INF7440 : Devoir #1

(À remettre le 4 octobre (au début du cours))

a. $a^{i+j} = a^i \times a^j$

b. $\lg\ n^k = k \times \lg\ n$

c. $\lg\ (n_1 \times n_2) = \lg\ n_1 + \lg\ n_2$

d. $log_y\ n = \dfrac{log_x\ n}{log_x\ y}$

e. $\lg\ n \leq n$, si $n \geq 1$ et $a \geq 2$

f. $x^{-y} = \dfrac{1}{x^y}$

g. $\sqrt[x]{y} = y^{\frac{1}{x}}$

Indice 1: Rappels utiles sur les logarithmes et les exposants

1. Définitions de O, Ω et Θ (10 pts)

Pour chacune des fonctions indiquées plus bas, donnez un estimé O ou Θ (selon ce qui est demandé) le plus précis possible. Dans chaque cas, *justifiez* votre réponse en référant de façon détaillée à la définition de O ou Θ et à la fonction f_i — en d'autres mots, identifiez les constantes N et c (ou N et c_1 et c_2) telles que la définition appropriée (O ou Θ) s'applique pour la fonction f_i indiquée.

$$\begin{array}{lcll} f_1(n) & = & \lg\ (3n^5 \times 4^{n+1}) + \lg\ 2n & \in O(?) \\ f_2(n) & = & 2^{-3n} + 6n^3 + 4n & \in O(?) \\ f_3(n) & = & 3^{n+2} + 2n^4 + 6n & \in \Theta(?) \end{array}$$

Pour appliquer les définitions, simplifiez les expressions et utilisez "$\leq$" (ou $\geq$). Notez que lorsque vous établissez des relations entre diverses expressions, il faut indiquer, à chaque étape, la relation la plus précise qui s'applique. Par exemple, pour $\lg\ (n^2 + 1)$, on a les relations (étapes) suivantes :

$$\begin{array}{rcll} \lg\ (n^2+1) & \leq & \lg(n^2+n^2) & (\text{pour } n \geq 1) \\ & = & \lg(2n^2) & \\ & = & \lg\ 2 + \lg\ n^2 & \\ & = & \lg\ 2 + 2\ \lg\ n & \\ & \leq & \lg\ n + 2\ \lg\ n & (\text{pour } n \geq 2) \\ & = & 3\ \lg\ n & \end{array}$$

D'où l'on conclut que $\lg\ (n^2+1) \in O(\lg\ n)$ (pour $N = 2$ et $c = 3$).

Indice 2: Indice pour appliquer les définitions de O, Ω, Θ

2. Analyse d'un algorithme itératif (15 pts)

Soit l'algorithme suivant (n'essayez pas de le comprendre ;) :

```
PROCEDURE proc( n: Nat ): Nat
DEBUT
   r ← 1
   POUR i ← 1 A n FAIRE
     POUR j ← i A 2*n FAIRE
       r ← 2 * r
       s ← 0
       POUR k ← 1 A n FAIRE
         s ← s + r
       FIN
     FIN
   FIN
   RETOURNER( s )
FIN
```

a. Considérons les opérations d'affectation (←) comme étant les opérations élémentaires pour cet algorithme — *en incluant aussi* les affectations (implicites) aux diverses variables de contrôle des boucles `POUR`. Déterminez alors le *nombre exact* d'opérations élémentaires effectuées par cet algorithme.

b. Simplifiez l'expression obtenue en a. de façon à déterminer la complexité temporelle de l'algorithme. Justifiez brièvement votre raisonnement, et ce en indiquant, à chaque étape, les propriétés de Θ qui s'appliquent (section 1.5 du manuel).

c. Identifiez une *opération barométrique* appropriée pour cet algorithme puis trouvez un estimé Θ pour le nombre de fois où cette opération est exécutée par l'algorithme.

   Justifiez brièvement *le choix* de cette opération barométrique, c'est-à-dire expliquez pourquoi il s'agit bien d'une opération *barométrique* appropriée pour cet algorithme.

Note : Si nécessaire, pour simplifier l'analyse, vous pouvez supposer que n *est d'une forme appropriée.*

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

Indice 3: Rappels utiles sur les sommations

3. Application du théorème général (15 pts)

Utilisez l'une des variantes de la méthode générale (Neapolitan & Naimipour, Cormen *et al.*) pour obtenir une solution asymptotique la plus précise possible pour chacune des récurrences suivantes — dans tous les cas, on suppose que $T(1) \in \Theta(1)$ et que n est une puissance appropriée de b :

