# Solutions pour le TP 6 : écriture en base deux (partie 1)

## 1 Ecriture en base deux

**Question 1 :**

Bien comprendre comment on fabrique la boucle : à chaque étape, on prend le reste de la division euclidienne de `n` par 2, qu'on doit stocker dans une liste, et on remplace `n` par `n//2`. Une fois qu'on a compris cela, on a compris la boucle « en régime de croisière », il ne reste plus qu'à initialiser et mettre la condition d'arrêt.

```
## Ecriture Base 2 en commençant par les poids faibles
def base2_poids_faible(n):
    liste=[]
    if n==0:
        liste=[0] # Cas à mettre à part car n'entre pas dans la boucle qui suit
    else :
        while (n>0):
            a=n%2
            liste=[a]+liste # on rajoute a du bon côté...
            n=n//2 # quotient de la div. eucl.
    return liste
```

**Question 2 :**

Ce que j'ai vu parfois :

```
def puissance2inf1(n):
    if n==0:
        return 0
    else:
        i=0
        while 2**i <n:
            i=i+1
        if 2**i==n:
            return i
        else :
            return i-1
```

Ce code illustre bien la différence entre ce qui se passe si notre nombre est « pile » une puissance de 2 ou pas. Mais on peut éviter les deux cas à la fin, avec un test `2**i<=n` plus malin (beaucoup vu aussi) :

```
def puissance2inf2(n):
    if n==0:
        return 0
    else:
        i=0
        while 2**i <=n:
            i+=1
    return i-1 # rattrapage final...
```

Informatiquement, les deux codes précédents ont un gros défaut : le calcul de `2**i` à chaque étape. Il est préférable de créer un *accumulateur* qui est multiplié par 2 à chaque étape. C'est ce qu'on fait dans la v.3 ci-dessous.

```
def puissance2inf3(n):
        if n==0:
        return 0
    else:
        i=0
        acc=1
        while acc <=n:
            i=i+1
            acc=2*acc
    return i-1
```

On peut aussi penser le problème *en sens inverse* et « éviter le rattrapage final » avec le $-1$.

```
def puissance2inf4(n):
    r=0
    while n>1:
       r = r+1
       n = n//2
    return r
```

**Explication :** cette fois on pense au résultat du puissance2inf comme au nombre de fois qu'on peut diviser $n$ par 2 avant de tomber sur 0.

**Question 3 :**

**Une solution souvent écrite :**

```
def poids_fort(n):
    #création  d'une liste auxiliaire avec la bonne longueur
    if n==0:
        liste=[0]
    else:
        r=puissance2inf(n)
        liste=[1]+[0]*r
        l=r+1 # la longueur de la liste
        # on a créé une liste du bon format, on va continuer à la remplir
        n=n-2**r #
        while n!=0:
            r=puissance2inf(n)
            liste[l-1-r]=1
            n=n-2**r
    return liste
```

**Remarque 1** L'algo. des poids forts, au départ plus intuitif pour un humain est : plus difficile à programmer, et surtout plus lent (facteur 5 pour un test avec `time`). On peut bien sûr essayer de l'optimiser un peu. Par exemple le `n=n-2**r` est très mauvais : on recalcule `2**r`. Pour éviter cela, on pourrait modifier `puissance2inf` pour lui faire renvoyer non seulement l'exposant $r$, mais la valeur de $2^r$, puisque il est facile de la faire calculer à moindre coût par cette fonction : inutile de la calculer deux fois. Mais même ainsi l'algo. reste plus lent. La division euclidienne est une arme très efficace !