Chapitre 2

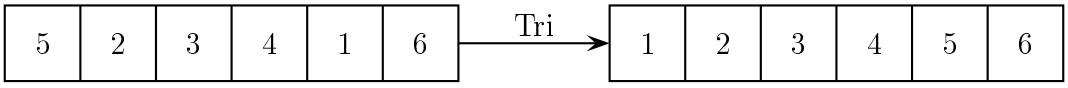
Algorithmes de tri et de recherche dichotomique

2.1 Trier

Durant la seconde guerre mondiale, Betty Holberton fut embauchée en tant que calculatrice, et fut vite choisie pour être l'une des six programmatrices de l'ENIAC, le premier ordinateur entièrement électronique. Elle y développe notamment la première routine de tri.

Le tri est un outil essentiel de l'informatique. Il existe de nombreux algorithmes, plus ou moins performants et plus ou moins complexes à mettre en œuvre mais ils sont fondamentaux pour gérer de grandes quantités de données et y accéder dans des délais très brefs.

La fonction d'un algorithme de tri est d'organiser un ensemble de données selon un ordre déterminé. Généralement les données à trier sont des nombres ou des chaînes de caractères qui peuvent être triés numériquement ou alphabétiquement dans l'ordre croissant ou décroissant.



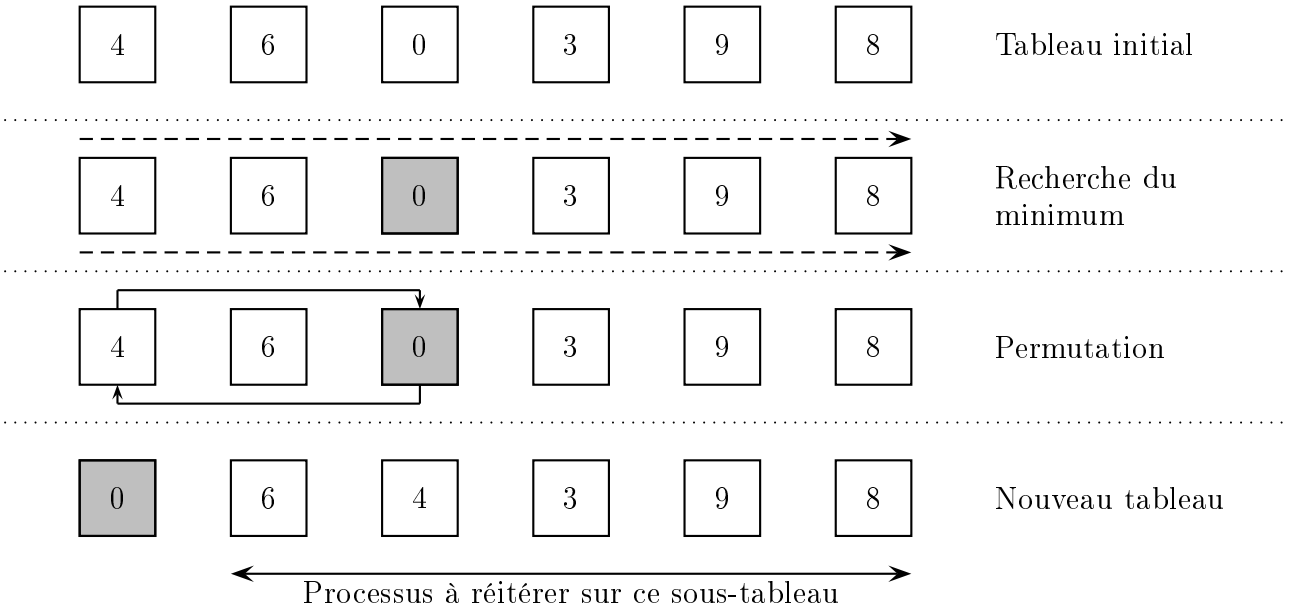
Si le coût de la vérification du tri dépend linéairement de la taille N du tableau à trier – i.e. l'ordre de grandeur du nombre d'opérations à effectuer est égal N –, l'action de trier, elle, peut s'avérer très coûteuse. En effet, les tris ne peuvent pas s'effectuer avec un coût d'ordre N, ils ont nécessairement un coût supérieur. Ceux que nous verrons dans ce chapitre ont un coût d'ordre N^2 ; il en existe toutefois ayant un coût inférieur à cela qui seront vus en terminale.

