Algorithmes et Pensée Computationnelle

Algorithmes de tri et Complexité - exercices basiques

Le but de cette série d'exercices est d'aborder les notions présentées durant la séance de cours. Cette série d'exercices sera orientée autour des points suivants :

1. la récursivité,
2. la complexité des algorithmes et,
3. les algorithmes de tri

Les langages de programmation qui seront utilisés pour cette série d'exercices sont Java et Python.
Le temps indiqué (🕓) est à titre indicatif.

1 Récursivité (10 minutes)

Le but principal de la récursivité est de résoudre un gros problème en le divisant en plusieurs petites parties à résoudre.

Question 1: (🕓 *10 minutes*) **Somme des chiffres**

Écrivez un algorithme récursif en Python ou en Java qui prend un nombre et retourne la somme des chiffres dont il est composé. Par exemple, la somme des chiffres de 126 est : 1+2+6 = 9.

> **Conseil**
>
> Pour obtenir les chiffres qui composent un nombre, utilisez l'opérateur **%** (modulo - `https://fr.wikipedia.org/wiki/Modulo_(op%C3%A9ration)`).
> Pour obtenir le nombre 12 à partir du nombre 126, il vous suffit de faire la division entière par 10. En Python, on utilise l'opérateur **//** : 126 // 10 = 12. En Java, la division entre deux variables de type **int** est entière, et vous n'aurez ainsi qu'à utiliser l'opérateur de division normal **/** : 126 / 10 = 12.

2 Complexité (40 minutes)

Indiquez en une phrase, ce que font ces algorithmes ci-dessous et calculez leur complexité temporelle avec la notation $O()$. Le code est écrit en Python et en Java.

Question 2: (🕓 *10 minutes*) **Complexité**

Quelle est la complexité du programme ci-dessous ?

Python :

```python
# Entrée: n un nombre entier
def algo(n):
    s = 0
    for i in range(10*n):
        s += i
    return s
```

Java :

```java
public static int algo(int n) {
    int s = 0;
    for (int i=0; i < 10*n; i++){
        s += i;
    }
    return s;
}
```

1. $O(n)$
2. $O(n^3)$
3. $O(\log(n))$

4. $O(n^n)$

> **Conseil**
>
> Rappelez-vous que la notation $O()$ sert à exprimer la complexité d'algorithmes dans le **pire des cas**. Les règles suivantes vous seront utiles. Pour n étant la taille de vos données, on a que :
>
> 1. Les constantes sont ignorées : $O(2n) = 2 * O(n) = O(n)$
> 2. Les termes dominés sont ignorés : $O(2n^2 + 5n + 50) = O(n^2)$

Question 3: (*10 minutes*) **Complexité**
Quelle est la complexité du programme ci-dessous ?
Python :

```python
# Entrée: L est une liste de nombres entiers et M un nombre entier
def algo(L, M):
    i = 0
    while i < len(L) and L[i] <= M:
        i += 1
    s = i - 1
    return s

```

Java :

```java
public static int algo(int[] L, int M) {
    int i = 0;
    while (i < L.length && L[i] <= M){
        i += 1;
    }
    int s = i - 1;
    return s;
}

```

1. $O(n^3)$
2. $O(\log(n))$
3. $O(n)$
4. $O(n^n)$

Question 4: (*10 minutes*) **Complexité**
Quelle est la complexité du programme ci-dessous ?
Python :

```python
#Entrée: L et M sont 2 listes de nombres entiers
def algo(L, M):
    n = len(L)
    m = len(M)
    for i in range(n):
        L[i] = L[i]*2
    for j in range(m):
        M[j] = M[j]%2

```

Java :

```java
public static void algo(int[] L, int[] M) {
    int n = L.length;
    int m = M.length;
    for (int i=0; i < n; i++){
        L[i] = L[i]*2;
    }
    for (int j=0; j < m; j++){
        M[j] = M[j]%2;
    }
}

```

1. $O(n^2)$
2. $O(n+m)$
3. $O(n)$
4. $O(2^n)$

Question 5: (🕓 *10 minutes*) **Complexité**
Quelle est la complexité du programme ci-dessous ?
Python :

```python
#Entrée: L est une liste de nombre entiers
def algo(L):
    n = len(L)
    i = 0
    s = 0
    while i < math.log(n):
        s += L[i]
        i += 1
    return s

```

Java :

```java
import java.lang.Math;

public static void algo(int[] L) {
    int n = L.length;
    int s = 0;
    for (int i=0; i < Math.log(n); i++){
        s += L[i];
    }
}
```

