Algorithmique 1

Les Algorithmes de Tri

Tarek Boutefara

2025-10-31

Table of Contents

Chapitre 1. Introduction

Après avoir maîtrisé le stockage de collections de données, notamment à l'aide des tableaux, nous abordons l'une des opérations les plus fondamentales et les plus critiques en informatique : le tri. Le tri est le processus qui consiste à réorganiser les éléments d'un ensemble (typiquement le contenu d'un tableau) selon un ordre précis, qui est le plus souvent croissant (du plus petit au plus grand) ou décroissant (du plus grand au plus petit).

L'importance de cette opération est immense. Une structure de données triée permet d'effectuer des recherches ultérieures (comme la recherche d'un nom dans un annuaire ou d'un mot dans un dictionnaire) de manière exponentiellement plus rapide et efficace. De plus, de nombreux problèmes algorithmiques complexes, tels que la recherche des doublons ou la fusion de listes, utilisent le tri comme une étape préparatoire indispensable.

La question n'est donc pas de savoir s'il faut trier, mais comment le faire efficacement. Il n'existe pas un unique "meilleur" algorithme de tri ; il en existe des dizaines, chacun possédant sa propre logique, ses propres avantages et ses propres inconvénients. L'étude des algorithmes de tri est un pilier de l'algorithmique car elle introduit une notion essentielle : l'efficacité (ou la complexité). Certains algorithmes sont très simples à comprendre et à implémenter, mais se révèlent extrêmement lents sur de grands volumes de données, tandis que d'autres sont plus complexes mais bien plus performants.

Dans ce chapitre, nous allons analyser quelques-uns des algorithmes de tri les plus fondamentaux et classiques.

Chapitre 2. Types des algorithmes de tri

2.1. Le Tri "In-Place" (sur place)

L'algorithme réorganise les données à l'intérieur du tableau d'origine. Ce type d'algorithmes utilisent très peu de mémoire supplémentaire.

Exemples :

- Tri par Sélection,
- Tri à Bulles,
- Tri par Insertion,
- Quicksort.

L'iinconvénient est que le tableau d'origine est détruit (perdu) au profit du tableau trié.

2.2. Le Tri "Out-of-Place" (non en place)

L'algorithme nécessite de l'espace mémoire supplémentaire (souvent un ou plusieurs autres tableaux, parfois de taille identique à l'original) pour effectuer le tri. Dans ce cas, le tableau d'origine reste intact.

Exemples :

- Le Tri Fusion (Merge Sort) et
- le Tri par Comptage (Counting Sort).

Chapitre 3. Tri par comptage

Le **tri par comptage** est un algorithme de tri très rapide, mais qui ne fonctionne que dans une situation bien précise. Il n'utilise pas de comparaisons (comme `T[j] < T[IndiceMin]`). C'est un tri **non en place** : il ne modifie pas le tableau original et nécessite un tableau auxiliaire (un "compteur") pour fonctionner.

Condition d'application :

Le tri par comptage est efficace **uniquement si** nous connaissons à l'avance l'intervalle des valeurs à trier, et si cet intervalle est petit et composé d'entiers. L'exemple typique est le tri d'un tableau de notes, où les valeurs sont obligatoirement comprises entre `0` et `20` (un intervalle `[Min..Max]`).

Principe :

L'algorithme utilise un tableau auxiliaire, `TabComptes`, dont les indices correspondent aux valeurs possibles (ex: `[0..20]`).

1. **Phase 1 (Comptage) :** On initialise `TabComptes` à zéro. On parcourt ensuite le tableau original (les notes). Pour chaque note lue, on incrémente la case correspondante dans `TabComptes`.
   - **Exemple :** Si on lit la note `15`, on fait `TabComptes[15] ← TabComptes[15] + 1`.
   - **À la fin :** `TabComptes[15]` contiendra le nombre total de '15' présents dans le tableau original.
2. **Phase 2 (Reconstruction) :** On crée un nouveau tableau `TabResultat` (le tableau trié). On parcourt `TabComptes` (de 0 à 20).
   - **Exemple :** Si `TabComptes[0]` vaut `2` (il y a deux '0'), on écrit deux '0' au début de `TabResultat`. Si `TabComptes[1]` vaut `0`, on ne fait rien. Si `TabComptes[2]` vaut `3`, on écrit trois '2' à la suite...
3. On obtient un nouveau tableau `TabResultat` qui est trié, sans avoir jamais modifié le tableau original.