2.2 Tri par sélection

2.2.1 Principe

On commence par rechercher le plus petit élément m du tableau. Une fois trouvé, on le place en première position du tableau en effectuant un échange avec l'élément qui occupe initialement cette première place. On s'est ainsi assuré que le premier élément du tableau est le plus petit.

On réitère ensuite avec le sous-tableau constitué des éléments suivants et ainsi de suite. À la fin de chaque étape, le minimum du sous-tableau est placé à la tête de celui-ci.



2.2.2 Algorithme

Algorithme 1 : Tri par sélection

Données : tableau T

```
1 N ← longueur(T)
2 Pour i allant de 0 à N − 2 :
3     i_min = 0 ← i
4     Pour j allant de i + 1 à N − 1 :
5         Si T[j] < T[j] :
6             i_min ← j
7     Si i_min ≠ i :
8         Échanger T[i] et T[i_min]
```

Remarque : il est aussi possible de procéder avec le maximum à la place du minimum, voire les deux en mêmes temps.

2.2.3 Terminaison de l'algorithme

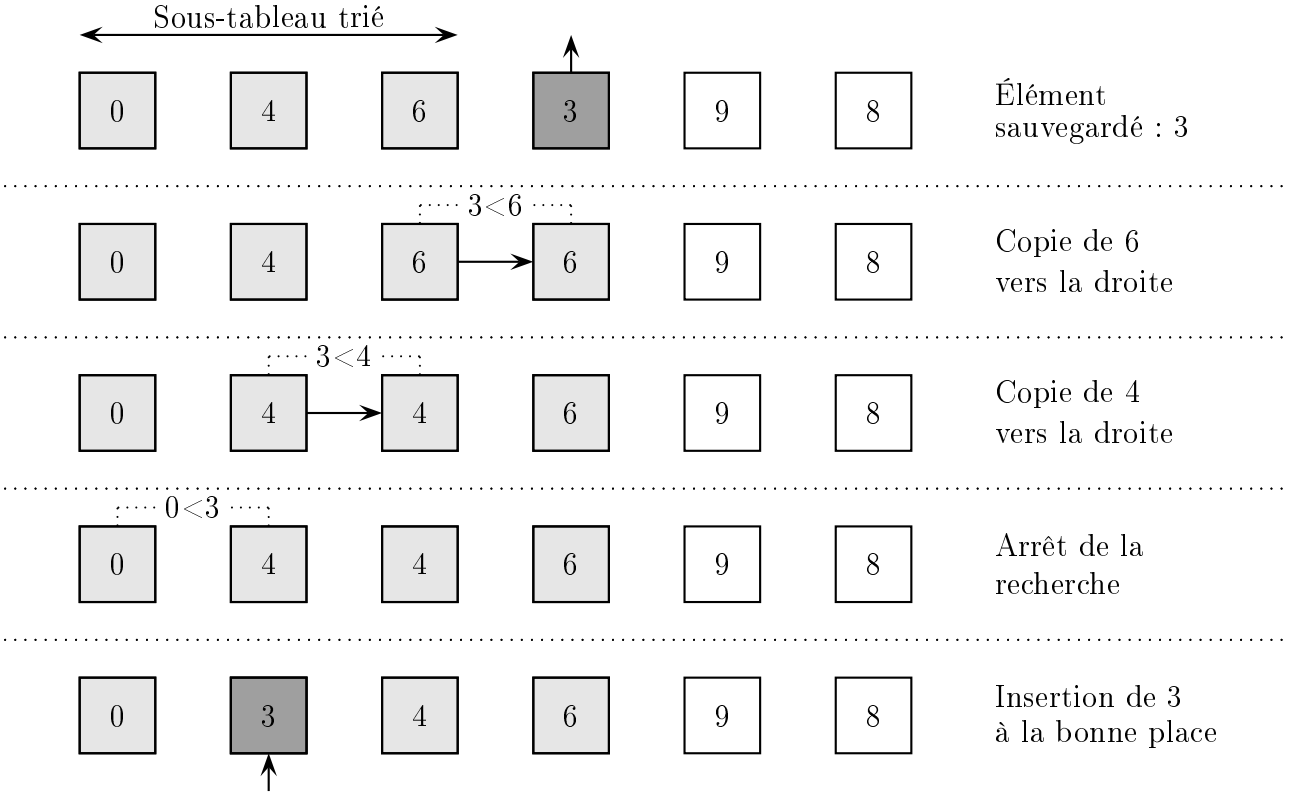
Comme les deux boucles de l'algorithme sont des boucles **Pour**, celles-ci sont bornées et donc l'algorithme se termine.

