# Chapitre 12

# Tri par sélection du minimum

Comparer des algorithmes de tri entre eux permet de voir qu'ils existent plein d'algorithme de tri et de questionner leur efficacité.
On ne connaît rien sur les valeurs à trier. On verra que lorsqu'on connaît des informations, il est souvent possible d'améliorer les choses (tri par comptage).

Il existe plein de manières différentes de trier des tableaux :

- tri à bulle
- tri par sélection (du minimum)
- tri par insertion (d'une valeur à sa bonne place)
- tri par partition – fusion
- tri rapide
- tri par comptage (demande une connaissance particulière sur les valeurs)

Cette année, nous voyons les tri par sélection et par insertion, vous verrez le tri par partition-fusion l'année prochaine en terminale NSI.

## 1 Le tri par sélection du minimum

`min` est un mot-clé réservé en python à la fonction `min()` : on ne pourra pas l'utiliser comme variable !

Le **tri par sélection du minimum** (souvent simplement appelé *tri par sélection*) consiste

1. à chercher l'indice `mini` de la valeur minimale du sous-tableau `T[i..n]`
2. à permuter `T[i]` avec `T[mini]` : la valeur minimale du sous-tableau est mise en premier
3. à recommencer sur le sous-tableau `T[i+1..n]`

On commence à l'indice `i=0` et on arrête quand le tableau `T` a été parcouru en entier.

**Exercice 31** (Exemple de tri sélection)

Appliquer le tri sélection sur le tableau `T=[7,13,12,-2,2,5]` : expliquer à quoi correspond le tableau `T` à la fin de chaque itération, préciser les valeurs des indices `i` et `mini`.

| | 7 | 13 | 12 | -2 | 2 | 5 |
|---|---|---|---|---|---|---|
| i=0 | -2 | 13 | 12 | 7 | 2 | 5 |
| i=1 | -2 | 2 | 12 | 7 | 13 | 5 |
| i=2 | -2 | 2 | 5 | 7 | 13 | 12 |
| i=3 | -2 | 2 | 5 | 7 | 13 | 12 |
| n-2=i=4 | -2 | 2 | 5 | 7 | 12 | 13 |

## 2 Implémentation du tri en python

**Exercice 32** (En vue du tri sélection)

1. Ecrire une fonction `minimum(T,j)` qui recherche la valeur minimale du sous-tableau `T[j..n]` où `n=len(T)-1` et qui renvoie l'indice de cette valeur minimale.
2. Ecrire une fonction `permute(T,i,j)` qui permute les valeurs d'indice `i` et `j` du tableau, si `i` et `j` sont différents.
3. En déduire une fonction `tri_selection(T)` qui trie en place le tableau `T` (cela signifie qu'on n'a pas besoin d'un nouveau tableau et donc pas besoin de mémoire supplémentaire pour faire ce tri).

```
def minimum(T,j) :
    pos, valmin = j, T[j] # initialisation de la valeur minimale à T[j] et non à 0
    for k in range( j+1, len(T) ) :
        if T[k] < valmin :
            pos, valmin = k, T[k]
    return pos
```

```
def permute(T, i, j) :
    if i != j :
        T[i], T[j] = T[j], T[i]
```

```
def tri_selection(T) :
    for j in range( len(T)-1 ) :
        i = minimum(T, j)
        permute(T, i, j)
```

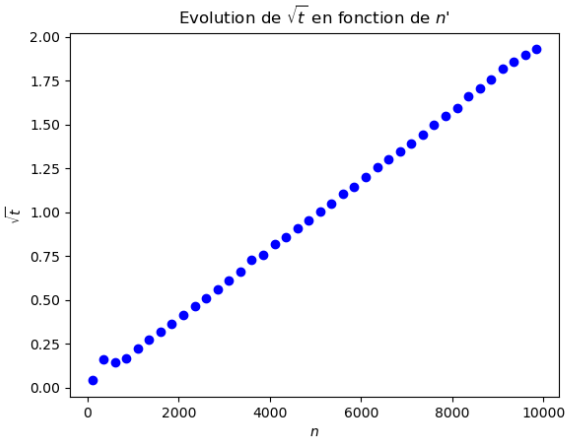
## 3 Taux de croissance

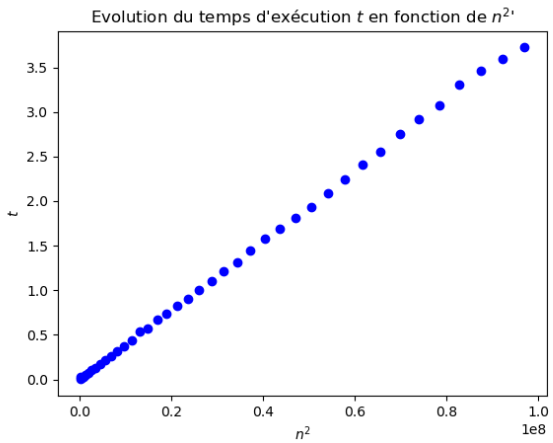
On veut calculer le taux de croissance de cet algorithme dans le pire des cas. On regarde donc le nombre de comparaison qui sont faites : à chaque fois qu'il y a une comparaison, on a représenté la case en bleu dans l'exemple. Le nombre de comparaison est donc environ de

$$\frac{n \times (n-2)}{2} = \frac{n^2 - 2n}{2}$$

On a donc un algorithme dont le taux de croissance est en $\mathcal{O}(n^2)$ : on dit qu'on a un algorithme **quadratique**.

