R1.01 - Initiation au développement

Date: 5-7 Dec 2023

Sujet: Introduction au tri

Instructeur: Joseph AZAR - joseph.azar@univ-fcomte.fr

Stratégie

L'HISTOIRE SE SOUVIENT DES GÉNÉRAUX qui utilisent une stratégie solide pour obtenir de grands résultats. Exceller dans la résolution de problèmes nécessite d'être un bon stratège. Les principales stratégies de conception d'algorithmes :

Gérer les tâches répétitives par itération 

Itérer élégamment en utilisant la récursivité 

Diviser pour régner 

Utilisez la force brute lorsque vous êtes paresseux mais puissant 

Testez les mauvaises options puis revenez en arrière (Backtracking) 

Gagner du temps avec les heuristiques 

identifiez dynamiquement les anciens problèmes pour ne plus gaspiller d'énergie 

etc

Tri

Avant les ordinateurs, le tri des données était un goulot d'étranglement majeur qui prenait énormément de temps à effectuer manuellement. Lorsque la Tabulating Machine Company (qui deviendra plus tard IBM) a automatisé les opérations de tri dans les années 1890, elle a accéléré la compilation des données du recensement américain de plusieurs années.

Le tri, comme la recherche, est une tâche courante en programmation informatique. De nombreux algorithmes différents ont été développés pour trier:

- Tri à bulles (Bubble sort)
- Tri par sélection (Selection sort)
- Tri par insertion (Insertion sort)
- Tri par fusion (Merge sort)
- Tri par tas (Heap sort)
- Tri rapide (Quick sort)
- etc

Paramètres pour évaluer un algorithme de tri :

- La stabilité
- Complexité spatiale (space complexity)
- Complexité du meilleur cas (best case complexity)
- Complexité moyenne des cas (average case complexity)
- Complexité dans le pire des cas (worst case complexity)

Tri à bulles (Bubble sort)

Idée

Le tri à bulles compare chaque paire successive d'éléments dans une liste non ordonnée et inverse les éléments s'ils ne sont pas dans l'ordre. Bon pour les tableaux courts.

Si nous regardons les éléments du tableau trié :
Le plus grand élément est au **(n - 1)** ème indice, le 2ème plus grand est au **(n - 2)** ème indice, et ainsi de suite.
Donc, une idée de base serait de parcourir le tableau et de placer le plus grand élément au **(n - 1)** ème index. Encore une fois, parcourez le tableau et placez le deuxième plus grand élément au **(n - 2)** ème index et ... ainsi de suite.

Après la 1ère itération, le plus grand élément remonte vers le **(n - 1)** ème index. De même, après la 2ème itération, le 2ème plus grand élément remonte vers le **(n - 2)** ème indice. Etc!

En général, à toute **ième** itération, on place le ième élément maximum au **(n - i)** ième indice. Ce processus se poursuivra jusqu'à ce que tout le tableau soit trié.

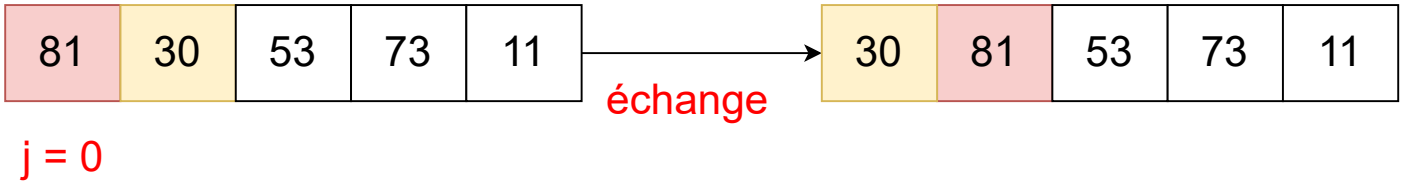
Tri à bulles (Bubble sort)