1. $O(n^2)$
2. $O(n)$
3. $O(\log(n))$
4. $O(n^n)$

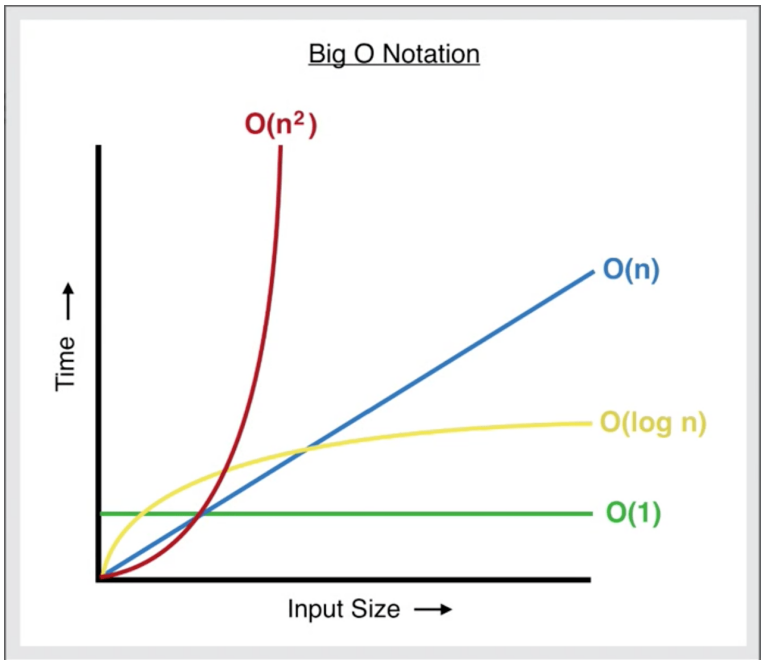

FIGURE 1 – Représentation de complexités temporelles

Question 6: (🕒 *10 minutes*) **Complexité** Optionnel
Quelle est la complexité du programme ci-dessous ?
Python :

```python
# Entrée: n un nombre entier
def algo(n):
    m = 0
    for i in range(n):
        for j in range(i):
            m += i+j
    return m

```

Java :

```java
public static int algo(int n) {
    int m = 0;
    for (int i=0; i < n; i++){
        for (int j=0; j < i; j++){
            m += i+j;
        }
    }
    return m;
}

```

1. $O(n^2)$
2. $O(n)$
3. $O(\log(n))$
4. $O(2^n)$

3 Algorithmes de Tri (45 minutes)

Question 7: (🕒 *15 minutes*) **Insertion dans une liste triée (Python)**
Soit un nombre entier `n`, et une liste triée `l`. Ecrivez un programme Python qui insère la valeur `n` dans la liste `l` tout en s'assurant que la liste `l` reste triée.

```python
def insertion_entier(liste, number):
    # TODO : Compléter ici

print(insertion_entier([2, 4, 6], 1))
```

Exemple

En passant les arguments suivants à votre programme : n=5 et l=[2,4,6]. Ce dernier devra retourner **l =[2,4,5,6]**

Question 8: (🕒 *30 minutes*) **Tri fusion (Merge Sort) - Python**
À partir de deux listes triées, on peut facilement construire une liste triée comportant les éléments issus de ces deux listes (leur *fusion*). Le principe de l'algorithme de tri fusion repose sur cette observation : le plus petit élément de la liste à construire est soit le plus petit élément de la première liste, soit le plus petit élément de la deuxième liste. Ainsi, on peut construire la liste élément par élément en retirant tantôt le premier élément de la première liste, tantôt le premier élément de la deuxième liste (en fait, le plus petit des deux, à supposer qu'aucune des deux listes ne soit vide, sinon la réponse est immédiate).
Les étapes à suivre pour implémenter l'algorithme sont les suivantes :

1. Si le tableau n'a qu'un élément, il est déjà trié.
2. Sinon, séparer le tableau en deux parties plus ou moins égales.
3. Trier récursivement les deux parties avec l'algorithme de tri fusion.
4. Fusionner les deux tableaux triés en un seul tableau trié.

Soit la liste `l` suivante [38, 27, 43, 3, 9, 82, 10], triez les éléments de la liste en utilisant un tri fusion. Combien d'itération effectuez-vous ?

— **Python :**

```
1  def merge(partie_gauche, partie_droite):
2      # TODO: Code à compléter
3
4  def tri_fusion(l):
5      # TODO: Code à compléter
6
7  if __name__ == "__main__":
8      l = [38, 27, 43, 3, 9, 82, 10]
9      print(tri_fusion(l))
```

Conseil

— L'algorithme est récursif.
— Revenez à la visualisation de l'algorithme dans les diapositives 80 à 108 pour comprendre comment marche concrètement le tri fusion.

Question 9: (*20 minutes*) **Tri à bulles (Bubble Sort) - Python** **Optionnel**

Le tri à bulles consiste à parcourir une liste et à comparer ses éléments. Le tri est effectué en permutant les éléments de telle sorte que les éléments les plus grands soient placés à la fin de la liste.
Concrètement, si un premier nombre x est plus grand qu'un deuxième nombre y et que l'on souhaite trier l'ensemble par ordre croissant, alors x et y sont mal placés et il faut les inverser. Si, au contraire, x est plus petit que y, alors on ne fait rien et l'on compare y à z, l'élément suivant.
Soit la liste `l = [1, 2, 4, 3, 1]`, triez les éléments de la liste en utilisant un tri à bulles. Combien d'itérations effectuez-vous ?

— **Python :**

```
1  def tri_bulle(l):
2      for i in range(len(l)):
3          #TODO: Code à compléter
4
5  if __name__ == "__main__":
6      l = [1, 2, 4, 3, 1]
7      tri_bulle(l)
8      print(l)
```