Exponentiation naïve, exponentiation rapide

Question 1

In [1]:
```
def puiss(x,n):
    """renvoi x**n avec x réel et n entier positif"""
    sol=1
    for i in range(n):
        sol=sol*x
    return sol
```

In [2]:
```
puiss(2,10)
```

Out[2]: 1024

Question 2

In [3]:
```
def puiss_rec(x,n):
    """renvoi x**n avec x réel et n entier positif"""
    if n==0:
        return 1 # condition d'arrêt avec x**0=1
    else:
        return x*puiss_rec(x,n-1)# appel récursif
```

In [4]:
```
puiss_rec(2,10)
```

Out[4]: 1024

Question 3 : variant de boucle

n est un variant de boucle. A chaque appel de la fonction récursive , il décroit d'une unité et finit par atteindre la valeur 0 correspondant à une condition d'arrêt. Le programme se termine si $n > 0$.

Question 4 : invariant de boucle

On considère la propriété (l'invariant de boucle) $P_n = x^n$

- Initialisation : P_0 est vrai puisque c'est la condition d'arrêt
- Récurrence : on suppose la propriété vraie au rang k, qu'en est-il au rang $k+1$?

  La fonction **puis_rec(n+1)** réalise l'opération $xpuiss\_rec(n) = xx^n = x^{n+1}$ puisque la propriété P_n est vraie au rang n.
  La propriété P_{n+1} est donc vérifiée.
- Sortie de boucle : La fonction récursive est appelée n fois entre n=n et $n = 1$, le résultat renvoyé est donc x^n et l'invariant de boucle est vérifié.

Question 5 : Complexité

On définit $T(n)$ le nombre d'opérations élémetaires de la fonction $puiss\_rec$

A chaque appel de la fonction récursive, on a un test pour savoir si $n = 0$ et une multiplication soit la relation de récurrence $T(n) = T(n-1) + 2$ avec $T(0) = 1$ (une comparaison pour $n = 0$). On alors $T(n) = 1 + 2n = O(n)$. La complexité est linéaire.

Question 6 : Complexité : puiss_rapide

In [5]:
```
def puiss_rapide(x,n):
    if n==0:
        return 1
    elif n%2==0:
        return puiss_rapide(x,n//2)**2
    else:
        return x*puiss_rapide(x,(n-1)//2)**2
```

In [6]:
```
puiss_rapide(2,10)
```

Out[6]: 1024

On définit $T(n)$ le nombre d'opérations élémetaires de la fonction $puiss\_rapide$.

A chaque appel de la fonction récursive, on a :

- un test pour savoir si $n = 0$
- un test pour savoir si n est pair

- deux appels de la fonction récursive pour calculer le carré
- un ou deux multiplications selon les cas

Si on se place dans le pire des cas, on a $T(i) = 2T\left(\frac{i}{2}\right) + 4$ où i est le numéro d'appel récursif.

Pour simplifier les calculs on considère que l'entier n est une puissance de 2. On calcule le nombre de fois où on fait un appel de la fonction récursive :

- Après un appel de la fonction récursive, on l'appelle à nouveau avec n/2
- Après un deuxième appel, on l'appelle à nouveau avec $n/2^2$
- Après k appels, on l'appelle à nouveau avec $n/2^k$

On sort de la fonction récursive pour $n/2^k = 1$ soit $\ln(n) = k\ln 2$ où $k = \log_2(n)$, La fonction nécessite k appels récursifs.

Pour calculer la complexité, on considère la suite définie par récurrence : $u_n = au_{n-1} + b$ avec $a = 2$, $b = 4$ et $u_0 = 1$ et on pose $r = \frac{b}{1-a}$. On rappelle qu'on peut alors écrire $u_n = a^n(u_0 - r) + r$ (pour les sceptiques, on peut le montrer par récurrence...).

On a alors $T(\log_2(n)) = 2^{\log_2(n)}(1 - \frac{4}{1-2}) + \frac{4}{1-2} = 5n - 4 = O(n)$

On obtient également une complexité linéaire, ce qui n'améliore pas la complexité...

Question 7 : exponentiation rapide : un seul appel récursif

In [7]:
```
def puiss_rapide2(x,n):
    if n==0:
        return 1
    else:
        sol=puiss_rapide2(x,n//2)
        if n%2==0:# n est pair
            return sol**2
        else:
            return x*sol**2
```

In [8]:
```
puiss_rapide2(2,10)
```

Out[8]:
```
1024
```

On définit $T(n)$ le nombre d'opérations élémetaires de la fonction $puiss\_rapide2$.

A chaque appel de la fonction récursive, on a :

- un test pour savoir si $n = 0$
- un appel de la fonction récursive
- un test pour savoir si n est pair
- une ($sol * *2$) ou deux multiplications ($x * sol * *2$) selon les cas

Si on se place dans le pire des cas, on a $T(i) = T\left(\frac{i}{2}\right) + 4$ où i est le numéro d'appel récursif.

Pour simplifier les calculs on considère que l'entier n est une puissance de 2. On calcule le nombre de fois où on fait un appel de la fonction récursive :

- Après un appel de la fonction récursive, on l'appelle à nouveau avec n/2
- Après un deuxième appel, on l'appelle à nouveau avec $n/2^2$
- Après k appels, on l'appelle à nouveau avec $n/2^k$

On sort de la fonction récursive pour $n/2^k = 1$ soit $\ln(n) = k\ln 2$ où $k = \log_2(n)$, La fonction nécessite k appels récursifs.

Pour calculer la complexité, on considère la suite définie par récurrence : $u_n = u_{n-1} + 4$ avec $u_0 = 1$. On a cette fois une suite arithmétique et on peut alors écrire $u_n = 4n + 1$

On ainsi $T(\log_2(n)) = 4\log_2(n) + 1 = O(\log_2(n)$. La complexité est cette fois ci logarithmique

Remarque : On utilise la méthode "diviser pour régner" qui se décompose en trois étapes :

- Diviser (ou partitionner) : on divise le problème initial en plusieurs sous-problèmes
- Régner: on traite récursivement chacun des sous-problèmes.
- Combiner : on combine les différents sous-problèmespour résoudre le problème de départ.

Cette méthode sera revu en TP pour les tris fusion et tris rapides...