Exemple

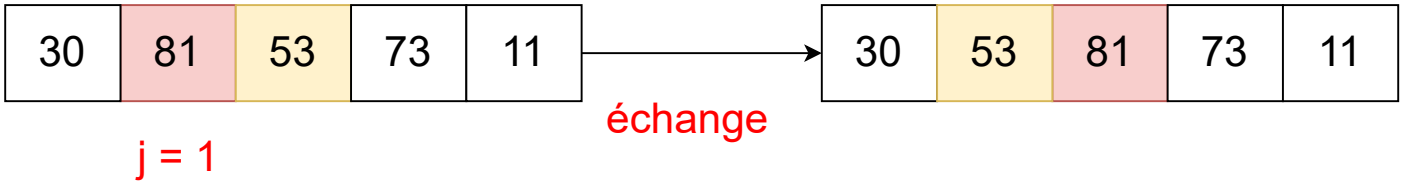
Tableau d'entrée

81	30	53	73	11

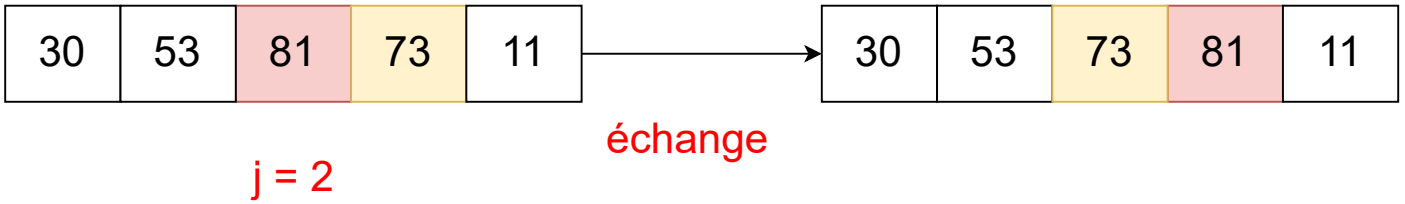
Iteration 1



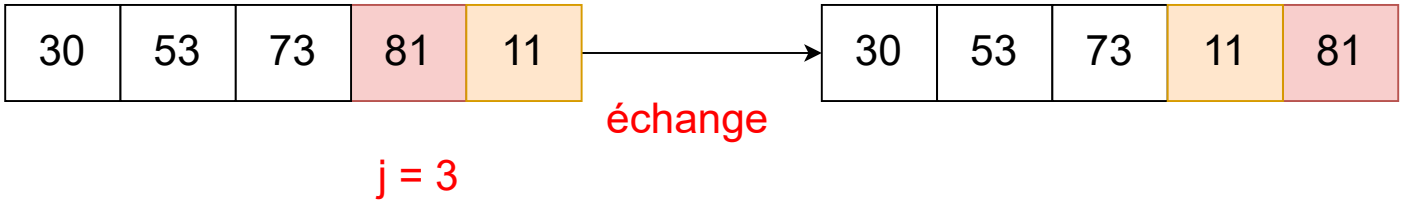
j = 0



j = 1



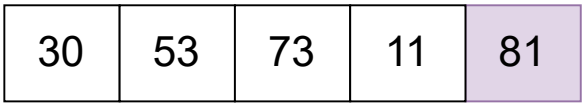
j = 2



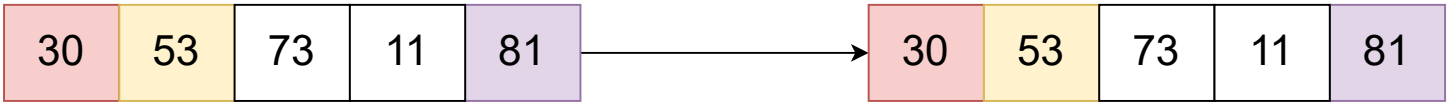
j = 3

Tri à bulles (Bubble sort)