Ansi, le temps d'exécution d'un tri par sélection est de la forme $t = kn^2$ quand $n$ est grand.
Si cela est vrai, alors $\sqrt{t} = \sqrt{k} \times n$ Si cela est vrai, alors $t = kn^2$



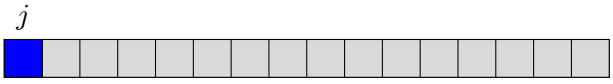
# 4 Peut-on garantir le résultat ?

Peut-on garantir que le résultat sera toujours un tableau trié correctement (par valeur croissante) ?
On va faire une démonstration par graphique !

### Itération 1

début itération

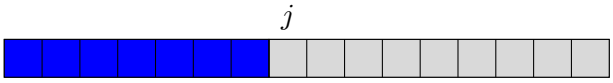
On sélectionne le minimum de la partie grisée et s'il n'est pas en première position, on le met en première position.



fin itération

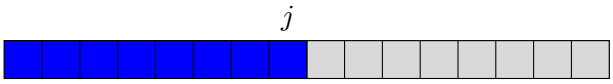
La première case correspond bien à la plus petite valeur du tableau `T` et la zone en couleur bleue correspond à la zone triée du tableau `T`.

### Itération quelconque



début itération

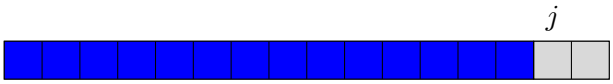
On recherche la valeur minimale de la partie grisée du tableau, valeur qui est forcément plus grande que n'importe quelle valeur de la zone bleue. Quand on la trouve, on la positionne en position `j`. Ainsi, la zone bleue est triée et elle est suivie immédiatement de la plus petite valeur de la zone grisée du tableau.



fin itération

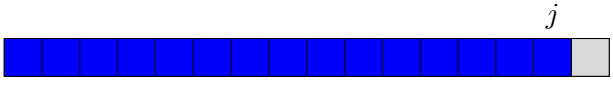
La nouvelle zone bleue est donc intégralement triée correctement.

### Dernière itération



début itération

On recherche la valeur minimale de la partie grisée du tableau, valeur qui est forcément plus grande que n'importe quelle valeur de la zone bleue. Quand on la trouve, on la positionne en position `j`. Ainsi, la zone bleue est triée et elle est suivie immédiatement de la plus petite valeur de la zone grisée du tableau.



fin itération

La nouvelle zone bleue est donc intégralement triée correctement. Et il se trouve qu'il ne reste plus qu'une seule valeur dans la zone grisée, c'est donc la valeur minimale de la zone grisée, et elle est plus grande que toutes celles de la zone bleue, le tableau `T` est donc trié !

```python
def tri_selection(tab:list) -> None :
    "tri par sélection du minimum (en place) du tableau tab"
    def permute(tab:list, i:int, j:int) -> None :
        "permute les éléments du tableau tab d'index i et j"
        if i != j :
            tab[i], tab[j] = tab[j], tab[i]
    def minimum(tab:list,j:int) -> int :
        "renvoie l'indice du minimum du sous-tableau tab[j..n]"
        pos, valmin = j, tab[j] # initialisation de la valeur minimale à tab[j] et non à 0
        for k in range( j+1, len(tab) ) :
            if tab[k] < valmin :
                pos, valmin = k, tab[k]
        return pos
    # tri sélection proprement dit
    for j in range( len(tab)-1 ) :
        i = minimum(tab, j)
        permute(tab, i, j)
```

```
import timeit
from math import sqrt
from random import randint
import matplotlib.pyplot as plt

def minimum(T,j) :
    pos, valmin = j, T[j] # initialisation de la valeur minimale à T[j] et non à 0
    for k in range( j+1, len(T) ) :
        if T[k] < valmin :
            pos, valmin = k, T[k]
    return pos
def permute(T, i, j) :
    if i != j :
        T[i], T[j] = T[j], T[i]
def tri_selection(T) :
    for j in range( len(T)-1 ) :
        i = minimum(T, j)
        permute(T, i, j)

valx, valy = [], []
valx2, valy2 = [], []
for n in range(100,10000,250) :
    T = list(range(n))[::-1]
    t1 = timeit.default_timer()
    tri_selection(T)
    t2 = timeit.default_timer()
    valx.append(n**2)
    valy.append(t2-t1)
    valx2.append(n)
    valy2.append(sqrt(t2-t1))

plt.figure()
plt.plot(valx, valy, 'ob')
plt.title("Evolution du temps d'exécution $t$ en fonction de $n^2$'")
plt.xlabel("$n^2$")
plt.ylabel("$t$")
#
plt.figure()
plt.plot(valx2, valy2, 'ob')
plt.title("Evolution de $\sqrt{t}$ en fonction de $n$'")
plt.xlabel("$n$")
plt.ylabel("$\sqrt{t}$")
plt.show()
```