Algorithmes et Pensée Computationnelle

Algorithmes de tri et Complexité - exercices avancés

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 Récursivité

Question 1: (🕓 *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

Solution

Python :

```python
def sum_digits(number):
    if number == 0:
        return 0
    else:
        return (number % 10) + sum_digits(number // 10)

print(sum_digits(126))
```

Java :

```java
public class question6 {
    public static int sum_digits(int number) {
        if(number == 0){
            return 0;
        } else{
            return (number%10) + sum_digits(number/10);
        }
    }

    public static void main(String[] args){
        System.out.println(sum_digits(126));
    }
}
```

2 Algorithmes de Tri

Question 2: (🕓 *20 minutes*) **Tri à bulles (Bubble Sort) en java**
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 ?

— **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.

Solution

Java :

```java
public class question8 {
    public static void tri_bulle(int[] l) {
        int n = l.length;
        for (int i = 0; i < n - 1; i++){
            for (int j = 0; j < n-i-1; j++) {
                if (l[j] > l[j+1]) {
                    // échange l[j+1] et l[i]
                    int temp = l[j];
                    l[j] = l[j+1];
                    l[j+1] = temp;
                }
        }
    }

    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);
    }
}
```

L'algorithme a une complexité de $O(n^2)$ car il contient deux boucles qui parcourent la liste.

Question 3: (*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 :**

```python
def tri_insertion(l):
    for i in range(1, len(l)):
        #TODO: Code à compléter

if __name__ == "__main__":
    l = [2, 43, 1, 3, 43]
    tri_insertion(l)
    print(l)
```

— **Java :**

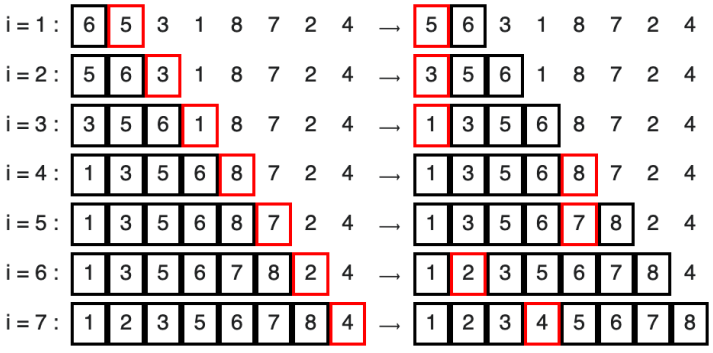
```java
public class question10 {
    public static void tri_insertion(int[] l) {
        for (int i = 1; i < l.length; i++){
            //TODO: Code à compléter
        }
    }

    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 = {2, 43, 1, 3, 43};
        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.



Solution

Python :

```python
def tri_insertion(l):
    for i in range(1, len(l)):
        key = l[i]
        j = i - 1

        while j >= 0 and key < l[j]:
            l[j + 1] = l[j]
            j -= 1
        l[j + 1] = key


if __name__ == "__main__":
    l = [2, 43, 1, 3, 43]
    tri_insertion(l)
    print(l)
```

Java :

```java
public class question10 {
    public static void tri_insertion(int[] l) {
        for (int i = 1; i < l.length; i++){
            int key = l[i];
            int j = i - 1;

            while (j >= 0 && l[j] > key) {
                l[j + 1] = l[j];
                j = j - 1;
            }
            l[j + 1] = key;
        }
    }

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

    public static void main(String[] args){
        int[] l = {2, 43, 1, 3, 43};
        tri_insertion(l);
        printArray(l);
    }
}
```

La complexité de l'algorithme est de $O(n^2)$ car nous utilisons 2 boucles imbriquées, qui dans le pire des cas, parcourent la liste deux fois.

Question 4: (*20 minutes*) **Tri fusion (Merge Sort) en java**

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 ?

— **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
```

```
7       }
8
9       // Fonction principale qui trie arr[l..r] en utilisant
10      // merge()
11      public static void tri_fusion(int arr[], int l, int r){
12          // 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.

Solution

Java :

```java
// Solution question 11 – 1/2
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) {
        // Trouver la taille des deux sous–listes à fusionner
        int n1 = m – l + 1;
        int n2 = r – m;

        /* Créer des listes temporaires */
        int L[] = new int[n1];
        int R[] = new int[n2];

        /*Copier les données dans les sous–listes temporaires */
        for (int i = 0; i < n1; ++i) {
            L[i] = arr[l + i];
        }
        for (int j = 0; j < n2; ++j) {
            R[j] = arr[m + 1 + j];
        }

        /* Fusionner les sous–listes temporaires */
        // Indexes initiaux de la première et seconde sous–liste
        int i = 0, j = 0;

        // Index initial de la sous–liste fusionnée
        int k = l;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) {
                arr[k] = L[i];
                i++;
            } else {
                arr[k] = R[j];
                j++;
            }
            k++;
        }

        /* Copier les élements restants de L[] */
        while (i < n1) {
            arr[k] = L[i];
            i++;
            k++;
        }

        /* Copier les élements restants de R[] */
        while (j < n2) {
            arr[k] = R[j];
            j++;
            k++;
        }
    }

    // Fonction principale qui trie arr[l..r] en utilisant
    // merge()
    public static void tri_fusion(int arr[], int l, int r) {
        if (l < r) {
            // Trouver le milieu de la liste
            int m = (l + r) / 2;

            // Trier les première et la deuxième parties de la liste
            tri_fusion(arr, l, m);
            tri_fusion(arr, m + 1, r);

            // Fusionner les deux parties
            merge(arr, l, m, r);
        }
    }
```

Le tri fusion est un algorithme récursif. Ainsi, nous pouvons exprimer sa complexité temporelle via une relation de récurrence : $T(n) = 2T(n/2) + O(n)$. En effet, l'algorithme comporte 3 étapes :

1. "Divide Step", qui divise les listes en deux sous-listes, et cela prend un temps constant