Questionnaire informatique

A. Représentation des nombres

A.1) Quels entiers peut-on coder sur 1 octet (= 8 bits) ?

A.2) Quel est l'*entier signé* sur 16 bits représenté par 0 ?

Rappel : Un *entier signé* r entre -2^{n-1} et $2^{n-1}-1$ est représenté par l'entier (décalé) $r+2^{n-1} \in [\![0, 2^n-1]\!]$.

A.3) Un flottant simple précision est codé sur 32 bits : 1 bit pour le signe, 8 bits pour l'exposant compris entre -126 et 127 (décalé de 127 de 1 à 254, l'exposant 0 et 255 codant $-\infty$ et $+\infty$) et 23 bits pour la mantisse (ne sont codés dans la mantisse `1,...` que les bits après la virgule).
Donner le flottant (en base 10) défini par 11000000110100000000000000000000

A.4) Indiquer ce que renvoie : `1.0 + 2`

B. Représentation des images

B.1) Un pixel est codé sur trois niveaux de couleur (RGB), chacun représenté par un entier compris entre 0 et 255. Quelle est la mémoire nécessaire pour stocker une image de 1024×768 pixels ?

B.2) On représente ici le segment entre deux points d'une représentation vectorielle par des pixels encrés dans une image bitmap.
Le module PIL (Python Image Library) fournit le sous module Image :
- `im = Image.new(mode,size,color=0)` alloue une nouvelle image matricielle, de type `bitmap` si mode vaut `"1"` ; le tuple `size` donne la largeur et la hauteur de l'image en pixels ; le paramètre facultatif `color` précise la couleur par défaut des pixels, en bitmap 1 pour blanc et 0 pour noir
- `im.putpixel((x,y),1)` attribue la valeur 1 au pixel de coordonnées `(x,y)` de l'image `im`
- `im.save(nom_fichier)` sauvegarde l'image dans un fichier dont on donne le nom;
- `im.show()` affiche l'image dans une fenêtre graphique.

```
01.  from PIL import Image
02.  from math import floor # renvoie l'entier immédiatement inférieur
03.  def dessin(im:img,p0:(int),p1:(int)) :
04.        x0,y0 = p0  ;  x1,y1 = p1
05.        n = 1 + max(abs(x1-x0),abs(y1-y0))
06.        for k in range(n+1):
07.             p = floor(x0+(x1-x0)*k/n),floor(y0+(y1-y0)*k/n)
08.             im.putpixel(p,0)
09.  im = Image.new("1", (10, 10), color=1)
10.  dessin(im,((2,4),(8,6)) :
11.  im.show()
```

Expliquer ce que retourne l'exécution du code des lignes 01 à 11.

C. Algorithmes de tris

C.1) On considère le code suivant, où L est une liste d'entiers :

```
01.  n = len(L)
02.  for i in range(n) :
03.       for j in range(i+1,n) :
04.            if a[j-1]<a[j] :  a[j-1],a[j] = a[j],a[j-1]
```

L'exécution du code effectue-t-elle un tri (par ordre croissant et pour toute liste L) ?

Pour obtenir le *tri à bulles*, que doit-on modifier dans le code ?

C.2) Pour fusionner deux listes triées a et b, on considère le code :

```
01.  n,m,c,i,j = len(a),len(b),[],0,0
02.  for k in range(...)  :
03.        if i<n and j<m :
04.             if a[i]<b[j] :
05.                  c.append(a[i]); i = i + 1
06.             else :
07.                  c.append(b[j]); j = j + 1
08.        if i<n :  ...
09.        else :  ...
```

Compléter les lignes 02, 08 et 09.

D. Programmation

D.1) On considère

```
01.  def fibo(n) :
02.       if n = 0 or n = 1 :  return 1
03.       return fibo(n-1)+fibo(n-2)
```

Indiquer la complexité $c(n)$ de `fibo(n)`.

Proposer un algorithme de complexité linéaire $O(n)$ en itératif et en récursif.

D.2) On considère l'algorithme glouton dans le rendu de monnaie. Le jeu de pièces disponibles est codé par la liste P strictement décroissante d'entiers naturels non nuls.

