# TD Informatique TSI2

## Recursion and sorting recursion

# SOLUTION

## EXERCICE 01

**1.**

```
In [1]: def somme(n):
   ...:     s = 0
   ...:     for k in range(n+1):
   ...:         s += k
   ...:     return s
```

**2.**

```
In [2]: def sommeRec(n):
   ...:     if n == 0 :
   ...:         return 0
   ...:     else :
   ...:         return n + sommeRec(n-1)
   ...:

In [3]: somme(20), sommeRec(20)
Out[3]: (210, 210)
```

## EXERCICE 02

**1.**

```
In [4]: def fact(n):
   ...:     p=1
   ...:     for k in range(1,n+1):
   ...:         p *= k
   ...:     return p
In [5]: fact(1),fact(5), fact(14)
Out[5]: (1, 120, 87178291200)
```

**2.**

```
In [6]: def factRec(n):
   ...:     if n == 0:
   ...:         return 1
   ...:     else :
   ...:         return n * factRec(n-1)
   ...:
In [7]: factRec(1),factRec(5),factRec(14)
Out[7]: (1, 120, 87178291200)
```

# EXERCICE 03

**1.**

```
In [8]: def puissance(x,n):
   ...:     p=1
   ...:     for k in range(1,n+1):
   ...:         p *= x
   ...:     return p
   ...:

In [9]: puissance(2,4),puissance(7,3)
Out[9]: (16, 343)
```

**2.**

```
In [10]: def puissanceRec(x,n):
    ...:     if n == 0 :
    ...:         return 1
    ...:     else :
    ...:         return x * puissanceRec(x,n-1)

In [11]: puissanceRec(2,4),puissanceRec(7,3)
Out[11]: (16, 343)
```

# EXERCICE 04

**1.**

```
In [12]: def quotient2(n):
    ...:     q=0
    ...:     r=n
    ...:     while r > 1 :
    ...:         q += 1
    ...:         r -= 2
    ...:     return q

In [13]: def quotient2Rec(n):
    ...:     if n <= 1 :
    ...:         return 0
    ...:     else :
    ...:         return 1 + quotient2Rec(n-2)

In [14]: quotient2(179), quotient2Rec(179)
Out[14]: (89, 89)
```

**2.**

```
In [15]: def reste2(n):
    ...:     r=n
    ...:     while r > 1 :
    ...:         r -= 2
    ...:     return r

In [16]: def reste2Rec(n):
    ...:     if n <= 1 :
    ...:         return n
    ...:     else :
    ...:         return reste2Rec(n-2)

In [17]: reste2(179),reste2Rec(179)
Out[17]: (1, 1)
```

## EXERCICE 05

**1.**

```
In [2]: def nbChiffresRec(n):
   ...:     if n <= 9 :
   ...:         return 1
   ...:     else :
   ...:         return 1 + nbChiffresRec(n//10)
In [3]: nbChiffresRec(12485)
Out[3]: 5
```

**2.**

```
In [4]: def nbChiffres(n):
   ...:     a = n ; c = 1
   ...:     while a > 9 :
   ...:         a = a // 10
   ...:         c += 1
   ...:     return c
In [5]: nbChiffres(12485)
Out[5]: 5
```

## EXERCICE 06

Commençons par taper le code pour voir.

```
In [1]: def f(a,b):
   ...:     if b == 0:
   ...:         return 0
   ...:     else :
   ...:         return a+f(a,b-1)
   ...:
In [2]: f(3,5)
Out[2]: 15
In [3]: f(17,2), f(4,4)
Out[3]: (34, 16)
```

**1.** On a : $f(3,5) = 3 + f(3,4)$ puis $f(3,4) = 3 + f(3,3)$ puis $f(3,3) = 3 + f(3,2)$ puis $f(3,2) = 3 + f(3,1)$ puis $f(3,1) = 3 + f(3,0)$ et enfin $f(3,0) = 0$. Alors :

$$f(3,5) = 3 + (3 + (3 + (3 + (3 + 0)))) = 15.$$

**2.** La fonction se termine toujours pour $b = 0$. Dans le cas général, elle se réappelle avec $b - 1$ au lieu de $b$. Si $b \in \mathbb{N}$, on finira par $b = 0$. Donc la fonction se terminera. Si $b \notin \mathbb{N}$, la fonction ne se terminera pas.

