Solution QUIZZ et questions ouvertes MT-EL 28/10/2016

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.

Notation: on utilise dans ce quizz la virgule pour séparer les puissances positives des puissances négatives de la base dans la notation positionnelle des nombres.

Nombres entiers sur 32 bits.

Question 1 : cette question travaille avec des nombres non-signés. Quel est le résultat en hexadécimal de l'addition des deux nombres hexadécimaux: $1234_{16} + 9986_{16}$

A BABA
B B0B0
C DADA
D ABBA

Question 2 : soit le nombre hexadécimal $FFF4_{16}$; on suppose que son motif binaire est la représentation d'un nombre décimal dans la représentation du *complément à deux* sur 16 bits. Ce nombre décimal est :

A -4
B -12
C -10
D 12

Nombres à virgule sur 8 bits.

Question 3 : quelle est le nombre décimal représentée par le motif binaire 10101001 dans la représentation *non-signée* à virgule fixe avec 4 bits de partie entière et 4 bits de partie fractionnaire :

A $21,125_{10}$
B $10,125_{10}$
C $9,5675_{10}$
D $10,5625_{10}$

Question 4 : On s'intéresse à la représentation du nombre décimal $17,5_{10}$ avec la représentation à virgule flottante avec 4 bits pour l'exposant de la base et 4 bits pour la mantisse.

A Ce nombre est exactement représenté par le motif binaire 01000001
B Ce nombre est exactement représenté par le motif binaire 01000010
C Ce nombre est approché par troncation avec le motif binaire 01000001 en faisant une erreur absolue de $0,5_{10}$
D Ce nombre est approché par troncation avec le motif binaire 01000001 en faisant une erreur absolue de $0,25_{10}$

Etant donnée une liste *non-triée* d'entiers relatifs, on s'intéresse au problème de trouver la plus grande distance entre deux valeurs de cette liste (cette distance est définie par la valeur absolue de la différence de 2 valeurs). Par exemple, pour la liste {16, -1, -7, 21}, la réponse est 28.

Question 5 : Le meilleur algorithme capable de résoudre ce problème pour une liste *non-triée* (de taille connue N) a pour ordre de complexité :

A O(log(N)) mais pas O(1)
B $O(N^2)$ mais pas O(N)
C O(N) mais pas O(log(N)) // car la plus grande distance = Max(liste) – Min(liste)
D O(1)

Etant donnée une liste *triée* d'entier relatifs, on veut résoudre le même problème que pour la question précédente.

Question 6 : Le meilleur algorithme capable de résoudre ce problème pour une liste *triée* (de taille connue N) a pour ordre de complexité :

A O(1) // car la plus grande distance = L(N) – L(1)
B O(log(N)) mais pas O(1)
C $O(\log(N)^2)$ mais pas O(log(N))
D O(N) mais pas O(log(N))

Soit l'algorithme **Algo_X** qui reçoit une liste **L** en entrée. Un élément d'une liste **L** peut être accédé avec la notation **L(i)**, avec l'indice **i** compris entre **1** et **Taille(L)**. L'ordre de complexité de **Taille(L)** est ***O(1)***.
Remarque : l'opérateur de division produit le quotient de la division entière

Algo_X
entrée : liste L avec au moins 2 éléments sortie : liste S de même taille que la liste L
t ← Taille(L) x ← t/2 **Pour** i allant de x à t S(i – x +1) ← L(i) **Pour** i allant de 1 à x-1 S(t – i +1) ← L(i) **Sortir** : S

Question 7 : que fait cet algorithme ?

A Les éléments de la liste S sont dans l'ordre inverse de ceux de la liste L

B La liste S commence par les éléments de L compris entre x et t, puis ceux compris entre 1 et x-1

C La liste S commence par les éléments de L compris entre x et t, puis ceux compris entre 1 et x-1 mais dans l'ordre inverse (d'abord x-1 ensuite x-2 etc…)

D La liste S commence par les éléments de L compris entre x et t mais dans l'ordre inverse (d'abord t, puis t-1, etc..), puis ceux compris entre 1 et x-1 mais dans l'ordre inverse (d'abord x-1, puis x-2 etc…)

Question 8 : Quelle est son ordre de complexité ?

