IPT Devoir 3- Proposition de solutions

Solution 1 Étude d'un algorithme

1. Déroulement de l'algorithme pour $a = 7$ et $b = 23$. Le contenu des variables u, v, p est donnée dans le tableau suivant :

	u	v	p	
Initialisation	7	23	0	
Boucle 1	14	11	7	cas : v impair
Boucle 2	28	5	21	cas : v impair
Boucle 3	56	2	49	cas : v impair
Boucle 4	112	1	49	cas : v pair
Boucle 5	224	0	161	cas : v impair

L'algorithme s'arrête dès que $p = 0$. Le résultat est 161 c'est-à-dire 7×23. Par conjecture, cet algorithme effectue la multiplication de a par b.

2. Montrons que $p + u \times v = ab$ est un invariant de boucle.
On note u_k, v_k et p_k la valeur des variables u, v et p au début de la boucle de rang k.

- Initialisation : lors de l'initialisation,
$$p_1 + u_1 \times v_1 = 0 + a \times b = ab$$

- Hérédité : soit $k \in \mathbb{N}^*$, on suppose que $p_k + u_k \times v_k = ab$.

Considérons l'étape $k+1$, discutons suivant la parité de v_k.
Soit v_k est impair, notons $v_k = 2j + 1$, alors

$$\begin{cases} p_{k+1} = p_k + u_k \\ u_{k+1} = 2u_k \\ v_{k+1} = j = \dfrac{v_k - 1}{2} \end{cases}$$

Donc, $$\begin{aligned} p_{k+1} + u_{k+1} \times v_{k+1} &= p_k + u_k + 2u_k \frac{v_k - 1}{2} \\ &= p_k + u_k \times v_k = ab \end{aligned}$$

Soit v_k est pair, notons $v_k = 2j$, alors

$$\begin{cases} p_{k+1} = p_k \\ u_{k+1} = 2u_k \\ v_{k+1} = j = \dfrac{v_k}{2} \end{cases}$$

Donc, $p_{k+1} + u_{k+1} \times v_{k+1} = p_k + 2u_k \dfrac{v_k}{2} = p_k + u_k \times v_k = ab$

Dans tous les cas, $p_{k+1} + u_{k+1} \times v_{k+1} = ab$.

- Conclusion, $p + u \times v = ab$ est un invariant de boucle.

Or l'algorithme se termine avec $v = 0$, et donc $p = ab$.
Le résultat de l'algorithme est bien le produit de a par b.

3. Montrons que l'algorithme se termine. La suite des valeurs de la variable v est strictement décroissante et à valeurs dans $\mathbb{N}$, donc elle est de longueur finie. L'algorithme se termine.

4. Script de la fonction

```
1 def f(a,b):
2     u,v,p=a,b,0
3     while v>0:
4         if v%2==1:
5             p=p+u
6         u,v=2*u,v//2
7     return p
```

5. Script d'une fonction récursive

```
1 def f_rec(a,b):
2     if b==0:
3         return(0)
4     else:
5         if b%2==1:
6             return a+f_rec(2*a,b//2)
7         else:
8             return f_rec(2*a,b//2)
```

Explications sur l'écriture de la fonction récursive `f_rec(a,b)` : une idée est de considérer l'algorithme sur une seule étape et d'associer les variables locales avec les paramètres d'entrée : $v \leftrightarrow b$ et $u \leftrightarrow a$.

- Gestion de la condition d'arrêt : on note que la variable v conditionne le déroulement de l'algorithme ; sa taille diminue de moitié à chaque étape, initialement à b. On pose donc pour condition d'arrêt : $b = 0$.
- La valeur de sortie est p. Il y a une discussion en fonction de la parité de v (donc de b).

Si v est paire, la valeur de sortie est la même que celle de la nouvelle instance en $(2u, v//2)$, donc on appelle `f_rec(2*a,b//2)`.
Si v est impaire, la valeur de sortie est augmentée de u (donc de a) par rapport au prochain appel `f_rec(2*a,b//2)`.

