A] $\emptyset$ (liste vide) **✔B]** $(2, 4, 7)$ **C]** $(5, -4, -1)$ **D]** $(1, 7, 9)$

Question 12) Si l'on note n la taille de la liste L, quelle est la complexité de l'algorithme de la question précédente ?

✔A] $\mathcal{O}(n^3)$ mais pas $\mathcal{O}(n^2)$
B] $\mathcal{O}(2^n)$ mais pas $\mathcal{O}(n^4)$
C] $\mathcal{O}(n^2)$
D] $\mathcal{O}(n^4)$ mais pas $\mathcal{O}(n^3)$

(Vous pouvez utiliser cette partie pour répondre aux exercices suivants, mais veuillez s.v.p. préciser le numéro de la question traitée.)

suite au dos ☞

PARTIE EXERCICES

1 – Ecriture d'algorithmes [8 points]

On s'intéresse ici à raccourcir les répétitions de trois ou plus valeurs identiques successives ; par exemple à produire la liste $(6, 6, 4, 4, 12, 4, 6)$ à partir de la liste $(6, 6, 4, 4, 4, 12, 4, 6)$ en supprimant le 4 en cinquième position car il est présent trois fois consécutives.

A noter que :

- les seules valeurs supprimées sont celles qui sont répétées successivement trois fois ou plus (l'une à la suite de l'autre) ; on ne garde alors que deux de ces valeurs (cf la valeur 4 ci-dessus) ;
- toute valeur présente une ou deux fois successivement est préservée, et l'on conserve l'ordre de la liste ;
- en sortie on ne peut donc pas avoir plus de deux valeurs identiques consécutives.

Question 13) **[2.5 points]** Ecrivez un algorithme itératif (c.-à-d. non récursif, mais avec des boucles) résolvant ce problème.

Question 14) **[1 point]** Déterminez la complexité de votre algorithme. Justifiez votre réponse.

Question 15) **[3 points]** On s'intéresse maintenant à une solution récursive séparant le premier élément et le reste de la liste. Pour cela, il serait plus pratique que l'algorithme retourne également le nombre de répétitions successives de la première valeur dans la liste résultat ; par exemple à partir de la liste $(6, 6, 6, 6, 4, 4, 4, 12, 4, 6)$, l'algorithme récursif doit produire la sortie $(6, 6, 4, 4, 12, 4, 6)$ et 2, car il y a 2 fois la valeur 6 au début de la liste résultat. A noter que cette information supplémentaire sera donc nécessairement inférieure ou égale à 2.

Ecrivez un algorithme récursif résolvant le problème de cette façon.

Question 16) **[1.5 points]** Déterminez la complexité de votre algorithme. Justifiez votre réponse.

Réponses :

En notant $L' \oplus x$ l'ajout de x à la fin de L',[1] voici quatre exemples d'algorihtmes itératifs :

simplifie1

entrée : L, *liste de nombres*
sortie : L', *version expurgée de* L

```
n ⟵ taille(L)
Si n ≤ 2
    Sortir : L
L' ⟵ (L[1], L[2])
Pour i allant de 3 à n
    Si L[i] ≠ L[i − 1] ou L[i] ≠ L[i − 2]
        L' ⟵ L' ⊕ L[i]
Sortir : L'
```

simplifie2

entrée : L *liste* ***non vide***
sortie : L', *version expurgée de* L

```
n ⟵ taille(L)
a ⟵ L[1]
L' ⟵ (a)
m ⟵ 1
Pour i allant de 2 à n
    Si m < 2
        L' ⟵ L' ⊕ L[i]
        m ⟵ m + 1
    Si a ≠ L[i]
        m ⟵ 1
    a ⟵ L[i]
Sortir : L'
```

1. Vous pouvez le noter comme vous le voulez ; p.ex. $(L'[1], ..., L'[n'], x)$ avec n' la taille de L'.

simplifie3

entrée : L
sortie : L', *version expurgée de* L

$n \longleftarrow$ **taille**(L)
$L' \longleftarrow \emptyset$
$i \longleftarrow 1$
Tant que $i \leq n$
  $L' \longleftarrow L' \oplus L[i]$
  $j \longleftarrow i+1$
  Tant que $j \leq n$ **et** $L[j] = L[i]$
    $j \longleftarrow j+1$
  Si $j > i+1$
    $L' \longleftarrow L' \oplus L[i]$
  $i \longleftarrow j$
Sortir : L'

simplifie4

entrée : L
sortie : L *expurgée (modifie* L*)*

$n \longleftarrow$ **taille**(L)
$i \longleftarrow 1$
Tant que $i \leq n-2$
  Si $L(i) = L(i+1) = L(i+2)$
    supprimer le i-ième élément de L
    $n \longleftarrow n-1$
  Sinon
    $i \longleftarrow i+1$
Sortir : L