Exemple

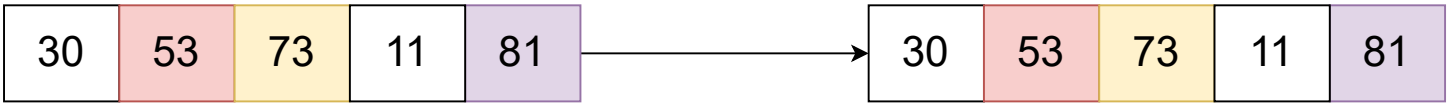


Iteration 2



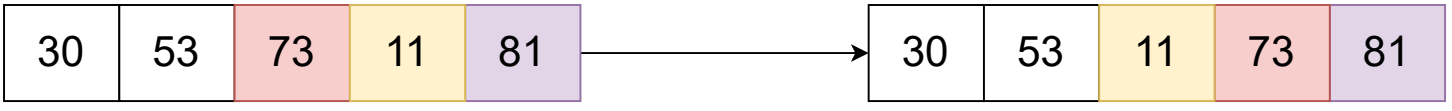
pas d'échange

j = 0



pas d'échange

j = 1

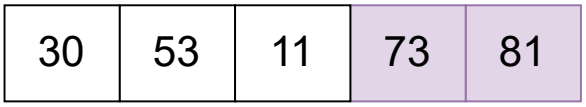


échange

j = 2

Tri à bulles (Bubble sort)

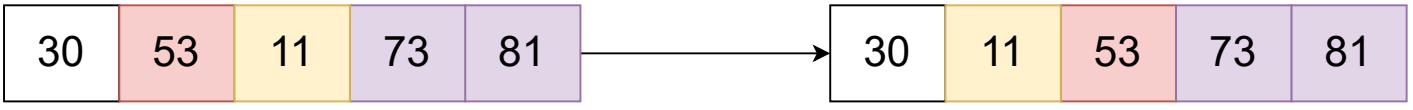
Exemple



Iteration 3

j = 0

pas d'échange

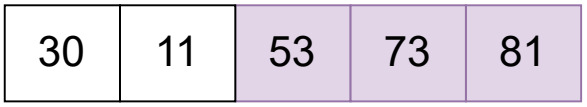


j = 1

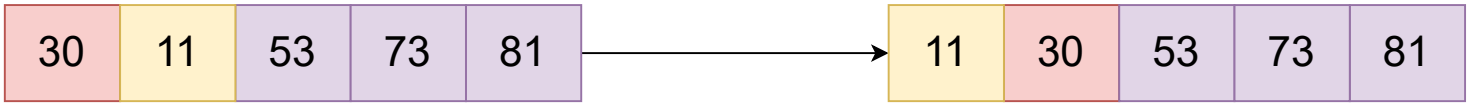
échange

Tri à bulles (Bubble sort)

Exemple

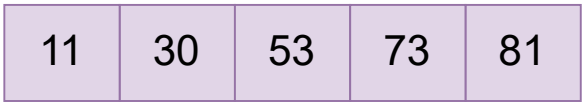


Iteration 4



j = 0

échange



La liste est maintenant triée !!

Tri à bulles (Bubble sort)

Algorithme 1

Pour i allant de 0 à N-1

Pour j allant de 0 à N-1-i

Si tab[j] > tab[j+1]

échanger(tab,j,j+1)

Fi Si

Fin pour

Fin pour

Algorithme 2

Pour i allant de N-1 à 0

Pour j allant de 0 à i-1

Si tab[j] > tab[j+1]

échanger(tab,j,j+1)

Fi Si

Fin pour

Fin pour

Tri par sélections (Selection sort)

Idée

Le tri par sélection est un algorithme de tri, en particulier un tri par comparaison sur place.

Ce tri est inefficace sur les grandes listes. Le tri par sélection est réputé pour sa simplicité et présente des avantages en termes de performances par rapport à des algorithmes plus compliqués dans certaines situations, en particulier lorsque la mémoire auxiliaire est limitée.

L'algorithme divise la liste d'entrée en deux parties :
la sous-liste des éléments déjà triés, qui est construite de gauche à droite au début (à gauche) de la liste, et la **sous-liste des éléments restant à trier qui occupent le reste de la liste**.

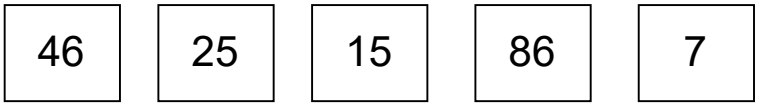
Initialement, la sous-liste triée est vide et la sous-liste non triée est la liste d'entrée entière. L'algorithme procède en trouvant l'élément le plus petit (ou le plus grand, selon l'ordre de tri) dans la sous-liste non triée, en **l'échangeant avec l'élément non trié le plus à gauche** et en déplaçant les limites de la sous-liste d'un élément vers la droite .

