1 Equivalents

- Donnez un équivalent simple puis un ordre de grandeur simple pour $f(n) = 4n^2 + 5n + 7$
- Quelle est la complexité du code suivant : `if b then print(bonjour) else faire 100 fois print(bonsoir)`
- (facultatif) Montrez que $\frac{2^n}{n+1} \leq C_n^{n/2} \leq 2^n$. Donnez un équivalent simple de $C_n^{n/2}$ puis de $C_n^{n/2+\lambda\sqrt{n}}$. Rappel, $n! \sim \sqrt{2\pi n}(n/e)^n$,
- Combien vaut $\sum_{i=1}^{n} i$? Donner un équivalent de $\sum_{i=1}^{n} i^2$, de $\sum_{i=1}^{n} i^k$, de $\sum_{i=1}^{n} 1/i$, de $\sum_{i=1}^{n} \ln i$.

2 Tailles des entiers

Soit n est un entier strictement positif s'écrivant avec k chiffres en binaire. Encadrez n en fonction de k. Donnez (un équivalent de) k en fonction de n. Quelle est la taille de n ?

Quel rapport y-a-il entre le nombre de chiffres en base 2 de n et celui de $n\,div\,2$? Combien d'itérations effectue le programme suivant : `i = n ; tant que i est non nul faire i = i div 2 ;` ?

3 Complexités de l'arithmétique

Quelle est la complexité des algos d'addition et de multiplication d'entiers vus au primaire ? Quelle est la complexité des algos utilisant les "bâtons" ? Quels sont les meilleurs algos ?

(facultatif) Déterminer les plus petits entiers m_k et n_k dont la recherche du PGCD nécessite k itérations par l'algorithme d'Euclide. En déduire la complexité en nombre d'itérations de cet algorithme.

4 Complexités de codes pour la fonction racine entière

La racine entière (notation: `RE`) d'un entier n est le plus grand entier p tel que $p^2 \leq n$, par exemple $RE(16) = 4$, $RE(30) = 5$. On propose les codes ci-dessous pour calculer la fonction RE.

Donnez l'ordre de grandeur de la complexité de chacun d'eux en fonction de la VALEUR de l'entier n en argument. Les additions et multiplications d'entiers seront considérées de complexité constantes.

```
+---------------------------+---------------------------+-----------------------------------+
|                           | fonction RE2 (n:int): int | fonction RE3 (n:int): int         |
| fonction RE1 (n:int): int | si n == 0                 | si n == 0                         |
| int cpt = 0               | alors rendre 0            | alors rendre 0                    |
| tant que cpt*cpt <= n     | sinon int y = RE2(n-1)    | sinon                             |
|      faire cpt++          |       si (y+1)*(y+1) == n |   si (RE3(n-1)+1)*(RE3(n-1)+1)==n |
| rendre cpt-1              |       alors rendre y+1    |   alors rendre RE3(n-1)+1         |
|                           |       sinon rendre y      |   sinon rendre RE3(n-1)           |
+---------------------------+-----+---------------------+-----------------------------------+------+
| fonction RE4 (n:int): int       | fonction RE5 (n:int) :int                                      |
| si n == 0                       | int j = 1                                                      |
| alors rendre 0                  | tant que j*j <= n faire j = 2*j                                |
| sinon int y = RE4(n / 4)        | int i = j / 2                                                  |
|       si (2*y+1)*(2*y+1) <= n   | tant que j>i+1 faire si ((i+j)/2)*((i+j)/2) <= n               |
|       alors rendre 2*y+1        |                       alors i = (i+j)/2                        |
|       sinon rendre 2*y          |                       sinon j = (i+j)/2                        |
|                                 | rendre i                                                       |
+---------------------------------+----------------------------------------------------------------+
```

5 Faut-il trier ?

On a deux listes non triées de longueur N et n. On supposera que $N \geq n$. On cherche à savoir s'il y a un élément commun aux deux listes. Donnez 4 stratégies et comparer les ordres de grandeurs de leurs complexités. Après avoir rapidement expliqué comment elles fonctionnent, vous pouvez utiliser les fonctions et procédures suivantes (les complexités sont en nombre de comparaisons d'éléments des listes, l désigne la longueur de la liste L):

- $EstDans(x, L)$ qui teste si l'élément x apparaît dans la liste L, $EstDans(x, L)$ est de complexité $l + O(1)$
- $EstDansTriee(x, L)$ fait de même, mais en supposant la liste L triée. Sa complexité est $\ln_2 l + O(1)$
- $ElementCommunDansTriees(L1, L2)$ suppose que $L1$ et $L2$ sont triées et regarde si elles ont un élément commun. Cette fonction est de complexité $l_1 + l_2 + O(1)$.
- $Tri(L)$ qui trie la liste L. Cette fonction est de complexité $\sim l \ln_2 l$.

6 Complexité amortie

(1) Sur une année, vous devez payer un euro par jour sauf le 31 décembre où vous devez payer 366 euros. Quel est le coût quotidien au mieux, au pire ?

Si vous gagnez une somme fixe par jour, combien faut-il gagner pour vous en sortir ?

(2) Si la valeur de cpt est comprise entre 0 et N, quelles sont les complexité au mieux et au pire de l'opération "`cpt++`" en nombre de bits à modifier ? (les nombres sont écrits en base 2).