Tri par comptage

```
Constantes
    NOTE_MIN = 0
    NOTE_MAX = 20

Type
    TypeVecteur = Tableau [1..N] de Entier
    TypeComptes = Tableau [NOTE_MIN..NOTE_MAX] de Entier

Procédure TriComptage (T_Original : TypeVecteur, N : Entier, Var T_Resultat :
TypeVecteur)

Variables
    TabComptes : TypeComptes
    i, j, k : Entier
    NoteLue : Entier

Début
```

```
    Pour i AllantDe NOTE_MIN A NOTE_MAX Faire
        TabComptes[i] ← 0
    FinPour

    Pour i AllantDe 1 A N Faire
        NoteLue ← T_Original[i]
        TabComptes[NoteLue] ← TabComptes[NoteLue] + 1
    FinPour

    k ← 1

    Pour i AllantDe NOTE_MIN A NOTE_MAX Faire
        Pour j AllantDe 1 A TabComptes[i] Faire
            T_Resultat[k] ← i
            k ← k + 1
        FinPour
    FinPour

Fin
```

Chapitre 4. Tri par Sélection

Le tri par sélection est l'un des algorithmes de tri les plus simples à comprendre, car sa logique est très intuitive. Son principe consiste à « trouver le plus petit et à le mettre à sa place » de manière répétée.

L'algorithme divise mentalement le tableau en two parties : une partie triée (à gauche, initialement vide) et une partie non triée (à droite, le reste du tableau).

Le processus est le suivant :

1. On se place à la première case du tableau (la première de la zone non triée).
2. On recherche dans toute la zone non triée (y compris la case où l'on se trouve) l'indice de l'élément le plus petit.
3. Une fois cet indice du minimum (IndiceMin) trouvé, on échange (on permute) l'élément de IndiceMin avec l'élément de la première case.

La première case est maintenant considérée comme "triée". On se décale d'une position (à la deuxième case) et on recommence le processus (recherche du minimum dans le reste du tableau, puis échange) jusqu'à ce que tout le tableau soit trié.

Tri par sélection

```
Procédure TriSelection (Var T : TypeTableau, N : Entier)

Variables
    i, j, IndiceMin : Entier

Début

    Pour i AllantDe 1 A N-1 Faire
        IndiceMin ← i

        Pour j AllantDe i+1 A N Faire
            Si (T[j] < T[IndiceMin]) Alors
                IndiceMin ← j
            Finsi
        FinPour

        Echanger(T[i], T[IndiceMin])

    FinPour

Fin
```

Chapitre 5. Tri à bulles

Le tri à bulles est, comme le tri par sélection, un algorithme de tri "sur place" (in-place) très classique et simple à implémenter.

Son principe est basé sur la comparaison de paires d'éléments adjacents. L'algorithme parcourt le tableau plusieurs fois. À chaque passage, il compare chaque élément avec son voisin immédiat (`T[j]` et `T[j+1]`) et les échange (permute) s'ils sont dans le mauvais ordre.

Ce mécanisme fait que, lors du premier passage complet, l'élément le plus grand du tableau "remonte comme une bulle" pour se placer à la toute dernière position (sa place définitive). Lors du second passage, le deuxième plus grand élément remonte à l'avant-dernière position, et ainsi de suite. L'algorithme s'arrête lorsque tous les éléments ont été "bullés" à leur place correcte.

Bien qu'il soit simple à comprendre, le tri à bulles est notoirement inefficace pour les grands ensembles de données, car il nécessite un très grand nombre d'échanges.

Tri à bulles

```
Procédure TriBulles (Var T : TypeTableau, N : Entier)

Variables
    i, j : Entier

Début

    Pour i AllantDe 1 A N-1 Faire
        Pour j AllantDe 1 A N-i Faire
            Si (T[j] > T[j+1]) Alors
                Echanger(T[j], T[j+1])
            Finsi
        FinPour
    FinPour

Fin
```

Une version plus optimisée permet d'arrêter la boucle dès que le parcours de la table ne donne plus lieu à des permutations, c'est-à-dire, tout élément dans `T[i]` est inférieur à l'élément dans `T[i+1]`.

Tri à bulles optimisé

```
Procédure TriBullesOptimise (Var T : TypeTableau, N : Entier)

Variables
    i, j : Entier TableauTrie : Booléen

Début

    i ← 1
    TableauTrie ← FAUX
```

```
    TantQue (i <= N-1) ET (TableauTrie = FAUX) Faire

        TableauTrie ← VRAI

        Pour j AllantDe 1 A N-i Faire
            Si (T[j] > T[j+1]) Alors
                Echanger(T[j], T[j+1])
                TableauTrie ← FAUX
            Finsi
        FinPour

        i ← i + 1

    FinTantQue

Fin
```