A O(N) mais pas O(log(N)) // car on traite une seule fois les N éléments
B O(log(N)) mais pas O(1)
C $O(N^2)$ mais pas O(N)
D O(1)

Soit l'algorithme récursif **Algo_R(L,min,max)** qui reçoit une liste **L** en entrée ainsi que deux valeurs d'indices **min** et **max**. Un élément d'une liste **L** peut être accédé avec la notation **L(i)**, avec l'indice **i** compris entre **1** et **Taille(L)**. L'ordre de complexité de **Taille(L)** est ***O(1)***.
Remarque : l'opérateur de division produit le quotient de la division entière

```
Algo_R
entrée : liste non-vide L d'entiers positifs, min, max
sortie : un entier x
Si min = max
    Si (L(min) est impair)
        Sortir : 1
    Sinon
        Sortir : 0
Sinon
    Sortir : Algo_R(L, min, min +(max-min)/2) + Algo_R(L, ??? , max)
```

Question 9 : cet algorithme compte le nombre d'entiers impairs dans la liste L lorsqu'il est appelé avec : **Algo_R(L , 1 , Taille(L))** . Indiquer quelle expression doit remplacer « **???** » dans le pseudocode pour que cette tâche soit correctement effectuée.

A (max – min)/2 + 1
B min + (max - min)/2 + 1
C min + (max - min)/2
D max - (max –min)/2 + 1

Question 10 : Quelle est son ordre de complexité ?

A O(log(N)) mais pas O(1)
B $O(N^2)$ mais pas O(N)
C O(N) mais pas O(log(N)) // = 2 ^(profondeur des appels = $Log_2(N)$) => O(N)
D $O(2^N)$ mais pas $O(N^3)$

Question 11 : Indiquer la proposition correcte

A On sait avec certitude à l'heure actuelle que si un problème est dans la classe NP, alors il est aussi dans la classe P.
B Si un problème est dans la classe P, alors il n'est pas dans la classe NP.
C On sait avec certitude à l'heure actuelle qu'il existe un problème dans la classe NP qui n'est pas dans la classe P.
D Si un problème est dans la classe P, alors il est aussi dans la classe NP.

Question 12 : Appelons PNP un problème connu pour être dans NP. Par rapport au problème de « l'arrêt de programmes » (*halting problem*) vu en cours, que peut-on dire ?

A PNP est plus difficile que « l'arrêt de programme »
B Que toute solution de « l'arrêt de programme » est facilement vérifiable
C Qu'on peut calculer une solution de PNP et de «l'arrêt de programmes » en temps non-polynomial
D Que PNP est décidable tandis que « l'arrêt de programme » est indécidable

Questions Ouvertes

Question 1 : algorithme de conversion.

1) Soit l'algorithme **Algo_1** qui reçoit une liste **L** en entrée et produit un entier positif **x** en sortie. Un élément d'une liste **L** peut être accédé avec la notation **L(i)**, avec l'indice **i** compris entre **1** et **Taille(L)**. L'ordre de complexité de **Taille(L)** est ***O(1)***

La liste **L** contient seulement des 0 et des 1 et représente un nombre binaire non-signé avec le poids fort de la base2 au début de la liste (**L(1)**) et ainsi de suite jusqu'au poids faible en fin de liste **L(Taille(L))**.

Ecrire **Algo_1** pour qu'il calcule la valeur entière **x** en base dix correspondant au nombre binaire représenté par la liste **L**

Cette solution commence par les poids faibles et va vers les poids fort. Pour cette raison on itère avec un indice i variant de Taille(L) vers 1

Algo_1 : conversion du binaire vers un nombre en base dix
Entrée : une liste non-vide **L** contenant des 0 et des 1 Sortie : un entier **x**
x ← 0 b ← 1 Pour i de Taille(L) à 1 par pas de -1 x ← x + b*L(i) b ← b*2 Sortir : x

2.1) On veut écrire l'**Algo_2** pour qu'il calcule la liste **O** des chiffres en base 8 correspondant au nombre binaire représenté par une liste **L** du même type que pour la question 1)**.** Par construction, chaque élément de la liste **O** contient un chiffre compris entre 0 et 7 : **O(1)** contient le chiffre associé au poids fort de la base 8, etc…

Tout d'abord justifier quelle est la taille minimum de la liste **O,** notée **Taille_O**, à partir de **Taille(L)** (on pose que l'opérateur de la division produit le quotient de la division entière, et on appelle ***modulo*** l'opérateur qui donne le reste dans la division entière):

L'octal s'exprime en regroupant les bits par groupes de 3 car $8 = 2^3$ en commençant du coté des poids faibles. Si taille(L) est un multiple de 3 alors Taille_O est donné par le quotient de la division entière. Sinon on rajoute un chiffre octal à Taille_O. Sous forme de pseudocode, cela donne :

Taille_O ← Taille(L)/3
Si Taille(L) modulo 3 est différent de 0
Taille_O ← Taille_O +1

……

2.2) utiliser l'information **Taille_O** pour écrire **Algo_2** ; on peut ré-utiliser **Algo_1** si on le désire mais ça n'est pas obligatoire.