> En bref : le tri par sélection trouve le plus petit nombre dans la liste et l'échange avec le premier élément. Il trouve ensuite le plus petit nombre restant et l'échange avec le deuxième élément, et ainsi de suite, jusqu'à ce qu'il ne reste plus qu'un seul nombre.

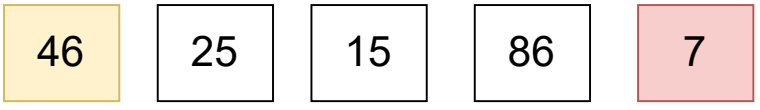
Tri par sélections (Selection sort)

Exemple

Tableau d'entrée



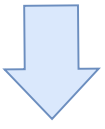
Iteration 1

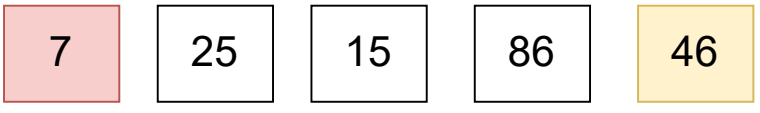


i = 0

min

La valeur minimale est 7 et la première valeur est 46 dans la sous-liste non triée.

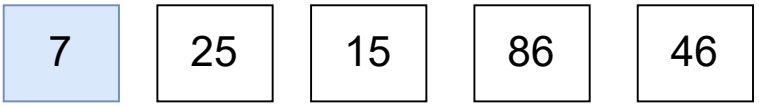




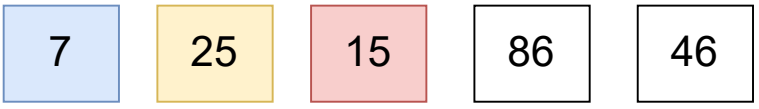
7 est échangé avec 46

Tri par sélections (Selection sort)

Exemple

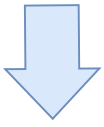


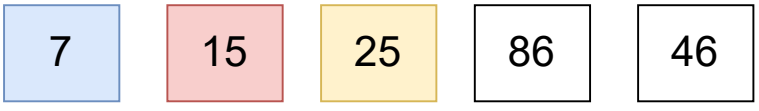
Iteration 2



i = 1 min

La valeur minimale est 15 et la première valeur est 25 dans la sous-liste non triée.

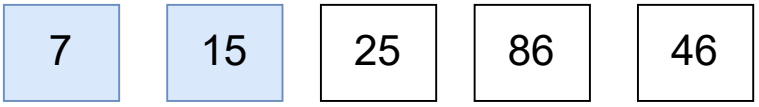




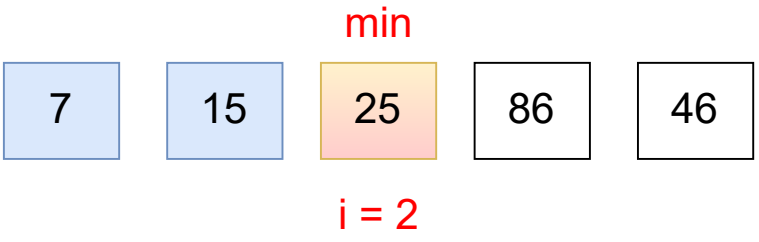
15 est échangé avec 25

Tri par sélections (Selection sort)

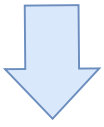
Exemple



Iteration 3



La valeur minimale est 25 et la première valeur est 25 dans la sous-liste non triée.



L'élément minimum est le premier élément de la liste restante. Aucun échange n'est nécessaire.

Tri par sélections (Selection sort)

Exemple

Iteration 4

La valeur minimale est 46 et la première valeur est 86 dans la sous-liste non triée.

46 est échangé avec 86

La liste est maintenant triée !!

Tri par sélections (Selection sort)

Algorithme

```
fonction tri_sélection(liste[1..n], k)
    Pour i allant de 0 à k-1
        minIndex = i
        minVal = liste[i]
        Pour j allant de i+1 à k-1 inclus
            Si liste[j] < minVal
                minIndex = j
                minVal = liste[j]
        echanger(liste[i], liste[minIndex])
```

Remarque : Un algorithme de tri est dit **stable** si deux objets avec des clés égales ou identiques apparaissent dans le même ordre dans la sortie triée qu'ils apparaissent dans le tableau d'entrée non trié.