a. $T(n) \leq 16T(n/2) + n^4$

b. $T(n) = 3T(n/3) + n^2$

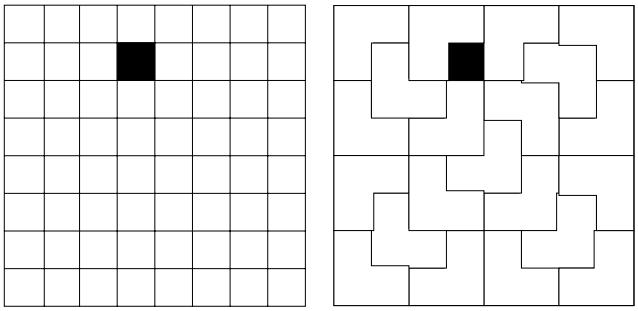
c. $T(n) = 2T(n/4) + \sqrt[3]{n}$

d. $T(n) \geq 9T(n/2) + n^3$

e. $T(n) = 3T(n/5) + n^2 \lg n$

4. Algorithme diviser-pour-régner (20 pts)

Supposons que l'on ait un échiquier de taille $n \times n$ — avec n une puissance de 2 — où l'une des cases a été «retirée» — par exemple, on a fait un trou : voir Figure 1 (a).
Un *tromino* est une pièce «en équerre» (en L) pouvant couvrir trois cases de l'échiquier.
Soit alors un ensemble de trominos, numérotés de 1 à $\frac{n^2-1}{3}$. On veut utiliser ces trominos pour couvrir l'ensemble de l'échiquier, mais en laissant libre toutefois la case retirée (trou), tel qu'illustré à la Figure 1 (b).



(a) Échiquier initial (b) Échiquier couvert

```
{Devoir1@linux} trominos 8 2 4 1
a::
  1   1   2   2   6   6   7   7
  1   5   2   0   6  10  10   7
  3   5   5   4   8   8  10   9
  3   3   4   4  21   8   9   9
 11  11  12  21  21  16  17  17
 11  15  12  12  16  16  20  17
 13  15  15  14  18  20  20  19
 13  13  14  14  18  18  19  19
```

(c) Résultat du programme

Figure 1: Un échiquier 8×8 avec un trou, puis couvert avec des trominos.

a. Concevez un algorithme *récursif*, basé sur la stratégie diviser-pour-régner, permettant de résoudre ce problème — pour un petit coup de pouce sur comment procéder, voir l'indice 4.

   Votre algorithme doit être écrit dans le langage MPD. **Plus précisément** : vous devez *compléter* le squelette de programme fourni sur le site *Web* du cours en définissant la procédure `couvrirDeTrominos` — vous pouvez évidemment introduire des procédures auxiliaires.

   Un exemple de résultat produit par le programme (lorsque l'impression est activée : dernier argument du programme supérieur ou égal à 1) est présenté à la Figure 1 (c), où la valeur 0 indique la présence du «trou» (2ième ligne, 4ième colonne).

   Vous devrez compiler et exécuter sur divers tests (sur `arabica` ou sur tout autre machine Unix ou Linux) le programme résultant. Pour ce faire, un fichier `Makefile` vous est fourni : voir le commentaire au début de ce fichier qui explique les différents appels possibles à «`make`».

Vous devrez remettre votre programme final (fichier `trominos.mpd`) en exécutant la commande suivante sur la machine `arabica` :

`oto rendre_tp tremblay INF7440` <*codePerm*> `trominos.mpd`

Note : Dans cette commande, <*codePerm*> est soit votre code permanent si vous travaillez seul, soit les codes permanents des deux coéquipiers séparés par une «,» (sans espaces blancs) si vous travaillez avec quelqu'un d'autre, par exemple :

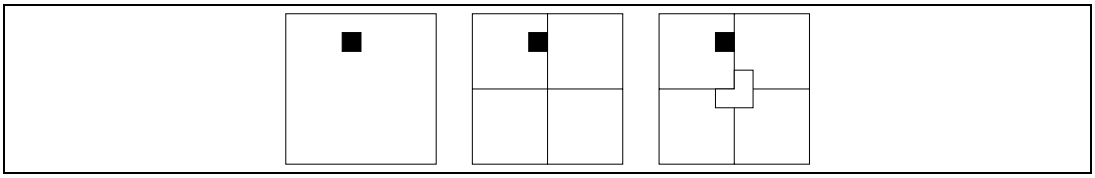
```
oto rendre_tp tremblay INF7440 TREG04127602 trominos.mpd
oto rendre_tp tremblay INF7440 TREG04127602,DAVA26628003 trominos.mpd
```

Vous pouvez aussi remettre votre programme en utilisant l'interface *Web* pour l'outil Oto : `http://labunix.uqam.ca:8181/~oto/application-web`.

Note : L'outil de correction `oto` sera utilisé pour vérifier le bon fonctionnement de votre programme. Les tests utilisés pour la correction comprendront ceux du `Makefile` ainsi que des tests additionnels.

b. Donnez les équations de récurrence décrivant le temps d'exécution de votre algorithme.

c. Déterminez la complexité asymptotique de votre algorithme (complexité Θ) en trouvant la solution des équations de récurrence indiquées en b.



Indice 4: Indice pour la stratégie diviser-pour-régner.