(3) Dans le programme suivant : "`cpt = 0 ; pour i de 1 à N faire cpt++`", quel est le coût amorti de `cpt++` ?

7 Lourdeurs

Réécrire le code suivant de manière plus élégante :

```
Si p != 0 alors si (b == faux)
                alors si q != 0 alors rendre vrai
                                sinon rendre faux
                sinon rendre (q==1)
          sinon rendre (b == vrai)
```

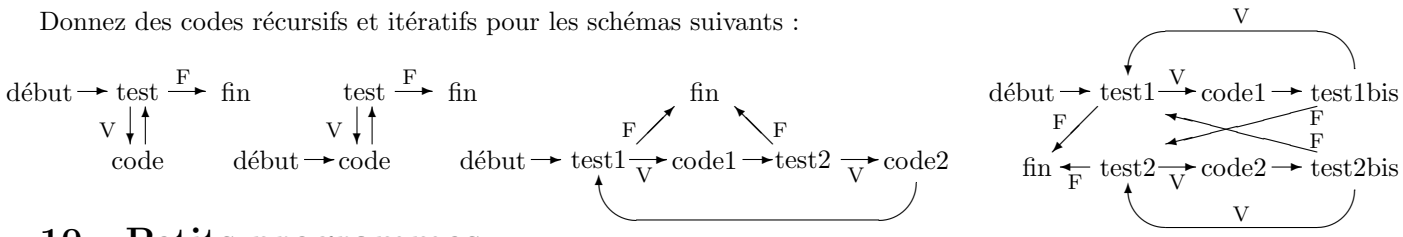
8 Logique et codage

```
pour tout a dans A
    si     Q(a)
rendre
```

Complétez le code ci-contre pour rendre vrai ssi $\forall a \in A, Q(a)$.
Complétez le code ci-contre pour rendre vrai ssi $\exists a \in A, Q(a)$.
Donnez un code qui rend vrai ssi $\forall a \in A, \exists b \in B, \forall c \in C, P(a,b,c)$.

9 Boucles

Donnez des codes récursifs et itératifs pour les schémas suivants :



10 Petits programmes

- le log entier d'un entier strictement positif n est le nombre de fois qu'il faut le diviser par 2 pour obtenir 1. Donnez un code itératif, un code récursif et un code récursif terminal qui calcule le log entier.
  Que pensez-vous des trois codes ci-dessous ?

```
-------------------------------------------------------------------------------------------
|                            | LogEntier (in int n) : int    | LogEntier (in int n) : int    |
| LogEntier (in int n) : int |     int cpt                   |   int cpt = 0                 |
| int cpt = 0                |     LE(n,&cpt)                |   LE(n,cpt)                   |
| si n == 1 alors rendre cpt |     rendre cpt                |   rendre cpt                  |
| sinon cpt++                | LE (in int n, inout int *cpt) |                               |
|       LogEntier(n / 2)     | *cpt = 0                      | LE (in int n, in int cpt)     |
|       rendre cpt           | si n > 1 alors (*cpt)++       | si n > 1 alors cpt++          |
|                            |                LE(n / 2, cpt) |                LE(n / 2, cpt) |
-------------------------------------------------------------------------------------------
```

- Donner des pseudo-codes pour la fonction `power(x,n)` qui prend en entrée x réel et n entier (supposé positif ou nul) et rend x^n: En se basant sur le principe que $x^n = x * x^{n-1}$, (1) en récursif simple, (2) En itératif, (3) en récursif terminal. (4) En se basant sur le principe que $x^p = (x^{p/2})^2$,
- Le jeu de Hanoï: On a 3 piquets. En position initiale, on a n disques sur le premier piquet, chacun reposant sur un disque de taille plus grande ; sur les second et troisième piquets, il n'y a pas de disques. À chaque étape, on a le droit de déplacer un disque d'un piquet à l'autre, à condition qu'un disque ne soit jamais posé sur un disque plus petit. Le but du jeu est de déplacer les n disques du piquet 1 au piquet 3.

  Montrez qu'il y a une solution pour tout n. Puis crire une procédure qui prend n en argument et qui affiche la liste des opérations à effectuer pour résoudre le problème des tours de Hanoï à n disques.

11 La fonction d'Ackermann

Les fonctions d'Ackermann sont définies comme suit : $A_0(n) = n+1$ puis :
$A_{m+1}(n) = A_m(A_m(A_m....(A_m(1))...))$ avec $n+1$ appels composés de A_m.

- Donnez un pseudo-code de Ackermann(m,n) $= A_m(n)$ avec de l'itératif et du récursif.
- Donnez un pseudo-code de Ackermann purement récursif.
- Donnez les premières valeurs de la suite $(A_m(0))_{m \in N}$

12 Les pièces

Que fait la fonction suivante ? :

```
function f (n : integer) : integer ;     (* n >= 0 ; n est une somme en francs *)
var     i,j,k,l,m,  sum  : integer ;
begin   sum := 0 ;
        for i := 0 to n        do
         for j := 0 to n div 2  do
          for k := 0 to n div 5  do
           for l := 0 to n div 10 do
            for m := 0 to n div 20 do
                if i + 2*j + 5*k + 10*l + 20*m = n then sum := sum +1 ;
        f := sum
end ;
```

Donnez un équivalent de sa complexité en nombre de tests. Peut-on améliorer cette complexité ?