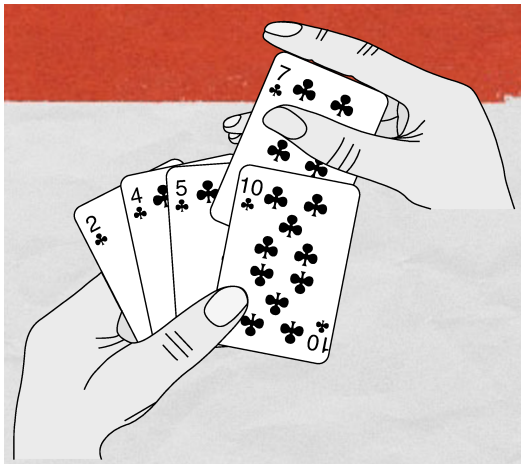
Algorithmique de tris
Tris par insertion

Chap. 17,02

1 Tri du joueur de cartes

Le tri par insertion est un tri « naturel » souvent qualifié de « tri du joueur de carte ». Comment un joueur de carte peut-il trier les cartes ?

- Au début, la main gauche du joueur est vide et ses cartes sont posées sur la table.
- Le joueur prend alors sur la table les cartes, une par une avec sa main droite, pour les placer dans sa main gauche.
- Pour savoir où placer une carte dans son jeu, le joueur la compare avec chacune des cartes déjà présentes dans sa main gauche, *en examinant les cartes de la droite vers la gauche*.
- À tout moment, les cartes tenues par la main gauche sont triées ; ces cartes étaient, à l'origine, les cartes *situées au sommet de la pile sur la table*.



1) Choisir sept cartes à jouer. Les placer en ligne au hasard sur une table et mettre en œuvre la technique décrite ci-dessus. **Se filmer pendant toute l'opération en commentant chacune des étapes !**

2 Tri par insertion

2.1 Introduction

La méthode du tri par insertion est ilustré à cette adresse : https://youtu.be/K4CuPzdiAIo (ou, de façon plus folklorique ... à cette adresse : https://youtu.be/ROalU379l3U).

2) Visualiser la première vidéo (s'arrêter au bout de 4min40). Essayer de bien comprendre la méthode.

3) La méthode étant bien comprise, choisir sept cartes à jouer (avec les valeurs numériques, pas des figures). Les placer en ligne au hasard sur une table et les trier en appliquant le tri par insertion. **Se filmer pendant toute l'opération en commentant chacune des étapes !**

2.2 Algorithme

Le tri par insertion est efficace lorsqu'il s'agit de trier un petit nombre d'élements.

Idée. À chaque étape du processus, le tableau est divisé en deux sous-tableaux : le premier (indices plus petits que l'indice courant) est trié, le second (indices plus grands que l'indice courant n'est pas encore trié). On cherche la place de l'élément courant (non encore trié) dans le tableau trié.

Le *tri par insertion* est basé sur l'utilisation de deux boucles imbriquées :

- La première boucle, une boucle `Pour`, parcourt la liste des valeurs, de la deuxième à la dernière ;
- La seconde boucle, une boucle `TantQue`, cherche à placer l'élément courant dans la première partie triée du tableau (indices inférieurs à l'indice courant).

Remarque. Dans l'algorithme ci-dessous, le comportement « Python » a été utilisé : `nb` *étant le nombre de valeurs dans le tableau, les indices de ce derniers varient de 0 à* $nb-1$

Algorithme 1

```
Fonction tri_insertion(tab, nb)
Déclarations
        Paramètre tab           tableau[20] d'Entiers
        Paramètre nb            Entier, nbre d'élements du tableau
        Variables i, j, clé     Entiers
Début
        Pour i variant de 1 à nb-1 Faire
                clé = tab[i]  // sauvegarde de la valeur de l'indice courant
                // insère tab[i] dans le tableau trié tab[0, ..., i-1]
                j = i - 1
                TantQue j >= 0 et tab[j] > clé Faire
                        tab[j + 1] = tab[j]
                        j = j - 1
                FinTantQue
                tab[j + 1] = clé
        FinPour
Fin
```

Remarque. L'algorithme du tri par insertion est un *algorithme de tri en place*. La réorganisation du tableau ne nécessite *pas la création d'un nouveau tableau*, ce qui économise de la place en mémoire.

4) Faire tourner « à la main » l'algorithme lorsque la fonction reçoit le tableau `tab = [5,2,4,6,1,3]` et la variable `nb = 6`.

5) Implémenter la fonction en langage Python et la tester en l'appelant avec les arguments donnés à la question précédente.

6) Comment prouve-t-on, de façon générale, la terminaison d'un algorithme ?

7) Est-il nécessaire de prouver la terminaison de la boucle externe de cet algorithme ? De la boucle interne ?

8) Démontrer la terminaison de cet algorithme.

9) Comment prouve-t-on, de façon générale, la correction d'un algorithme ?

10) Définir un invariant de boucle et prouver que l'algorithme est correct.

11) Déterminer la complexité de l'algorithme.