Examen : Structure de donnée

Dr. Jehan-Antoine Vayssade

17 juin 2025

Instructions

— **Durée :** 1 heure 30 min
— **Note totale / maximale :** 20 points
— **Des points bonus peuvent être cachés !**
— **Les réponses doivent être argumentées**
— **Prenez uniquement un stylo**
— **Pour la rédaction des codes :** ils peuvent être rédigés dans des langages du même paradigme que C, tels que Python, C++ ou Java. (Attention aux coûts cachés).

1 Exercice : Notions fondamentales (3 points)

Répondez aux questions suivantes de manière concise et claire :

1. On vous donne deux algorithmes A et B :
   — A a une complexité en temps de $O(n^2)$
   — B a une complexité en temps de $O(n \log n)$
   Pour de très grandes valeurs de n, lequel est plus efficace ? Justifiez votre réponse en utilisant la notion de complexité asymptotique. (0,5 pt)
2. Donnez un exemple simple (avec une phrase ou un pseudo-code) illustrant chacun des trois types de notations asymptotiques :
   — O (grand O)
   — Ω (grand Omega)
   — Θ (grand Thêta)

   (1 pt)
3. Pourquoi utilise-t-on plutôt la notation O dans l'analyse des algorithmes, même si elle donne une majoration approximative ? Quel serait un inconvénient potentiel ? (0,5 pt)
4. La **sémantique axiomatique** est souvent utilisée en vérification formelle.
   — Expliquez brièvement ce qu'elle permet de faire, avec un exemple de contexte où elle est utile. (1 pt)

2 Choix d'algorithmes (3 points)

Dans un système embarqué (ou les performances et la mémoire sont limitées), une routine de tri doit être utilisée pour organiser de petites listes de tâches (entre 5 et 50 éléments). Deux algorithmes sont envisagés pour cette opération :

— **Tri à bulles (Bubble Sort)**
— **Tri par fusion (Merge Sort) utilisant la récurssion**

Questions :

1. (1 point) Donnez la complexité asymptotique, en notation O, des deux algorithmes mentionnés.
2. (1 point) En tenant compte du contexte (tailles de listes entre 5 et 50 éléments), quel algorithme recommanderiez-vous ? Justifiez votre choix en vous appuyant sur les notions de performance pratique et de contraintes système.
3. (1 point) Expliquez pourquoi un algorithme de complexité temporelle théorique plus élevée peut, dans certains cas, être plus efficace en pratique qu'un algorithme théoriquement moins complexe.

3 Distance minimale sur une grille 1D (2 points)

Soit un point $x_0 \in R$ et une grille régulière de points $G = \{i \times s \mid i = 0, \ldots, n-1\}$ où $s > 0$ est le pas. On souhaite calculer la distance minimale entre x_0 et un point de la grille G. Le programme suivant fait cela naïvement :

```
double minDistance(double x0, int n, double s) {
    double minDist = 1e9;
    for (int i = 0; i < n; ++i) {
        double dist = fabs(x0 - i * s);
        if (dist < minDist)
            minDist = dist;
    }
    return minDist;
}
```

1. (1 pt) Quelle est la complexité temporelle de cette fonction en fonction de n ?
2. (2 pt) Donnez une formule analytique permettant de calculer cette distance minimale en temps constant $O(1)$.

4 Arbres binaires de recherche (6 points)

Partie 1 : Compréhension et représentation (3 points)

On considère un arbre binaire de recherche initialement vide. Et l'on suposera les valeurs suivantes : $S = [2, 4, 6, 8, 10, 11, 13, 15, 18, 22, 24, 26]$. En fonction de la stratégie d'inserssion l'arbre résultant diffère.

1. Quelle prioriétée mathématique assure un BST (Binary Search Tree) ?
2. Proposez un BST parfaitement équilibré.
3. Proposez un BST dit dégénéré.
4. Comparez la hauteur h obtenue des deux solutions.
5. Écrivez la complexité dans le pire cas pour une recherche dans ces deux configurations. La complexité dois faire apparaitre une variable h représentant la hauteur de l'arbre.
6. Proposez une méthode pour rééquilibrer l'arbre et justifiez son impact sur les performances, notament dans le pire cas.

Partie 2 : Programmation (3 points)

Écrire une fonction récursive qui renvoie vrai si et seulement si l'arbre passé en paramètre est un arbre binaire de recherche. Vous pouvez utiliser la structure suivante :

```
typedef struct Node {
    char data;
    struct Node *left, *right;
} Node;
```

5 Optimisation (3 points)

Analysez la complexité du code de tri boustrophédon ci-dessous, en termes de temps (comparaison et swap) et d'espace (pire cas et meilleur cas). Expliquez comment la complexité pourrait être améliorée. Proposez des suggestions pour réduire le nombre de comparaisons et de swaps inutiles. Modifiez le code pour optimiser ces aspects et donnez les nouvelles complexités.

```
1  void boustrophedonsort(void *data, int count, size_t size, CompareFunc
       compare) {
2      int start = 0, end = count - 1;
3      char *cdata = (char*)data;
4
5      while (end != start)
6      {
7          for (int i = start; i < end; ++i)
8              if (compare(cdata + i * size, cdata + (i + 1) * size) > 0)
9                  memswap(cdata + i * size, cdata + (i + 1) * size, size)
                       ;
10         end--;
11
12         if(end == start)
13             break;
14
15         for (int i = end; i >= start; --i)
16             if (compare(cdata + (i - 1) * size, cdata + i * size) > 0)
17                 memswap(cdata + (i - 1) * size, cdata + i * size, size)
                       ;
18         start++;
19     }
20 }
```

6 FIFO et LIFO (3 points)

1. (1 pt) Que veulent dire les acronymes suivants :
   — LIFO
   — FIFO
2. (1 pt) Associez chaque scénario suivant à la structure la plus adaptée (`FIFO` ou `LIFO`), et justifiez brièvement :
   — Gestion d'appels récursifs par un interpréteur.
   — Planification des tâches dans une file d'attente d'impression.
   — Navigation dans l'historique d'un navigateur web.
   — Simulation d'un embouteillage routier.
3. (1 pt) Pour chaque opération fondamentale sur une pile ou une file (push, pop, enqueu, dequeu), donnez la complexité temporelle dans les cas suivants :
   — Implémentation avec une liste simplement chaînée.
   — Implémentation avec un tableau dynamique.