Entrée : 4_A 5 3 2 4_B 1

Non stable

Sortie : 1 2 3 4_B 4_A 5

Tri par insertion (Insertion sort)

Idée

Le tri par insertion est un algorithme de tri simple qui fonctionne de la même manière que vous triez les cartes à jouer entre vos mains. Le tableau est virtuellement divisé en une partie triée et une partie non triée. Les valeurs de la partie non triée sont sélectionnées et placées à la bonne position dans la partie triée.

Cet algorithme est l'un des algorithmes les plus simples avec une implémentation simple.

Fondamentalement, le tri par insertion est efficace pour les petits tableaux.

Le tri par insertion est de nature adaptative, c'est-à-dire qu'il convient aux ensembles de données qui sont déjà partiellement triés.

la différence principale entre le tri à bulles et le tri par insertion est que le tri à bulles effectue le tri en vérifiant les éléments de données voisins et en les échangeant s'ils sont dans le mauvais ordre tandis que le tri par insertion effectue le tri en transférant un élément à la fois dans un tableau partiellement trié.

Tri par insertion (Insertion sort)

Exemple

6 5 3 1 8 7 2 4

https://cours-info.iut-bm.univ-fcomte.fr/upload/supports/S1/ALGO/JAZAR/Partie_2/Difference-Between-Bubble-Sort-and-Insertion-Sort_Figure-2.gif

Tri par insertion (Insertion sort)

Algorithme

```
fonction tri_insertion(liste[1..n], k)
    Pour i allant de 1 à k-1 inclus
        clé = liste[i]
        j = i - 1
    // Déplacer les éléments de liste[0..i-1], qui sont supérieurs à clé, vers une position devant leur position actuelle
        Tant que j >=0 et liste[j] > clé
            liste[j+1] = liste[j]
            j = j - 1
        liste[j+1] = clé
```

Tri dichotomique par insertion (Binary Insertion sort)

Le tri par insertion binaire est un algorithme de tri similaire au tri par insertion et utilise la recherche binaire pour trouver l'emplacement où un élément doit être inséré.

Approche pour implémenter le tri par insertion binaire :

- Itérer le tableau du deuxième élément au dernier élément.
- Stocker l'élément courant tab[i] dans une variable "clé".
- Trouvez la position de l'élément juste supérieur à tab[i] dans le sous-tableau de tab[0] à tab[i-1] en utilisant la recherche binaire. Disons que cet élément est à l'index "pos".
- Décaler tous les éléments de l'index "pos" à i-1 vers la droite.
- tab["pos"] = "clé".

Tri dichotomique par insertion (Binary Insertion sort)

```java
public class TriInsertion {
    static void tri_insertion(int[] tab){
        int n = tab.length;
        for(int i = 1; i < n; i++){
            int cle = tab[i];
            int j = i - 1;
            while(j>=0 && tab[j] > cle){
                //decalage a droite
                tab[j+1] = tab[j--];
            }
            tab[j+1] = cle;
        }
    }
    static void afficher_tab(int[] tab){
        for (int elem : tab){
            System.out.print(elem + " ");
        }
        System.out.println();
    }
    // avec modification: trouver l'elem a la pos
    // juste superieur a id dans tab
    static int recherche_dichotomique(int[]tab, int id,
                                      int low, int high){
        while(high >= low){
            int milieu = (high+low) / 2;
            if(id < tab[milieu]){
                high = milieu - 1;
            }else if(id == tab[milieu]){
                // pour respecter la stabilite
                return milieu + 1;
            }else {
                low = milieu + 1;
            }
```

```
        }
        return low;
    }
    static void tri_insertion_dichotomique(int[] tab){
        int n = tab.length;
        for(int i = 1; i < n; i++){
            int cle = tab[i];
            int j = i - 1;

            int pos = recherche_dichotomique(tab,cle,0,j);

            while(j>=pos){
                //decalage a droite
                tab[j+1] = tab[j--];
            }
            tab[pos] = cle;
        }
    }
    public static void main(String[] args){
        int arr[] = {12,11,13,5,6};
        afficher_tab(arr);
        //tri_insertion(arr);
        tri_insertion_dichotomique(arr);
        afficher_tab(arr);
    }
}
```