La fonction `rendu(n,P)` renvoie une liste de pièces dont la somme vaut n.

Par exemple, `rendu(9,[5,2,1])` renvoie `[5,2,2]`.

```
01.  def rendu(n,P) :
02.       L = [] ; k = 0
03.       while n>0 :
04.            if ###  à compléter
```

```
05.             else :  L.append(P[k]) ; n = n - P[k]
06.       return L
```

a) Compléter la ligne 04.

b) Donner une CNS sur P pour que l'algorithme termine dans tous les cas.

D.3) On suppose données deux listes triées a et b, et on considère le code :

```
01.  n,m,c,i,j = len(a),len(b),[]
02.  while i<n and j<m :
03.       if a[i] < b[j] :  i = i + 1
05.       elif a[i] > b[j] :  j = j + 1
06.       else :  c.append(a[i]) ; i = i + 1
```

Préciser la valeur de c à la sortie, et préciser la complexité de l'algorithme.

Questionnaire informatique. Corrigé :

A.1) Un octet est un tuple de 8 bits, donc on peut stocker les entiers entre 0 et $2^8 - 1 = 255$.
C'est pourquoi chacun des 256 caractères usuels est stocké par 1 octet.

A.2) Les entiers sont codés entre 0 et $2^{16} - 1$ modulo 2^{16}, donc 0 correspond à $-2^{15} = -32768$.

A.3) Le premier bit est 1 codant le signe -1.
L'exposant est défini par l'entier signé 10000001 qui représente l'exposant $p = 2$.
La mantisse est 10100100000000000000, représentant $x = \frac{1}{2} + \frac{1}{2^3} = 0.5 + 0.125 = 0.625$
Doc le flottant est donc $y = -\left(\frac{1}{2} + \frac{1}{2^3}\right) \times 2^2 = -2 - \frac{1}{2} = -2.5$

A.4) Un entier est converti si nécessaire en flottant, donc on renvoie 3.0

B.1) Il faut $1024 \times 768 \times 3 \times 2^8 = 9 \times 2^{10+8+8} = 9 \times 2^{28}$ bits, c'est-à-dire 9×2^{20} octets, soit 9 Mégaoctets (ou plus précisement 9 Mébioctets).

B.2) On représente le segment reliant les points $A = (2, 4)$ et $B = (8, 6)$, en utilisant une subdivision de n points, où ici $n = 1 + \max(6, 2) = 7$.

C.1) Non, par exemple, si $L = [3, 2, 1]$, on obtient $[2, 1, 3]$.
En effet, la première boucle pour $i = 0$ ne met pas nécessairement le minimum en bonne position.
Pour obtenir un algorithme correct, il faut prendre `for j in range(n-1,0,-1)`.
Il s'agit alors de l'algorithme du "tri à bulles".

C.2)

```
02.  for k in range(n+m) :     # à chaque étape, on traite un et un seul élément
08.       if i<n :  c.append(a[i]):  i = i + 1
```

09. `else : c.append(b[j]): j = j + 1`

D.1) On a $c(0) = c(1) = K$ et pour $n \geq 2$, $c(n) = K + c(n-1) + c(n-2)$.

En posant $d(n) = c(n) + K$, on a $d(n) = d(n-1) + d(n-2)$.

On reconnaît une suite de Fibonacci, donc $d(n) \sim \lambda\ \varphi^n$, où $\varphi = \frac{1}{2}(1+\sqrt{5})$.

Algorithme plus efficace :

```
def fibo(n) :
    x = 1 ; y = 1
    for k in range(n) : x,y = y,x+y
    return x
```

Variante récursive :

```
def aux(n) :
    if n == 0 : return (1,1)
    x,y = aux(n-1)      # surtout ne pas appeler deux fois aux
    return y,x+y

def fibo(n) :
    return aux(n)[0]
```

D.2) a)

04. `if k<len(P) and P[k]>n : k = k+1`

b) Une CNS est que le dernier élément de P soit égal à 1.

D.3) On calcule l'intersection c des listes a et c, avec l'ordre de multiplicité des éléments de a.

La complexité est linéaire $O(n+m)$, car à chaque étape $i+j$ augmente de 1, et on a toujours $i+j \leq n+m$.