TD2 : Algorithmes de Tri

L'objectif de cette séance est d'explorer les différentes approches algorithmiques pour le tri, ainsi que leur complexité. Ce TD vous aidera à devenir capable de :

- Calculer la complexité d'un algorithme récursif
- Analyser les meilleurs et les pires cas d'algorithmes de tri.
- Comparer différentes versions d'un algorithme et décrire en quoi leur complexité diffère.
- Appliquer les techniques des algorithmes de tri afin de résoudre d'autres problèmes.

Instructions :

- Les exercices sont tous à faire, sauf les exercices difficiles marqués par (⚙).
- Lire bien toutes les informations fournies avant de faire chaque exercice du TD.
- Fournissez toujours des justifications valides pour toutes vos réponses.
- Si un exercice demande d'écrire un algorithme, écrivez votre solution en pseudo-code.

1 Complexité des algorithmes récursifs

Exercice 1 : Complexité d'un algorithme récursif. Quelle est la complexité de l'algorithme suivant :

Algorithme 1 Rec1

Données: Deux tableaux `t1`, `t2` de taille n
Résultat: **true** ou **false**

1: **if** $n = 1$ **then**
2: **return** $(\texttt{t1}[0] = \texttt{t2}[0])$ // ***true*** *si égaux, sinon* ***false***
3: **end if**
4: $m \leftarrow \texttt{int}(n/2)$
5: **if** $2m \neq n$ **then**
6: **return false**
7: **end if**
8: **if** $\textsc{Rec1}(\texttt{t1}[0..m], \texttt{t2}[0..m], m) \wedge \textsc{Rec1}(\texttt{t1}[m..n], \texttt{t2}[m..n], m)$ **then**
9: **return true**
10: **else if** $\textsc{Rec1}(\texttt{t1}[0..m], \texttt{t2}[m..n], m) \wedge \textsc{Rec1}(\texttt{t1}[m..n], \texttt{t2}[0..m], m)$ **then**
11: **return true**
12: **else**
13: **return false**
14: **end if**

Note: $t[a..b]$ représente le sous-tableau d'éléments de t aux positions a à $b-1$ (inclus).

Exercice 2 : Application du *master theorem*. Utilisez le *master theorem* pour obtenir des bornes les plus précises possibles pour les récursions ci-dessous :

1. $T(n) = 3T(n/2) + n^3$
2. $T(n) = T(8n/11) + n$
3. $T(n) = 2T(n/4) + \sqrt{n}$
4. $T(n) = 5T(n/3) + n \log n$
5. $T(n) = 2T(n/2) + n/\log n$

2 Algorithmes de Tri

Exercice 3 : Tri par insertion dans le meilleur cas. Dans le tri par insertion, pour insérer un élément à sa place, on peut rechercher d'abord la nouvelle position de l'élément, puis décaler tous les éléments pour mettre l'élément à sa place.

Cette méthode permet d'obtenir un code plus structuré, mais elle ne garantit pas toujours la complexité de $\Theta(n)$ dans le meilleur cas.

1. Quelle est la complexité d'un tri par insertion utilisant la recherche binaire, quand le tableau d'entrée est déjà trié ?
2. Et le tri par insertion utilisant la recherche séquentielle ?
3. L'algorithme, comme présenté dans le cours (TriInsertion, diapositive 37), a-t-il une complexité dans le meilleur cas de $\Theta(n)$?
4. Écrivez un algorithme modifié de recherche séquentielle qui atteint la complexité de $\Theta(n)$ dans le meilleur cas.

Exercice 4 : Tri rapide sur les tableaux triées. Dans cet exercice, nous allons étudier l'algorithme de tri rapide, avec le premier élément comme pivot, et son efficacité sur des tableaux d'entrée déjà triés.

1. Quelle est la complexité du tri rapide pour les tableaux triés ?
2. Quelle est la complexité du tri rapide pour les tableaux triés, si nous utilisons l'élément au milieu comme pivot ?
3. (⚙) Pouvez-vous créer un mauvais tableau pour cet algorithme (en utilisant l'élément du milieu comme pivot) ? Autrement dit, pouvez-vous donner un tableau de 10 éléments qui forcera une profondeur de récursion de 10 ?
4. (⚙) Dans le cas général, décrivez comment construire un tableau de n éléments, de telle sorte que le tri rapide avec pivot au milieu ait une complexité de $\Theta(n^2)$?

Exercice 5 : Problème de sélection. Le problème considéré dans cet exercice est le problème de sélection : étant donné un tableau t d'entiers de taille n (non ordonné), et un entier k, le objectif est de trouver le k-ième plus grand élément dans t.

1. Écrivez un algorithme qui trouve le k-ième plus grand élément dans un tableau de taille n en temps $O(nk)$.
2. En utilisant les idées du tri rapide, écrivez un algorithme pour résoudre le problème de sélection.
3. (⚙) Montrez que cet algorithme est $\Theta(n^2)$ dans le pire cas, mais argumentez qu'il devrait être plus performant ($\Theta(n)$) dans le cas moyen.

Exercice 6 : Exécution d'algorithmes de tri. Exécutez et comparez les algorithmes de tri suivants appliqués au tableau :

$$[6, 7, 2, 4, 8, 9, 11, 1].$$

Algorithmes : tri sélection, tri insertion, tri par tas, tri rapide.

3 Pour aller plus loin

Exercice 7 : Construction de tas alternative (⚙). Dans cet exercice, vous allez écrire une fonction différente pour construire un tas, qui fonctionne en insérant des éléments de manière itérative (un peu comme le tri par insertion).

1. Écrivez une fonction qui insère un élément dans un tas : l'entrée est un entier n et un tableau t de taille au moins $n + 1$, tel que les premiers n éléments de t forment un tas ; la fonction doit insérer $t[n]$ dans le tas de façon à ce que les premiers $n + 1$ éléments de t forment un tas valide.
2. Ecrivez une fonction qui, étant donné un tableau t et sa taille n, le transforme en un tableau tas en effectuant des insertions successives.
3. Quelle est la complexité des deux fonctions ?
4. Quels sont les résultats de cette méthode lorsqu'elle est appliquée au tableau

   $$[9, 4, 5, 13, 16, 1, 10, 14] \ ?$$

   Le résultat est-il le même que celui de la méthode présentée dans le cours ?

Exercice 8 : Algorithme de tri récursif (⚙). Le tri rapide, qui est un algorithme de tri récursif, sépare les éléments selon qu'ils sont plus grands ou plus petits qu'un pivot.

Une autre façon de trier un tableau de manière récursive consiste à le diviser en deux moitiés, à trier chaque moitié de manière récursive, puis à combiner les résultats.

Décrivez la tâche à effectuer pour combiner les deux moitiés et proposez un algorithme de complexité $\Theta(n)$ pour le faire.