

Institut Supérieur du Numérique

Devoir Surveillé - Structures de Données et Complexité (L2- S3). 26 décembre 2025

Durée : 1h30, Langage : Pseudo-code ou C++. Documents : non autorisés Structures STL interdites (vector, list, sort, etc.)

Exercice 1 : Questions de cours (4 pts)

1. Pourquoi l'insertion en tête d'une liste chaînée est en $O(1)$?
2. Comparer tableau et liste chaînée (2 avantages / 2 inconvénients).
3. Donner la complexité de la recherche dans un tableau non trié.
4. Pourquoi l'insertion à la fin d'un tableau partiellement rempli est en $O(1)$

Exercice 2 : Complexité (4 pts)

1. Calculez en fonction de n , le nombre total de comparaisons dans le code suivant (Montrez les étapes intermédiaires) :

$i \leftarrow n$

TantQue ($i \geq 1$) faire

$j \leftarrow 1$

TantQue ($j \leq i$) faire

$j \leftarrow j + 1$

FinTq

$i \leftarrow i - 1$

FinTq

2. En déduire la complexité temporelle de cet algorithme en notation O

Exercice 3 : Tri linéaire (3 pts)

On suppose que nous avons un tableau de n valeurs entières. Chaque valeur est comprise entre 1 et 1000.

1. Proposer un algorithme de tri linéaire exploitant le fait que les valeurs sont entières comprises entre 1 et 1000.
2. Donner sa complexité en temps et en mémoire.

Exercice 4 : Liste doublement chaînée spéciale (9 pts)

On considère une liste doublement chaînée où chaque élément pointe vers son suivant et son précédent. En plus, on garde 3 pointeurs pour manipuler les données de cette liste :

- **head** (début) : pointe sur la tête de la liste
- **tail** (fin) : pointe sur le dernier élément de la liste
- **mid** (milieu) : pointe sur l'élément au milieu de la liste

On garde également le nombre d'élément dans l'attribut **size**.

Questions :

1. Donner les structures de données en C++ permettant de représenter cette structure en mémoire.
2. Implémenter la fonction **AjoutDebut** qui ajoute un élément au début avec les mises à jour nécessaires.
3. Implémenter la fonction **SupprimerFin** qui supprime le dernier élément avec les mises à jour nécessaires.
4. Implémenter la fonction **GetAt** qui prend une liste **L** et une position **i** et retourne la valeur à la position **i**. Exploiter les propriétés de la liste pour optimiser le nombre d'opérations effectuées par cette fonction.