Solution 2 Variation sur la recherche d'une minium

1. Pour rechercher le maximum, il suffit de modifier le test :

```
1 def maxi(L):
6       if L[i]>=v:
```

2. Pour rechercher la position d'un minimum de la liste, il suffit de travailler par rapport à la position, plus que par rapport à la valeur :

```
1 def pos_mini(L):
2     if len(L)==0:
3         return None
4     p=0
5     for i in range(1,len(L)):
6         if L[i]<=L[p]:
7             p=i
8     return L[p]
```

3. Dans le cas d'une liste de couple, il suffit à nouveau de modifier la ligne où le teste est effectué :

```
1 def mini2(L):
6       if L[i][1]<=v[1]:
```

Solution 3 Longueur maximale d'une série de zéros

1. La fonction nombreZeros :

```
def nombreZeros(t,i):
    z=0
    while i<len(t) and t[i]==0:
        z,i=z+1,i+1
    return(z)
```

En particulier :

```
--> t=[0,1,1,1,0,0,0,1,0,1,1,0,0,0,0]
--> nombreZeros(t,4)
3
--> nombreZeros(t,1)
0
--> nombreZeros(t,8)
1
```

Avec une boucle for cela donnerait :

```
def nombreZeros(t,i):
    z=0
    for k in range(i,len(t)):
        if t[k]==1:
            break
        else:
            z=z+1
    return z
```

2. Dans le cas où l'on connais le liste des nombreZeros(t,i) pour toute les valeurs de i possibles, alors le maximum de cette liste donne le nombre maximal de zéros :

```
def nombreZerosMax(t):
    L=[]
    for i in range(len(t)):
        L.append(nombreZeros(t,i))
    return max(L)
```

Avec un algorithme explicite :

```
def nombreZerosMax(t):
    n=0
    for i in range(len(t)):
        nbz=nombreZeros(t,i)
        if nbz>n:
            n=nbz
    return n
```

3. Analyse de la complexité de nombreZerosMax : notons n la longueur de la liste t

- une boucle parcourant la liste t, indexée par i
- pour i fixé, l'appel nombreZeros(t,i) requiert entre 1 et $n-i$ comparaisons (et deux affectations)

Dans le meilleurs des cas : $\sum_{i=0}^{n-1} 1 = n$.

Dans le pire des cas :

$$\sum_{i=0}^{n-1} n-i = \sum_{j=1}^{n} j = \frac{n(n+1)}{2}$$

La complexité de nombreZerosMax est en $\mathcal{O}(n^2)$.

4. Il suffit d'intégrer au parcours de la liste t l'information obtenue à chaque appel de nombreZeros : on peut effectuer un saut afin d'ignorer des valeurs ne pouvant pas donner le début d'une liste maximale de zéros. Sur l'exemple donné, on note nbZ(t,i) les appels "*utiles*" :

i	0	1	2	3	4	5	6	7
t1[i]	0	1	1	1	0	0	0	1
nbZ(t,i)	1		0	0	3			

i	8	9	10	11	12	13	14
t1[i]	0	1	1	0	0	0	0
nbZ(t,i)	1		0	4			

```
def nombreZerosMax2(t):
    z=nombreZeros(t,0)
    nb,i=z,z+1
    while i<len(t)-nb:
        z=nombreZeros(t,i)
        i+=z+1
        if z>nb:
            nb=z
    return nb
```

En particulier :

```
--> nombreZerosMax2(t)
4
```

Pour tester des listes aléatoires de longueur 40, suivre les instruction suivantes :

```
--> import numpy.random as rd
--> t=list(rd.randint(0,2,size=40))
--> t
--> nombreZerosMax2(t)
```

➜ Analyse de complexité de la nouvelle version : Les deux boucles imbriquées (celle du parcours de la liste t et celle des appels de nombreZeros) se complète pour effectuer un simple parcours de t. Il y a donc n étapes. A chaque étape, il y a une ou deux comparaisons et une ou quatre affections.
La complexité de nombreZerosMax2 est en $\mathcal{O}(n)$.

Solution 4 Il convient d'introduire un indice pour contrôler la boucle :

```
u,i=2,1
while i<n:
    u=u+1/i
    i=i+1
```