Algo_2 : conversion du binaire vers une liste de chiffres en octal
Entrée : une liste non-vide **L** contenant des 0 et des 1, l'entier **Taille_O** Sortie : une liste **O** contenant des chiffres compris entre 0 et 7

```
k ← Taille_O
x ← 0
b ← 1

Pour i de Taille(L) à 1 par pas de -1
    x ← x + b*L(t)
    b ← b*2
    Si b = 8
        O(k) ← x
        k ← k-1
        x ← 0
        b ← 1

Si k=1
    O(k) ← x

Sortir : liste O
```

Question 2 : Représentation des nombres et calcul.

Dans cet exercice on ne demande pas de pseudocode, ni d'ordre de complexité. Par contre on s'intéresse à la manière d'effectuer des calculs pour obtenir un résultat correct dans un contexte particulier. On utilise une représentation en virgule fixe *non-signée* avec **E** (E>0) bits pour la partie entière et **F** (F>0) bits pour la partie fractionnaire.

1) Quel est le **domaine couvert [min, max]** de la représentation en virgule fixe ? (commencez par supposer que F vaut 0, puis F vaut 1, etc… puis généralisez)

Le minimum est le motif binaire avec des 0 partout : ..000,0000.. c'est-à-dire zéro
Le maximum est le motif binaire avec des 1 partout : ..111,1111..

Si F = 0 (pas de partie fractionnaire), dans ce cas le maximum pour E bits de partie entière est : ..111, = 2^E -1 car il suffit d'ajouter une unité à ce motif binaire pour obtenir 2^E . Remarquons que ce nombre s'écrit aussi = 2^E -2^0

Si F = 1 (un seul bit de partie fractionnaire), dans ce cas le maximum pour E bits de partie entière est : ..111,1 = 2^E -2^{-1} car il suffit d'ajouter une unité au poids 2^{-1} pour obtenir 2^E

Pour une autre valeur de F le maximum est donc : 2^E -2^{-F}

……

2) Quelle est l'**erreur absolue maximum** qu'il est possible de faire sur un nombre réel **x** appartenant au domaine couvert mais n'étant pas exactement représenté par cette représentation en virgule fixe.
L'erreur absolue maximum est de une unité sur le poids faible = 2^{-F}

3) On se propose de calculer la moyenne géométrique **M** d'une liste de **n** nombres réels $\{x_1, x_2, \ldots, x_n\}$ exactement représentés avec cette représentation en virgule fixe.

Par définition, **M** = $\sqrt[n]{x_1 x_2 \ldots x_n}$

Pour effectuer ce calcul on suppose qu'on dispose des opérateurs arithmétique habituels et des algorithmes suivants (optionnels, selon votre choix):

pow(x,n) : calcule la puissance n^{ième} de x
root(x,n) : calcule la racine n^{ième} de x
log(x) : calcule le logarithme de x,
exp(x) : calcule l'exponentielle de x (la fonction inverse de log(x))

En supposant que les nombres $\{x_1, x_2, \ldots, x_n\}$ sont tous supérieurs à 1, quel **risque** prend-on en appliquant directement la formule fournie pour M ?

Selon l'hypothèse de cette question, tous les nombre xi sont supérieurs à 1 et donc leur produit sera aussi supérieur à 1 et à chacun des nombres x_i. Même si chacun des nombres est représenté par ce format et que leur moyenne géométrique **M** appartient aussi au domaine couvert, le calcul intermédiaire de leur produit risque de sortir du domaine couvert. Le résultat risque d'être complètement incorrect en calculant le produit avant de prendre la racine n du produit.

4) Transformer la formule pour réduire le risque identifié dans le contexte de la question 3)

Deux pistes sont possibles :
a) décomposer M en un produit de la racine n-ième de chacun des termes. Chaque racine est comprise entre 1 et x_i et est donc représentable avec ce format si x_i est représentable ; si **M** appartient au domaine couvert on a la garantie de pouvoir calculer cette valeur. Cependant il y aura en général une perte de précision au niveau de la partie fractionnaire pour chaque racine.

b) prendre le logarithme de l'expression de **M** : log(M) = 1/n*(log (x_1) + log(x_2) + …) , ensuite on fait la somme des logarithmes des x_i, on multiple par 1/n et on prend l'exponentielle du résultat pour obtenir **M**. On a la garantie que chacun des logarithmes appartient au domaine couvert. si **M** appartient au domaine couvert on a la garantie de pouvoir calculer cette valeur. Cependant il y aura en général une perte de précision au niveau de la partie fractionnaire pour chaque logarithme.