Algorithmes et Pensée Computationnelle

Algorithmes de tri et Complexité

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

Les languages de programmation qui seront utilisés pour cette série d'exercices sont Java et Python.

1 Complexité (30 minutes)

Pour chacun des programmes ci-dessous, donnés à chaque fois en Python et Java, indiquez en une phrase, ce que font ces algorithmes et calculez leur complexité temporelle avec la notation $O()$. Le code est écrit en Python et en Java.

Question 1: (*10 minutes*) **Complexité**

Quelle est la complexité du programme ci-dessous ?
Python :

```python
# Entrée: n un nombre entier
def algo1(n):
    s = 0
    for i in range(10*n):
        s += i
    return s

```

Java :

```java
public static int algo1(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 2: (*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 algo2(L, M):
    i = 0
    while i < len(L) and L[i] <= M:
        i += 1
    s = i - 1
    return s
```

Java :

```java
public static int algo2(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 3: (*10 minutes*) **Complexité**

Quelle est la complexité du programme ci-dessous ?

Python :

```python
#Entrée: L et M sont 2 listes de nombre entiers
def algo3(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 algo3(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 4: (*10 minutes*) **Complexité**

Quelle est la complexité du programme ci-dessous ?

Python :

```python
#Entrée: L est une liste de nombre entiers
def algo4(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 algo4(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. $O(n^n)$

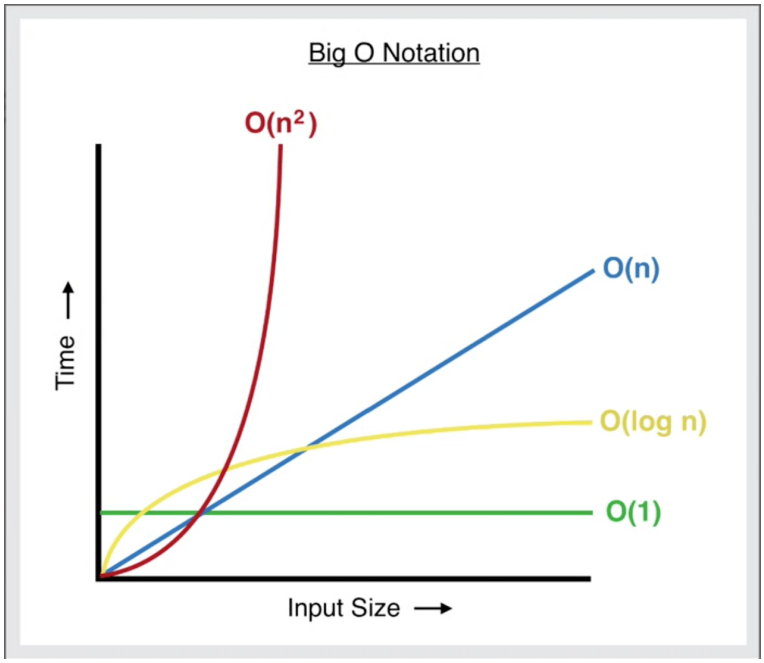


FIGURE 1 – Représentation de complexités temporelles

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

```python
# Entrée: n un nombre entier
def algo5(n):
    m = 0
    for i in range(n):
        for j in range(i):
            m += i+j
    return m
```

Java :

```java
public static int algo5(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. $O(2^n)$

2 Récursivité (15 minutes)

Le but principal de la récursivité est de résoudre un gros problème en le divisant en plusieurs petites parties à résoudre.
Question 6: (*5 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

Question 7: (*10 minutes*) **Fibonacci**
La suite de Fibonacci est définie récursivement par les propriétés suivantes :
— si n est égal à 0 ou 1 : fibo(0) = fibo(1) = 1
— si n est supérieur ou égal à 2, alors ; fibo (n) = fibo(n - 1) + fibo(n - 2)
Voici son implémentation en Java :

```java
public static int fibonacci(int n) {
    if(n == 0 | n == 1){
        return n;
    } else{
        return fibonacci(n-1) + fibonacci(n-2);
    }
}
```

Quel est la complexité de l'algorithme ci-dessus ?

Conseil

Aidez-vous d'un exemple (**fibonacci(3)**, **fibonacci(4)**,...)
Pour formaliser la formule de complexité, on peut poser que $T(n)$ énumère le nombre d'opérations requises pour calculer **fibonacci(n)**. Ainsi, $T(n) = T(n-1) + T(n-2) + c$, c étant une constante. Vous pouvez alors énumérer le nombre d'opérations pour **fibonacci(3)**, **fibonacci(4)**... et esssayer de trouver la complexité en terme de $O()$.

