Tutorat #1

Exercice 1

Ecrire six algorithmes différents pour déterminer si un nombre entier est premier ou composé.

Évaluer la complexité pour chacun des algorithmes proposés.

Exercice 2

1. Calculer les ordres de grandeurs suivantes en secondes :
   a. 1 heure
   b. 1 jour
   c. 1 semaine
   d. 1 mois
   e. 1 année
   f. 1 siècle
   g. 1 millénaire
2. Considérons 7 algorithmes A0, A1, …, A6 conçus pour résoudre un même problème de taille n. La complexité en temps de chacun de ces algorithmes est exprimée dans la table ci-dessous.

Algorithme	A0	A1	A2	A3	A4	A5	A6
Complexité temporelle	$O(1)$	$O(\log_2(n))$	$O(\sqrt{n})$	$O(n)$	$O(n^2)$	$O(n^3)$	$O(2^n)$

   a. En supposant qu'une unité de taille s'exécute en une milliseconde, calculer le temps nécessaire pour traiter des tailles respectivement égales à 10, 100 et 1000.
   b. Répéter la question 2 avec une unité de temps égale à une microseconde.
   c. Que peut-on en conclure?

Exercice 3

Considérons le théorème suivant sur le pgcd ou le plus grand commun diviseur :

$$pgcd(a, b) = pgcd(b, a \bmod b)$$

1. Utiliser le théorème pour écrire un algorithme de calcul itératif du pgcd de deux entiers a et b.
2. Calculer la complexité en temps de l'algorithme.
3. Écrire la version récursive de l'algorithme trouvé en 1. et calculer sa complexité.
4. Écrire un algorithme pour calculer le ppcm de deux nombres a et b et calculer sa complexité.

Pour rappel, le PGCD est le **P**lus **G**rand **C**ommun **D**iviseur et le PPCM est le **P**lus **P**etit **C**ommun **M**ultiple.

Exercice 4

1. Considérons l'alphabet {0,1}, un palindrome est un mot qui se présente sous le format ww^{-1} où w est un mot quelconque de l'alphabet et w^{-1} est l'image miroir de w.
   Ecrire un algorithme qui reconnaît des palindromes et calculer sa complexité.
2. Considérons l'alphabet {0,1}, et le langage L = {(0|1)*, nb de 0 = nb de 1}.
   Ecrire un algorithme qui reconnaît L et calculer sa complexité.

Exercice 5

Considérons un tableau constitué des n premiers nombres entiers (1, 2, …, n). Une méthode de détermination des nombres premiers inférieurs au sens large à n, consiste à considérer le nombre 2 qui est premier puis à éliminer tous les nombres multiples de 2 car ils ne sont pas premiers, ensuite itérer ce processus en continuant avec le nombre suivant non éliminé c'est-à-dire 3 jusqu'à traiter tous les entiers du tableau.

1. Illustrer chaque itération du procédé de calcul des nombres premiers décrit ci-dessus sur les 10 premiers nombres entiers.
2. Ecrire un algorithme de calcul et d'impression des nombres premiers inférieurs au sens large à n. Il est recommandé par souci de simplification de l'algorithme de mettre le nombre à 0 s'il est premier et à 1 sinon, une fois qu'il est traité.
3. Calculer la complexité de l'algorithme.

Exercice 6

Déterminer l'ordre de complexité des algorithmes ci-dessous.

```java
public class Exercise1_6 {

    /** Returns the sum of the integers in given array. */
    public static int example1(int[] arr) {
        int n = arr.length, total = 0;
        for (int j = 0; j < n; j++) { // loop from 0 to n-1
            total += arr[j];
        }
        return total;
    }

    /** Returns the sum of the integers with even index in given array. */
    public static int example2(int[] arr) { // note the increment of 2
        int n = arr.length, total = 0;
        for (int j = 0; j < n; j += 2) {
            total += arr[j];
        }
        return total;
    }

    /** Returns the sum of the prefix sums of given array. */
    public static int example3(int[] arr) {
        int n = arr.length, total = 0;
        for (int j = 0; j < n; j++) { // loop from 0 to n-1
            for (int k = 0; k <= j; k++) { // loop from 0 to j
                total += arr[j];
            }
        }
        return total;
    }

    /** Returns the sum of the prefix sums of given array. */
    public static int example4(int[] arr) {
        int n = arr.length, prefix = 0, total = 0;
        for (int j = 0; j < n; j++) { // loop from 0 to n-1
            prefix += arr[j];
            total += prefix;
        }
        return total;
    }

    /**
     * Returns the number of times second array stores sum of prefix sums from
     * first.
     */
    public static int example5(int[] first, int[] second) { // assume equal-length arrays
        int n = first.length, count = 0;
        for (int i = 0; i < n; i++) { // loop from 0 to n-1
            int total = 0;
            for (int j = 0; j < n; j++) { // loop from 0 to n-1
                for (int k = 0; k <= j; k++) { // loop from 0 to j
                    total += first[k];
                }
            }
            if (second[i] == total) {
                count++;
            }
        }
        return count;
    }
}
```

Exercice 7

Déterminer l’ordre de complexité des algorithmes ci-dessous.

```
% Algorithme 1 : Addition de matrices
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        a[i][j] = b[i][j] + c[i][j];

% Algorithme 2 : Multiplication de matrices
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        for (k = a[i][j] = 0; k < n; k++)
            a[i][j] += b[i][k] * c[k][j];

% Algorithme 3 : Transposition de matrices
for (i = 0; i < n - 1; i++)
    for (j = i+1; j < n; j++)
        tmp = a[i][j];
        a[i][j] = a[j][i];
        a[j][i] = tmp;

% Algorithme 4
for (cnt1 = 0, i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
        cnt1++;

% Algorithme 5
for (cnt2 = 0, i = 1; i <= n; i++)
    for (j = 1; j <= i; j++)
        cnt2++;

% Algorithme 6
for (cnt3 = 0, i = 1; i <= n; i *= 2)
    for (j = 1; j <= n; j++)
        cnt3++;

% Algorithme 7
for (cnt4 = 0, i = 1; i <= n; i *= 2)
    for (j = 1; j <= i; j++)
        cnt4++;
```