TD1 : décompte d'opérations et notations asymptotiques

Exercice 1 : Dénombrement avec les boucles

Pour chacune des fonctions, compter le nombre exact d'additions réalisées en fonction de n. Lorsqu'il est trop difficile de calculer le nombre exact, proposer une fonction de n qui majore ce nombre.

```
def f1 (n):
  r = 0
  for i in range(1,n+1):
    for j in range(1,n+1):
      r = r + 3
  return r
```

```
def f2 (n):
  r = 0
  for i in range(1,n+1):
    for j in range(1,i+1):
      r = r + 3
  return r
```

```
def f3 (n):
  r = 0
  for i in range(5,n-5+1):
    for j in range(i-5,i+5+1):
      r = r + 3
  return r
```

```
def f4 (n):
  r = 0
  for i in range(1,n+1):
    for j in range (1,n+1):
      for k in range (1,n+1):
        r = r + 3
  return r
```

```
def f5 (n):
  r = 0
  for i in range(1,n+1):
    for j in range (1,i+1):
      for k in range(1,j+1):
        r = r + 3
  return r
```

```
def f6 (n):
  r = 0
  i = n
  while i > 1:
    for j in range(1,i+1):
      r = r + 3
    i = i // 2
  return r
```

Exercice 2 :

Nous disposons de plusieurs algorithmes réalisant la même tache mais avec un nombre d'additions différent (colonne de gauche des tableaux).

Q 2.1 Indiquer le nombre d'opérations nécessaires à l'exécution de chaque algorithme pour les différentes tailles de données.

Complexité	$n = 10$	$n = 100$	$n = 1000$
n			
n^2			
n^3			
$\sqrt{n}$			
2^n			
$\log_2(n)$			

Q 2.2 Indiquer le temps nécessaire à l'exécution de chaque algorithme pour les différentes tailles de données sachant que l'ordinateur effectue 10^9 opérations à la seconde.

Complexité	$n = 10$	$n = 100$	$n = 1000$
n			
n^2			
n^3			
$\sqrt{n}$			
2^n			
$\log_2(n)$			

Q 2.3 Indiquer quelle est la plus grande taille d'instance traitable dans le temps donné.

Complexité	1s	1h	1an
n			
n^2			
n^3			
$\sqrt{n}$			
2^n			
$\log_2(n)$			

<u>Exercice 3</u> : Des coefficients bien connus ?

```
def coef (n):
  u = [ 0 for i in range(0,n) ]
  v = [ 0 for i in range(0,n) ]
  u[0] = 1;
  for i in range(1,n):
    v[0] = 1
    for j in range(1,i+1):
      v[j] = u[j-1] + u[j]
    for j in range(0,i+1):
      u[j] = v[j]
  return u
```

Q 3.1 Simuler le fonctionnement de cet algorithme pour $n = 4$. Que calcule cet algorithme ?

Q 3.2 Compter le nombre d'opérations d'affectations réalisées.

Q 3.3 Compter le nombre d'opérations d'additions réalisées.

<u>Exercice 4</u> :

```
"""
  CU: there exists p such that n = 2^p
"""
def exo4 (n):
  r = 0
```

```
  while n != 0:
    n = n // 2
    r = r + 1
  return r
```

Q 4.1 Calculer le nombre d'additions effectuées en fonction de n.

Exercice 5 : Pire et meilleur des cas.

Pour chacune des deux fonctions suivantes, trouver un exemplaire pour le meilleur des cas puis un exemplaire pour le pire des cas. Etudier le nombre d'additions dans les deux cas.

```
def f (t,v):
  """
    Computes the sum of elements of t that are strictly greater than v
    :type t: NumPy array of int
    :type v: int
  """
  s = 0
  n = len(t)
  for e in t:
    if e > v:
      s = s + e
  return s
```

```
def g (t):
  """
    :type t: NumPy array of int
  """
  n = len(t)
  a = t[0]
  b = t[n-1]
  r = Element(0)
  for i in range(n):
    if t[i] >= a and t[i] <= b:
      r = r + t[i]
  return r
```

Exercice 6 : A vous de jouer

On souhaite un algorithme prenant en entrée un tableau de taille n respectant les contraintes suivantes :

- il réalise au plus n additions,
- il réalise au moins $n/2$ additions,
- il utilise un espace mémoire supplémentaire indépendant de n.

Q 6.1 Ecrire un tel algorithme (peu importe ce qu'il calcule).

Q 6.2 Donner un exemplaire pour le meilleur des cas.

Q 6.3 Donner un exemplaire pour le pire des cas.

Exercice 7 : Application des définitions des bornes asymptotiques

Q 7.1 Montrer que :

1. $n^2 + 2n + 3 = \mathcal{O}(n^2)$,
2. $n^2 - 5n + 3 = \Omega(n)$,
3. $n^4 + n^2 = \Theta(n^4)$,
4. n^3 n'appartient pas à $\Theta(n^2)$.

Q 7.2 Donner et montrer un équivalent de la forme n^k des fonctions :

1. $f_1(n) = n^3 - n^2 + 2^{21}$
2. $f_2(n) = \sum_{i=1}^{n} i^2$

Exercice 8 : $\mathcal{O}$ et Θ

Reprendre les fonctions de l'exercice 1 et donner une borne asymptotique de la complexité en nombre d'additions avec la notation $\mathcal{O}$ puis avec la notation Θ.

Exercice 9 : Vrai ou faux

On note respectivement c_m, c_p et c la complexité dans le meilleur des cas, le pire des cas et la complexité en général.

1. si $c_m(n) = \Omega(f(n))$ alors $c(n) = \mathcal{O}(f(n))$
2. si $c_m(n) = \Omega(f(n))$ alors $c(n) = \Omega(f(n))$
3. si $c_m(n) = \mathcal{O}(f(n))$ alors $c(n) = \Omega(f(n))$
4. si $c_p(n) = \mathcal{O}(f(n))$ alors $c(n) = \mathcal{O}(f(n))$
5. si $c_p(n) = \Omega(f(n))$ alors $c(n) = \mathcal{O}(f(n))$
6. si $c_p(n) = \Omega(f(n))$ alors $c(n) = \Omega(f(n))$