

Amérique du Nord – 2024 – sujet2 - Correction

Exercice 1 (6 points)

1. Fonction `echange` (`tab`, `i`, `j`)

```
def echange(tab, i, j):  
    tmp = tab[i]  
    tab[i] = tab[j]  
    tab[j] = tmp
```

👉 *Commentaire : On utilise une variable temporaire `tmp` pour échanger les valeurs sans les perdre. La fonction modifie la liste sur place et ne renvoie rien (`return inutile`).*

2. Compléter les lignes 6, 7 et 8

Code complet :

```
def triStooge(tab, i, j):  
    if tab[i] > tab[j]:  
        echange(tab, i, j)  
    if (j - i) > 1:  
        k = (j - i + 1) // 3  
        triStooge(tab, i, j - k)  
        triStooge(tab, i + k, j)  
        triStooge(tab, i, j - k)
```

👉 *Commentaire :*

- On trie d'abord les deux premiers tiers : $(i, j-k)$
 - Puis les deux derniers tiers : $(i+k, j)$
 - Puis à nouveau les deux premiers tiers pour finaliser le tri.
-

3. L'algorithme est-il itératif ou récursif ?

Cet algorithme est **récursif**.

👉 *Commentaire : La fonction `triStooge` s'appelle elle-même.*

4. Valeur de k lors de l'appel `triStooge(A, 0, 5)`

On a :

```
k = (j - i + 1) // 3
k = (5 - 0 + 1) // 3
k = 6 // 3
k = 2
```

👉 *Commentaire : Il y a 6 éléments dans le tableau (indices 0 à 5). On divise la taille par 3 avec une division entière.*

k = 2

5. Nombre d'appels récurifs (sans compter l'appel initial)

À chaque appel avec plus de 2 éléments, la fonction appelle **3 nouvelles fois** la fonction.

Arbre des appels :

- Niveau 1 : 3 appels
- Niveau 2 : chacun de ces appels peut encore générer 3 appels

Dans cet arbre précis, on obtient **12 appels récurifs**

👉 *Commentaire : On ne compte pas l'appel initial `triStooge(A, 0, 5)`. On compte uniquement les appels générés par récursion.*

6. Appels dans les cases 1, 2 et 3

Premier appel : `triStooge(A, 0, 5)` avec **k = 2**

Les appels sont :

```
1 triStooge(A, 0, 3)
2 triStooge(A, 2, 5)
3 triStooge(A, 0, 3)
```

👉 *Commentaire : Les deux premiers tiers correspondent à (0, 3). Les deux derniers tiers correspondent à (2, 5). On retri ensuite les deux premiers tiers.*

7. Compléter le tableau ($A = [5, 4, 6, 2]$)

◆ Appel `triStooge(A, 0, 3)`

Avant : $[5, 4, 6, 2]$

Après tri complet : $[2, 4, 5, 6]$

👉 *Commentaire : Le tri de Stooge finit par trier entièrement le tableau.*

◆ Appel intermédiaire

Avant : $[2, 6, 4, 5]$

Appel correspondant : `triStooge(A, 1, 3)`

Après : $[2, 4, 5, 6]$

👉 *Commentaire : On trie uniquement la partie entre les indices 1 et 3.*

◆ Dernière ligne

Appel : `triStooge(A, 1, 3)`

Avant : $[2, 4, 5, 6]$

Après : $[2, 4, 5, 6]$

👉 *Commentaire : Le tableau est déjà trié, donc aucun échange n'est effectué.*

8) Complexité

Le coût dans le pire des cas est : $O(n^{2,71})$

👉 *Commentaire : Cette complexité est très mauvaise. Elle est bien pire que les algorithmes classiques.*

9. Algorithme plus efficace

Exemples d'algorithmes strictement meilleurs :

- **Tri fusion**
- **Tri rapide (quicksort)**

Complexité : $O(n \log n)$

👉 **Commentaire :** Ces algorithmes sont beaucoup plus efficaces pour les grands tableaux. Leur complexité est asymptotiquement inférieure à celle du tri de Stooge.
