

IPT Devoir 4- Proposition de solutions

Solution 1

On rappelle que le 0 des nombres réels en PYTHON est de l'ordre 10^{-15} . Or la condition d'arrêt nécessite le calcul de x^2 , donc si $|x| \leq 10^{-8}$ alors $x^2 \leq 10^{-15}$ et la valeur retournée est 1 qui est un point fixe de l'autre relation

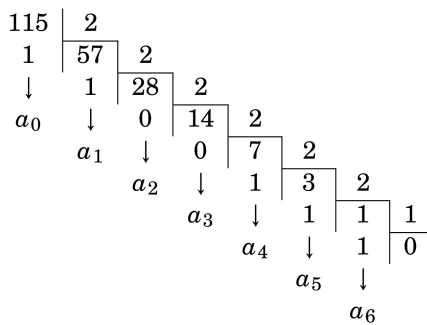
$$1 = 2 \times 1^2 - 1$$

donc la valeur finale retournée serait toujours 1.

Attention ! La condition d'arrêt doit être moins précise que $\text{abs}(x) < 1e-8$.

```
def c(x):
    if abs(x) < 1e-4:
        return 1-x**2/2
    else:
        return 2*c(x/2)**2-1
```

Solution 2 1. La représentation d'un nombre en base b consiste à effectuer des divisions successives d'un nombre par b jusqu'à obtenir un quotient nul. La représentation est donnée la collection des restes dans l'ordre inverse de l'obtention. Écriture de 115 en base 2 :



Alors $115 = \overline{1110011}^2$.
En effet, $2^6 + 2^5 + 2^4 + 2^1 + 2^0 = 64 + 32 + 16 + 2 + 1 = 115$.

2. Considérons la représentation binaire des nombres entiers relatifs sur 5 bits par la **méthode du complément à 2**.

a) L'intervalle des nombres représentés et $\llbracket -2^4, 2^4 - 1 \rrbracket = \llbracket -16, 15 \rrbracket$.

b) $13 = 8 + 4 + 1 = \overline{01101}^2$ et -9 est représenté par $-9 + 2^5 = 23 = 16 + 4 + 2 + 1 = \overline{10111}^2$.

Solution 3 1. Le tri par insertion consiste à : Méthode

- parcourir la liste de gauche à droite ;
 - à chaque étape, l'élément considéré, est classé parmi les éléments qui le précèdent (et donc qui sont déjà ordonnés).
2. Tri appliqué à L :

i	L
X	[2, 6, 1, 5, 4, 2, 4]
1	[2, 6, 1, 5, 4, 2, 4]
2	[1, 2, 6, 5, 4, 2, 4]
3	[1, 2, 5, 6, 4, 2, 4]
4	[1, 2, 4, 5, 6, 2, 4]
5	[1, 2, 2, 4, 5, 6, 4]
6	[1, 2, 2, 4, 4, 5, 6]

3. Le tri par sélection consiste à : Méthode

- on cherche le minimum de la liste et, par échange, on le place à la première place
 - puis on place le minimum des éléments restants à la second place
 - on recommence, ainsi de suite, jusqu'à ce que les éléments restants soient réduits à un élément.
4. Tri appliqué à L :

i	min(L[i:])	L
X	X	[2, 6, 1, 5, 4, 2, 4]
0	1	[1, 2, 6, 5, 4, 2, 4]
1	2	[1, 2, 6, 5, 4, 2, 4]
2	2	[1, 2, 2, 6, 5, 4, 4]
3	4	[1, 2, 2, 4, 6, 5, 4]
4	4	[1, 2, 2, 4, 4, 6, 5]
5	5	[1, 2, 2, 4, 4, 5, 6]

Solution 4 Tri fusion suivant l'ordre lexicographique

1.

Tri fusion suivant l'ordre lexicographique

```
def inf_ou_egal(a,b):
    for k in range(len(a)):
        if a[k]<b[k]:
            return True
        elif a[k]>b[k]:
            return False
    return True
```

Exemple :

```
--> inf_ou_egal([1,2,3],[1,2,4])
True
--> inf_ou_egal([1,2,3],[1,2,0])
False
```

2. Fusion de deux listes triées :

Fusion récursive de deux listes déjà triées

```
def fusion_rec(L1,L2):
    if len(L1)==0:
        return L2
    if len(L2)==0:
        return L1
    if inf_ou_egal(L1[0],L2[0]):
        return L1[:1]+fusion_rec(L1[1:],L2)
    else:
        return L2[:1]+fusion_rec(L1,L2[1:])
```

Exemple :

```
-->fusion_rec([[1,2],[1,5]],[[0,2],[1,4]])
[[0, 2], [1, 2], [1, 4], [1, 5]]
```

3. Méthode du tri par fusion :

Tri par fusion

```
def tri_fusion(L):
    L,n=list(L),len(L)
    if n==1:
        return L
    else:
        L1=tri_fusion(L[:n//2])
        L2=tri_fusion(L[n//2:])
        return fusion_rec(L1,L2)
```

Exemple :

```
--> tri_fusion([[1,2],[2,1],[1,0],[1,1]])
[[1, 0], [1, 1], [1, 2], [2, 1]]
```

Solution 5 Tri rapide

1. Suite des appels récursifs et les étapes de la remontée :

Profondeur	Liste
0	[5,4,8,4,3,7,2,6,1,9]
1	[4,4,3,2,1],[5],[8,7,6,9]
2	[4,3,2,1],[4],[],[5],[7,6],[8],[9]
3	[3,2,1],[4],[],[4],[],[5],[6],[7],[8],[9]
4	[2,1],[3],[],[4],[],[4],[],[5],[6],[7],[8],[9]
5	[1],[2],[],[3],[],[4],[],[4],[],[5],[6],[7],[8],[9]
4	[1,2],[3],[],[4],[],[4],[],[5],[6],[7],[8],[9]
3	[1,2,3],[4],[],[4],[],[5],[6],[7],[8],[9]
2	[1,2,3,4],[4],[],[5],[6,7],[8],[9]
1	[1,2,3,4,4],[5],[6,7,8,9]
0	[1,2,3,4,4,5,6,7,8,9]

2. Script basique :

tri_rapide.py

```
def tri_rapide(L):
    if len(L)<2:
        return(L)
    I,S=[],[]
    for e in L[1:]:
        if e<=L[0]:
            I.append(e)
        else:
            S.append(e)
    return tri_rapide(I)+[L[0]]+tri_rapide(S)
```

3. Version itérative :

```
1 def tri_rapide_ite(L):
2     if len(L)>1:
3         T=[(0,len(L))]
4     else:
5         T=[]
6     while len(T)>0:
7         i,j=T.pop()
8         I,S=[],[]
9         for e in L[i+1:j]:
10            if e<=L[i]:
11                I.append(e)
12            else:
13                S.append(e)
14        L[i:j]=I+[L[i]]+S
15        if len(I)>1:
16            T.append((i,i+len(I)))
17        if len(S)>1:
18            T.append((len(I)+i+1,j))
19    return L
```