**3.** Il semble pour $b \in \mathbb{N}$ que `f(a,b)=ab`. Montrons le par récurrence sur $b$.

Initialisation : C'est vrai pour $b = 0$.
Transmission : supposons vrai pour un $b$ donné. Alors : $f(a, b+1) = a + f(a,b) = a + ab = a(b+1)$.

## EXERCICE 07

**1.**

```
In [6]: def plusGrandEntre(l,deb,fin):
   ...:     if fin == deb :
   ...:         return l[fin]
   ...:     else :
   ...:         return max(l[fin], plusGrandEntre(l,deb,fin - 1))
In [7]: plusGrandEntre([4,0,8,1,-8,-4,7],0,4)
Out[7]: 8
In [8]: plusGrandEntre([4,0,8,1,-8,-4,7],3,5)
Out[8]: 1
```

**2.**

```
In [11]:  def plusGrand(l):
   ...:     return plusGrandEntre(l,0,len(l)-1)

In [12]: plusGrand([4,0,8,1,-8,-4,7])
Out[12]: 8
```

## EXERCICE 08

**1.**

```
In [13]: def chercheEntre(x,l,deb,fin):
    ...:     if fin < deb :
    ...:         return False
    ...:     else :
    ...:         m = (deb + fin) // 2
    ...:         if x == l[m] :
    ...:             return True
    ...:         elif x < l[m] :
    ...:             return chercheEntre(x,l,deb,m-1)
    ...:         else :
    ...:             return chercheEntre(x,l,m+1,fin)
    ...:
In [14]: chercheEntre(4,[2,4,5,7,8,10,12,15], 3,7)
Out[14]: False
In [15]: chercheEntre(4,[2,4,5,7,8,10,12,15], 0,7)
Out[15]: True
```

**2.** Le pire des cas est celui où l'élément est absent (nombre maximal de recherches).
À chaque étape, la longueur de la plage est divisée par 2. Supposons que la longueur de la plage est initialement $n$, elle est de longueur $n/2^p$ au bout de $p$ appels récursifs. Si l'élément est absent, l'algorithme se termine lorsque cette longueur est strictement inférieure à 1. Et donc $n/2^p < 1$. Le nombre d'appels récursifs est le plus petit entier $p$ vérifiant ceci. Pour cet entier, on a donc $n/2^{p-1} \geqslant 1$, c'est-à-dire $2^{p-1} \leqslant n$ ou encore $p - 1 \leqslant \log_2(n)$. La complexité est donc logarithmique (donc très bonne).

# EXERCICE 09

**1.**

```
In [16]: def chiffresFortVersFaible(n):
    ...:     if n == 0 :
    ...:         return []
    ...:     else :
    ...:         L = chiffresFortVersFaible(n // 10)
    ...:         L.append(n % 10)
    ...:         return(L)
In [17]: chiffresFortVersFaible(547)
Out[17]: [5, 4, 7]
```

**2.**

```
In [1]: def f(L,dernier):
   ...:     if dernier <0:
   ...:         return 0
   ...:     else :
   ...:         q = f(L,dernier-1)
   ...:         return 10 * q + L[dernier]
   ...:

In [2]: f([4,5,8],0), f([4,5,8],2)
Out[2]: (4, 458)
```

**2.** • Calcul de `f([4,5,8],0)`

On a `dernier` vaut 0 et $0 < 0$ est False. Donc `q=f([4,5,8],-1)` et $-1 < 0$ est True donc `q=f([4,5,8],-1)` renvoie 0 et donc `q=f([4,5,8],0)` retourne $10 \times 0 + 4 = 4$.

• Calcul de `f([4,5,8],1)`

En effet, ce calcul est utile pour `f([4,5,8],2)` à cause de la récursivité.

Ici `dernier` vaut 1 et $1 < 0$ est False. Donc `q=f([4,5,8],0)` et vaut donc 4 et on retourne enfin $10 \times 4 + 5 = 45$.

• Calcul de `f([4,5,8],2)`

Ici `dernier` vaut 2 et $2 < 0$ est False. Et donc `q=f([4,5,8],1)` qui vaut 45 et on retourne $10 \times 45 + 8 = 458$.

