Exercices 3

Philipp Ahrendt

November 28, 2025

Contents

1 Exercice 1: Palindromes

Un palindrome est une chaîne de caractères qui reste la même lorsqu'on inverse l'ordre des caractères. Par exemple, *kayak* et *elle* sont des palindromes. On considère aussi que la chaîne vide `""` est un palindrome, et que

toute chaîne à un seul caractère, comme par exemple `"a"` , est un palindrome. Écrire une fonction `palindrome` qui teste si une chaîne de caractères est un palindrome ou non.

2 Exercice 2: Tri par insertion récursif

Dans cet exercice, on va reformuler le tri par insertion comme un algorithme récursif. Dans un premier temps, on va considérer une version qui ne manipule pas la liste d'entrée `L`, mais qui rend une nouvelle liste `L_t` triée contenant les mêmes entrées que `L`.

2.1 Question a:

Écrire une fonction `inser(L,n)`, qui prend en entrée une liste `L`, que l'on suppose triée, et un nombre `n`, et renvoie la liste triée `L_1` obtenue en rajoutant `n` à `L`.

2.2 Question b:

Utiliser la fonction `inser` pour écrire une fonction récursive `tri_insertion_rec` qui implémente le tri par insertion.

2.3 Question c:

Écrire une fonction récursive `tri_insertion_rec_direct(L)` qui trie directement la liste d'entrée `L`.

3 Exercice 3: Tours de Hanoï

Dans cet exercice, on va implémenter la solution récursive du jeu des *tours de Hanoï* vue en cours.

La représentation du jeu est la suivante:

- Les trois piliers sont représentés par les caractères `'A'` , `'B'` et `'C'`.
- Un nombre `n` représente le nombre de disques dans le jeu.

Écrire une fonction `hanoi(n,P1,P2)`, qui prend en entrée le nombre `n` de piliers, le pilier de départ `P1` et d'arrivée `P2`, et qui affiche une par une les instructions pour déplacer `n` piliers de `P1` vers `P2`.

Par exemple, `hanoi(3,'A','C')` doit afficher:

```
deplacer disque 1 de A vers C
deplacer disque 2 de A vers B
deplacer disque 1 de C vers B
deplacer disque 3 de A vers C
deplacer disque 1 de B vers A
deplacer disque 2 de B vers C
deplacer disque 1 de A vers C
```

4 Exercice 4: Tri fusion

Cet exercice a pour but d'implémenter un algorithme de tri de type *diviser pour régner*; le **tri fusion**. Pour trier une liste `L`, l'idée est la suivante:

- On découpe `L` en deux morceaux `L1` et `L2`, et on les trie séparément.
- On fusionne les deux listes triées, afin d'obtenir un tri de la liste entière `L`.

4.1 Question a:

Écrire une fonction `fusion`, qui prend en entrée deux listes triées `L1` et `L2`, et renvoie une liste triée `L_fusion` contenant les entrées de `L1` et `L2`.

4.2 Question b:

Utiliser la fonction `fusion` pour écrire une fonction `tri_fusion` qui implémente le tri fusion.

5 Exercice 5: Chemins dans un graphe

Implémenter l'algorithme pour trouver le chemin le plus court dans un graphe, vu en cours.

6 Exercice 6: Sous-listes croissantes

Si `L` est une liste de nombres, une *sous-liste croissante* de `L` est donnée par une suite d'indices `i_1 < i_2 ... < i_k` tels que `L[i_1] <= L[i_2] ... <= L[i_k]`. Le but de cet exercice est de construire un algorithme `crois(L)` qui trouve, pour une liste `L`, la longueur de la sous-liste croissante la plus longue.

Exemple:

- `L = [1,8,6,4,5,7,2,5]`
- sous-liste la plus longue: `[1,4,5,7]`
- `crois(L) == 4`

6.1 Question a:

Pour un indice i de la liste, notons `crois(L,i)` la longueur de la sous-liste croissante la plus longue qui commence à l'indice i. Donner une formule pour `crois(L)` en termes des `crois(L,i)`.

6.2 Question b:

Notons I l'ensemble des indices j $>$ i tels que L[j] $>=$ L[i]. Donner une formule pour `crois(L,i)` en termes des `crois(L,j)`, pour j $\in$ I.

6.3 Question c:

En déduire un algorithme récursif (diviser pour régner) pour calculer `crois(L)` et l'implémenter. Estimer sa complexité.

6.4 Question d:

Implémenter un algorithme suivant une approche de programmation dynamique. Estimer sa complexité