Comme pour le tri par insertion, si on note N la taille de notre tableau, on a N répétitions de la boucle **Pour** et au plus N répétitions de la deuxième boucle **Pour** au sein de celle-ci. Ce qui fait au maximum $N \times N = N^2$ répétitions de boucles. Le coût de cet algorithme est dans le pire des cas quadratique lui aussi.

2.3 Tri par insertion

2.3.1 Principe

Le **tri par insertion** s'inspire de la manière dont la plupart des gens trient une poignée de cartes. On commence avec une main gauche vide et les cartes face contre table. On retire ensuite du paquet une carte à la fois, pour l'insérer à sa bonne place dans la main gauche. Pour trouver cette bonne place, on la compare avec chacune des cartes déjà présentes dans la main de la dernière à la première. Si la carte à placer est plus petite que la carte à laquelle on la compare, on décale cette dernière vers la droite puis on compare à la suivante. On insère la carte par la droite dès que l'on trouve une carte qui est plus petite.



2.3.2 Algorithme

Algorithme 2 : Tri par insertion

Données : tableau T

```
1 N ← longueur(T)
2 Pour i allant de 1 à N − 1 :
3     element ← T[i]
4     j ← i
5     Tant que j > 0 et element < T[j − 1] :
6         T[j] ← T[j − 1] # copie vers la droite
7         j ← j − 1 # on va regarder la carte suivante à gauche
8     T[j] ← element # insertion
```

2.3.3 Terminaison de l'algorithme

Pour prouver que l'algorithme se termine, il faut étudier les deux boucles.

Boucle Pour : c'est une boucle bornée donc le nombre de passages est déterminé et fini.

Boucle Tant que : les valeurs prises constituent une suite d'entiers décroissante incluse dans la suite d'entiers de j à 1. Il y a donc au plus j passages dans cette boucle **Tant que**.

Les deux boucles se terminent donc l'algorithme se termine.

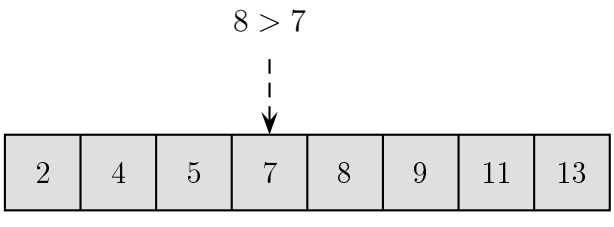
Remarque : si on note N la taille de notre tableau, on a N répétitions de la boucle **Pour** et au plus N répétitions de la boucle **Tant que** au sein de celle-ci. Ce qui fait en tout $N \times N = N^2$ répétitions de boucles. On dira que le coût de cet algorithme est dans le pire des cas quadratique.

2.4 Recherche dichotomique

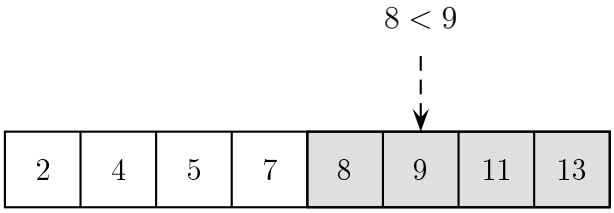
2.4.1 Principe

La recherche dichotomique est une méthode de recherche d'une valeur dans un tableau trié. Dans celle-ci, on compare la valeur recherchée à la valeur de l'élément situé au milieu du tableau. Si cette dernière est plus grande que la valeur recherchée, c'est qu'il faut chercher dans la partie gauche du tableau ; si elle est plus petite, c'est qu'il faut chercher dans la partie droite. Chacune des deux parties du tableau constitue un sous-tableau dans le lequel on réitère le procédé.

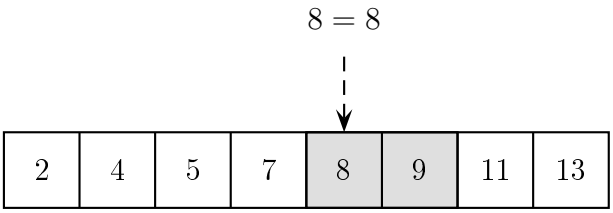
Exemple : On recherche 8 dans le tableau ci-dessous. On commence par comparer à l'élément à la positon $\left\lfloor \frac{i_D + i_F}{2} \right\rfloor$ partie entière de $\frac{i_D + i_F}{2}$ (égale au reste la division euclidienne par 2) où i_D et i_F sont respectivement les indices de début et de fin du sous-tableau considéré.