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

3 Algorithmes de Tri (60 minutes)

Question 8: (*20 minutes*) **Tri à bulles (Bubble Sort)**
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 :**

```python
def tri_bulle(l):
    for i in range(len(l)):
        #TODO: Code à compléter

if __name__ == "__main__":
    l = [1, 2, 4, 3, 1]
    tri_bulle(l)
    print(liste_triee)
```

— **Java :**

```java
public class question8 {
    public static void tri_bulle(int[] l) {
        for (int i = 0; i < l.length - 1; i++){
            //TODO: Code à compléter
        }
    }

    public static void printArray(int l[]){
        int n = l.length;
        for (int i = 0; i < n; ++i)
            System.out.print(l[i] + " ");

        System.out.println();
    }


    public static void main(String[] args){
        int[] l = {1, 2, 4, 3, 1};
        tri_bulle(l);
        printArray(l);
    }
}
```

Conseil

En Java, utilisez une variable temporaire que vous nommerez **temp** afin de faire l'échange de valeur entre deux éléments de la liste.

Question 9: (*10 minutes*) **Tri par insertion - 1 (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.

```
1 def insertion_entier(liste, number):
2     # TODO : Compléter ici
3
4 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 10: (*20 minutes*) **Tri par insertion - 2 (Insertion Sort)**

Dans l'algorithme de tri par insertion, on parcourt le tableau à trier du début à la fin. Au moment où on considère le i-ème élément, les éléments qui le précèdent sont déjà triés. Pour faire l'analogie avec l'exemple du jeu de cartes, lorsqu'on est à la i-ème étape du parcours, le i-ème élément est la carte saisie, les éléments précédents sont la main triée et les éléments suivants correspondent aux cartes encore en désordre sur la table.
L'objectif d'une étape est d'insérer le i-ème élément à sa place parmi ceux qui le précède. Il faut pour cela trouver où l'élément doit être inséré en le comparant aux autres, puis décaler les éléments afin de pouvoir effectuer l'insertion. En pratique, ces deux actions sont fréquemment effectuées en une passe, qui consiste à faire "remonter" l'élément au fur et à mesure jusqu'à rencontrer un élément plus petit.
Compléter le code suivant pour trier la liste **l** définie ci-dessous en utilisant un tri par insertion. Combien d'itérations effectuez-vous ?

— **Python :**

```
1 def tri_insertion(l):
2     for i in range(1, len(l)):
3         #TODO: Code à compléter
4
5 if __name__ == "__main__":
6     l = [2, 43, 1, 3, 43]
7     tri_insertion(l)
8     print(l)
```

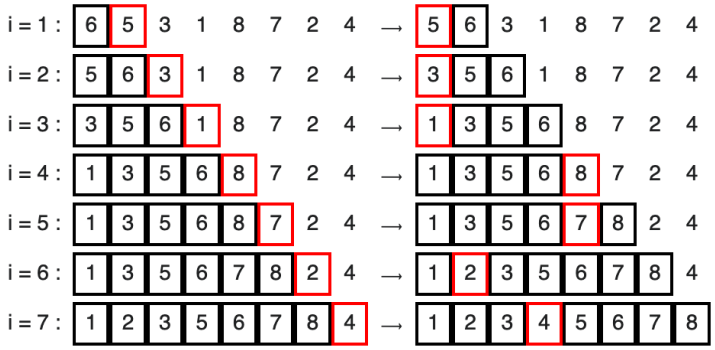
— **Java :**

```
1  public class question10 {
2      public static void tri_insertion(int[] l) {
3          for (int i = 1; i < l.length; i++){
4              //TODO: Code à compléter
5          }
6      }
7
8      public static void printArray(int l[]){
9          int n = l.length;
10         for (int i = 0; i < n; ++i)
11             System.out.print(l[i] + " ");
12         System.out.println();
13     }
14
15
16     public static void main(String[] args){
17         int[] l = {2, 43, 1, 3, 43};
18         tri_insertion(l);
```

```
        printArray(l);
    }
}
```

Conseil

Référez vous à la figure du dessous pour un exemple de tri par insertion.
Référez vous aussi aux diapositives 18 à 72 du cours.



Question 11: (*30 minutes*) **Tri fusion (Merge Sort)**

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 :**

```python
def merge(partie_gauche, partie_droite):
    # TODO: Code à compléter

def tri_fusion(l):
    # TODO: Code à compléter

if __name__ == "__main__":
    l = [38, 27, 43, 3, 9, 82, 10]
    print(tri_fusion(l))
```

— **Java :**

```java
public class question11 {
    // Fusionne 2 sous-listes de arr[].
    // Première sous-liste est arr[l..m]
    // Deuxième sous-liste est arr[m+1..r]
    public static void merge(int arr[], int l, int m, int r) {
        // TODO: Code à compléter
    }

    // Fonction principale qui trie arr[l..r] en utilisant
    // merge()
    public static void tri_fusion(int arr[], int l, int r){
        // TODO: Code à compléter
```

```
13      }
14
15      public static void printArray(int l[]){
16          int n = l.length;
17          for (int i = 0; i < n; ++i)
18              System.out.print(l[i] + " ");
19
20          System.out.println();
21      }
22
23
24      public static void main(String[] args){
25          int[] l = {38, 27, 43, 3, 9, 82, 10};
26          tri_fusion(l, 0, l.length - 1);
27          printArray(l);
28      }
29  }
```

Conseil

— L'algorithme est récursif.
— Revenez à la visualisation de l'algorithme dans les diapositives 83 à 111 pour comprendre comment marche concrètement le tri fusion.