Bilan : `f` fait l'opération inverse de `chiffresFortVersFaible`

# EXERCICE 10

```
In [51]: def New_partition(l,deb,fin):
    ...:     a=l[deb] ; m =deb
    ...:     for i in range(deb+1,fin+1):
    ...:         if l[i] <= a:
    ...:             m += 1
    ...:             l[i],l[m] = l[m],l[i]
    ...:     l[deb],l[m] = l[m], l[deb]
    ...:     return (m,l)
```

**1.** Avec Python, que faut-il trouver ?

```
In [52]: New_partition([3,-1,2,3,5,4,8,0,4,2,5,-5],8,11)
Out[52]: (10, [3, -1, 2, 3, 5, 4, 8, 0, -5, 2, 4, 5])
```

À la main : on a `a=l[8] =l[deb]=4` et `m=8=deb` pour commencer.

On prend `i=9` et `l[9] = 2` et comme $2 \leqslant 4$, on prend `m=9` et on fait `l[i],l[m] = l[m],l[i]` soit ici `2,2=2,2` et `l` n'est pas modifié.

On prend ensuite `i=10` et `l[10] = 5` et comme $5 \leqslant 4$ est False, on garde `m=9` et `l` n'est pas modifié.

On prend enfin `i=11` et `l[11] = -5` et comme $-5 \leqslant 4$, on prend `m=10` et on fait `l[i],l[m] = l[m],l[i]` soit ici `-5,5 = 5 ,-5` et `l=[3,-1,2,3,5,4,8,0,4,2,-5,5]`

La boucle est finie et on fait `l[deb],l[m] = l[m], l[deb]` soit `l[8],l[10] = l[10], l[8]` et il reste `l=[3,-1,2,3,5,4,8,0,-5,2,4,5]`

Bilan : On voit au départ que l'on travaille seulement dans la sous-liste `[4,2,5,-5]` et 4 est la valeur choisie qu'on compare aux autres. À la fin, la sous-liste devient `[-5,2,4,5]` et les éléments $2, 5, -5$ de la sous-liste qui sont plus petits que 4 sont avant lui et ceux qui sont plus grands après lui.

**2.** Avec Python, que faut-il trouver ?

```
In [53]: New_partition([3,-1,2,3,5,4,8,0,4,2,5,-5],0,11)
Out[53]: (6, [-5, -1, 2, 3, 0, 2, 3, 5, 4, 4, 5, 8])
```

À la main : on a `a=l[0] =l[deb]=3` et `m=0=deb` pour commencer.

D'abord `i=1` et `l[1] = - 1` et $-1 \leqslant 3$ donc `m=1` et `l[1],l[1] = l[1],l[1]` et `l` ne bouge pas.

Puis `i=2` et `l[2] = 2` et $2 \leqslant 3$ donc `m=2` et `l[2],l[2] = l[2],l[2]` et `l` ne bouge pas.

Puis `i=3` et `l[3] = 3` et $3 \leqslant 3$ donc `m=3` et `l[3],l[3] = l[3],l[3]` et `l` ne bouge pas.

Puis `i=4` et `l[4] = 5` et $5 \leqslant 3$ est False donc on a encore `m=3`

Puis `i=5` et `l[5] = 4` et $4 \leqslant 3$ est False donc on a encore `m=3`

Puis `i=6` et `l[6] = 8` et $8 \leqslant 3$ est False donc on a encore `m=3`

Puis `i=7` et `l[7] = 0` et $0 \leqslant 3$ donc `m=4` et `l[7],l[4] = l[4],l[7]` ou encore `0,5 = 5,0` et `l=[3,-1,2,3,0,4,8,5,4,2,5,-5]`

Puis `i=8` et `l[8] = 4` et $4 \leqslant 3$ est False donc on a encore `m=4`

Puis `i=9` et `l[9] = 2` et $2 \leqslant 3$ donc `m=5` et `l[9],l[5] = l[5],l[9]` ou encore `4,2 = 2,4` et `l=[3,-1,2,3,0,2,8,5,4,4,5,-5]`

Puis `i=10` et `l[10] = 5` et $5 \leqslant 3$ est False donc on a encore `m=5`

Puis `i=11` et `l[11] = -5` et $-5 \leqslant 3$ donc `m=6` et `l[11],l[6] = l[6],l[11]` ou encore `8,-5 = -5,8` et `l=[3,-1,2,3,0,2,-5,5,4,4,5,8]`

La boucle est finie et on fait `l[deb],l[m] = l[m], l[deb]` soit `l[3],l[6] = l[6], l[3]` et il reste `l=[-5,-1,2,3,0,2,3,5,4,4,5,8]`

Bilan : on travaille avec la liste complète et 3 est la valeur choisie qu'on compare aux autres.
À la fin, on a mis avant 3 tous ceux plus petits que lui (au sens large) et tous ceux strictement plus grands à sa droite.