Étape 1 :
Position du pivot : $\lfloor (8+0)/2 \rfloor = 4$. 8 est plus grand que la valeur pivot : 7. On réitère le processus sur la partie droite du tableau. La partie blanche dans le schéma représente la partie du tableau qui n'est plus considérée.



Étape 2 :
Position du pivot : $\lfloor (8+5)/2 \rfloor = 6$. 8 est cette fois-ci plus petit que la valeur pivot : 9. On réitère à nouveau le processus sur la partie gauche du sous-tableau actuellement considéré.



Étape 3 :
Position du pivot : $\lfloor (5+6)/2 \rfloor = 5$. La valeur du pivot est celle recherchée, l'algorithme prend fin.

2.4.2 Algorithme

Algorithme 3 : Recherche dichotomique

Données : tableau T, *element*
$i_D \leftarrow 0$
$i_F \leftarrow longueur(T)$
Tant que $i_D < i_F - 1$ **:**
  $i_{milieu} \leftarrow (i_D + i_F)//2$
  Si $element < T[i_{milieu}]$ **:**
    $i_F \leftarrow i_{milieu}$
  Sinon
    $i_D \leftarrow i_{milieu}$
Si $T[i_D] = element$ **:**
  Sorties : i_D
Sinon
  Sorties : L'élément n'est pas présent dans le tableau.

2.4.3 Terminaison de l'algorithme

Il s'agit de prouver que la boucle **Tant que** finit par s'arrêter, autrement dit que sa condition d'exécution devient fausse.

On note a_n et b_n les valeurs respectives de i_D et i_F à la n-ième itération de la boucle **Tant que**. On pose l_n la longueur de l'intervalle $[a_n\,;b_n]$; autrement dit $l_n = b_n - a_n$. On a donc avant la première itération $l_0 = b_0 - a_0$. Soit $n \in \mathbb{N}$, à la $n+1$-ième itération de la boucle **Tant que**, on a deux cas possibles :

1er cas : $i_F \leftarrow i_{milieu}$ **;** on a alors $a_{n+1} = a_n$ et $b_{n+1} = \dfrac{a_n + b_n}{2}$, d'où

$$l_{n+1} = b_{n+1} - a_{n+1} = \frac{a_n + b_n}{2} - a_n = \frac{b_n - a_n}{2} = \frac{1}{2} l_n.$$

2e cas : $i_D \leftarrow i_{milieu}$ **;** on a alors $a_{n+1} = \dfrac{a_n + b_n}{2}$ et $b_{n+1} = b_n$, d'où

$$l_{n+1} = b_{n+1} - a_{n+1} = b_n - \frac{a_n + b_n}{2} = \frac{b_n - a_n}{2} = \frac{1}{2} l_n.$$

Dans tous les cas, nous avons donc $l_{n+1} = \dfrac{1}{2} l_n$, autrement dit est une suite géométrique de

raison $\frac{1}{2}$ et premier terme l_0. Comme la raison de la suite appartient à $]0\,;1[$, on en déduit que $l_n \to 0$ lorsque $n \to +\infty$. Il existe donc $n \in \mathbb{N}$ tel que

$$l_n < 1 \iff b_n - a_n < 1.$$

Il existe donc une itération de la boucle **Tant que** pour laquelle sa condition d'exécution devient fausse : $1 < bSup - bInf$. Cette boucle prend alors fin et l'algorithme se termine.

2.5 Exercices

Exercice 2.1. [Trié ?] Écrire une fonction déterminant si un tableau est trié ou non. Elle renverra le résultat sous la forme d'un booléen. Quelle est la complexité de l'algorithme associé à cette fonction ?

Exercice 2.2. [Tri par insertion]

1. Écrire un programme Python effectuant un tri par insertion.
2. À partir du programme précédent, écrire un programme triant un tableau dans l'ordre décroissant sans créer un autre tableau.

Exercice 2.3. [Tri par sélection] Écrire un programme Python effectuant un tri par sélection.

Exercice 2.4. [Recherche dichotomique] Écrire un programme Python effectuant une recherche dichotomique sur un tableau trié. On ajoutera un test pour déterminer si la valeur recherchée appartient bien à l'intervalle formé par les bornes du tableau.

Exercice 2.5. [Médiane, quartiles...] Programmer des fonctions déterminant les médiane, quartiles, déciles et centiles d'une liste de valeurs. Les fonctions s'assureront qu'il est pertinent de calculer ces quantités.