Quelques erreurs fréquentes à ne pas commettre :

- Modifier i ou t dans une boucle « Pour i allant de a à t » : ce qui se passe n'est *pas défini* car ceci n'est plus une itération : utiliser une boucle « Tant que » dans de tels cas (voir l'algorithme **simplifie4** ci-dessus)
- Ecrire $L[i]$ pour les i non valables, p.ex. $L[1]$ pour une liste vide ou $L[2]$ pour une liste à 1 seul élément ;
- Ecrire « Supprimer $L[i]$ » tout seul, sans autre ; cela pose plusieurs problèmes :
  - Tout d'abord cela ne veut rien dire : $L[i]$ est une *valeur* : que signifie « Supprimer 36 » ? Indiquer clairement que l'on supprime dans la liste L.
  - Ensuite que veut-on dire : « Supprimer $L[i]$ de L » ? (qui enleverait cette **valeur** *partout* dans L). Indiquer clairement que l'on veut parler de *l'élément* et non pas de la valeur (voir l'algorithme **simplifie4** ci-dessus).
  - De plus, une telle suppression *modifie* L : il faut donc faire très attention aux variables que l'on manipule pour accéder à cette (nouvelle version de la) liste : mettre à jour la taille, ne pas avancer « trop vite » les index (en clair : « rester sur place » s'il y a eu une suppression). Voir encore une fois à ce sujet l'algorithme **simplifie4** ci-dessus.
  - Enfin « Supprimer le i-ème élément de L » n'est pas nécessairement une *opération élémentaire* et, en toute rigueur, ne fait pas partie des « algorithmes/opérations » vus en cours. Mais nous avons été tolérants sur ce dernier point.

  En fait, il est beaucoup plus simple de *construire* une autre liste plutôt que de supprimer des éléments d'une liste existante.

La complexité des algorithmes précédents est en $\mathcal{O}(n)$, où n est la taille de la liste (précisez vos notations !). On ne parcourt en effet qu'une seule fois la liste. Le fait que dans l'algorithme **simflifie3**, il y ait une boucle dans une boucle ne le rend pas pour autant $\mathcal{O}(n^2)$ puisque cette seconde boucle aura pour effet au final de faire avancer i d'autant. Chaque élément de la liste n'aura bien été vu qu'une seule fois.

On a toléré le fait de considérer (implicitement) que « Supprimer le i-ème élément de L » est une opération élémentaire.
Et il est plus que raisonnable de supposer que la complexité de **taille()** est dans $\mathcal{O}(n)$ (ce qui suffit ici), voire dans $\mathcal{O}(1)$.

Voici une version possible pour l'algorithme récursif qui n'utilise aucune boucle [2] :

simplifie

entrée : L
sortie : L', *version expurgée de* L, *et* l, *longueur du préfixe*

```
n ⟵ taille(L)
Si n ≤ 1
    Sortir : (L, n)
(L', m) ⟵ simplifie(L[2], ..., L[n])
Si L[1] = L'[1]
    Si m = 2
        Sortir : (L', m)
    Sinon
        Sortir : (L[1] ⊕ L', m + 1)
Sortir : (L[1] ⊕ L', 1)
```

en notant $x \oplus L'$ l'ajout de x au début de L' (vous pouvez le noter comme vous le voulez ; p.ex. $(x, L'[1], ..., L'[n'])$ avec n' la taille de L').

Une erreur fréquente sur cette question a été de ne pas ou de mal utiliser la valeur de retour de l'appel récursif : ne pas différencier (dans son utilisation) les deux parties de la sortie ou manipuler cette sortie comme si ce n'était qu'une liste (seulement).

La complexité de l'algorithme ci-dessus est en $\mathcal{O}(n)$, où n est la taille de la liste (précisez vos notations !). En effet, l'algorithme s'appelle lui-même qu'une seule fois, en décroissant à chaque fois la liste de (au moins) un élément. Il lui faut donc (au pire) ce nombre d'étapes pour arriver à la condition d'arrêt ; et comme tout le reste dans l'algorithme est en $\mathcal{O}(1)$ (il faut ici supposer la complexité de **taille()** dans $\mathcal{O}(1)$), on aboutit au total dans $\mathcal{O}(n)$.

2. C'était l'idée de départ mais j'aurais du le préciser dans la donnée...