**3.** Avec Python, que faut-il trouver ?

```
In [54]: New_partition([3,-1,2,3,5,4,8,0,4,2,5,-5],3,8)

Out[54]: (4, [3, -1, 2, 0, 3, 4, 8, 5, 4, 2, 5, -5])
```

On refait à la main quelque chose de similaire à plus haut. On voit au départ que l'on travaille seulement dans la sous-liste `[3,5,4,8,0,4]` et 3 est la valeur choisie qu'on compare aux autres. À la fin, la sous-liste devient `[0,3,4,8,5,4]` et l'élément 0 de la sous-liste est plus petit que 3 et est avant lui et les autres qui sont plus grands après lui.

## EXERCICE 11

Avec Python, que faut-il trouver ?

```
In [63]: def fusion(l,deb,mil,fin):
    ...:     aux = []
    ...:     i,j = deb, mil+1
    ...:     while i <= mil and j <= fin :
    ...:         if l[i] < l[j]:
    ...:             aux.append(l[i]) ; i += 1
    ...:         else :
    ...:             aux.append(l[j]) ; j += 1
    ...:     while i <= mil :
    ...:         aux.append(l[i]) ; i += 1
    ...:     while j <= fin :
    ...:         aux.append(l[j]) ; j += 1
    ...:     for k in range(deb,fin + 1) :
    ...:         l[k] = aux[k-deb]
    ...:     return l

In [64]: fusion([-2,2,3,7,9,10,1,8,1,4],1,5,7)
Out[64]: [-2, 1, 2, 3, 7, 8, 9, 10, 1, 4]
```

Travaillons à la main.

La sous-liste qu'on retient est `[2,3,7,9,10,1,8]` d'indices `[1 2 3 4 5 6 7]` dans la nomenclature de la liste `l` de départ.

Alors `deb=1`, `i=1`, `j=6`, `mil=5`, `fin=7` et `aux=[]`

Commençons par le premier `while` qui fonctionnera tant que $i \leqslant 5$ et $j \leqslant 7$

On a d'abord `i=1` et `j=6` et comme `l[1] < l[6]` ou encore $2 < 1$ est False, on a `aux=[1]` et `j=7` et `i=1`

Puis infol[1] < `l[7]` ou encore $2 < 8$ est True et donc `aux=[1,2]` et `j=7` et `i=2`

Puis infol[2] < `l[7]` ou encore $3 < 8$ est True et donc `aux=[1,2,3]` et `j=7` et `i=3`

Puis infol[3] < `l[7]` ou encore $7 < 8$ est True et donc `aux=[1,2,3,7]` et `j=7` et `i=4`

Puis infol[4] < `l[7]` ou encore $9 < 8$ est False et donc `aux=[1,2,3,7,8]` et `j=8` et `i=3`

La première boucle est finie car $j > 7$

On passe à la seconde boucle `while`

D'abord `i=4` et $4 \leqslant 5$ et donc on ajoute `l[4]=9` à `aux` et donc `aux=[1,2,3,7,8,9]` puis `i=5` et on ajoute `l[5]=10` à `aux` qui devient `aux=[1,2,3,7,8,9,10]` puis `i=6` et on arrête la boucle car `i>mil`

Passons à la troisième boucle `while`

Elle ne fonctionne pas car `j=8` et $8 > 7$.

Il reste la boucle `for` avec `deb=1` et donc `k` varie de 1 à 7.
On a `l[1]=aux[0]=1` puis `l[2]=aux[1]=2` et jusqu'à `l[7]=aux[6]=10` et notre liste complète est :

```
l=[-2, 1, 2, 3, 7, 8, 9, 10, 1, 4]
```