Tri par sélection – Tri par insertion

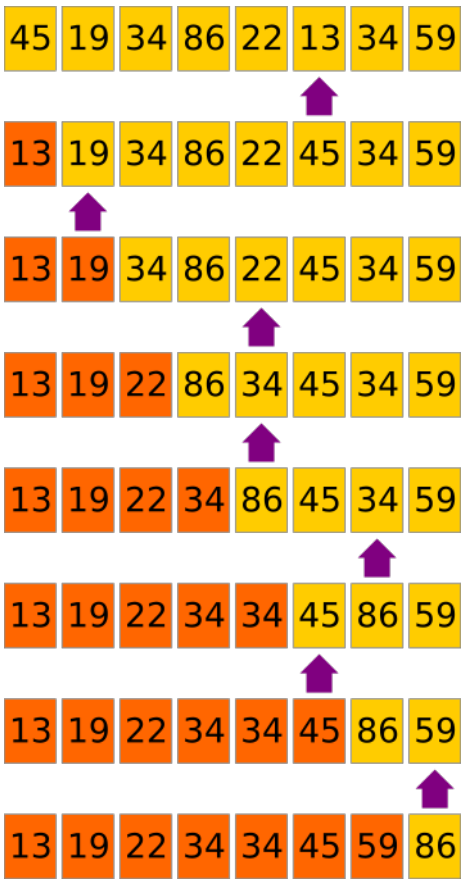
On veut trier un tableau par ordre croissant. On se limite au tri par sélection et tri par insertion, mais il existe d'autres méthodes. On peut adapter les tris à n'importe quel type d'ordre.

Tri par sélection

À mesure de la progression de l'algorithme, une zone triée se constitue au début du tableau. Une boucle **sélectionne** le plus petit élément parmi les non-triés et le place à la suite des triés. À chaque répétition, la zone triée augmente d'un élément si bien que tout le tableau finit par être trié.

Exemple ci-contre avec
`[45, 19, 34, 86, 22, 13, 34]`

Entrées : T : tableau à trier
1 $n \leftarrow$ longueur de T
2 **pour** i *allant de* 0 *à* $n-2$ **faire**
3 | *les éléments* $[:i]$ *sont triés*
4 | *recherche du plus petit dans* $[i:]$
5 | $i_{MIN} \leftarrow i$
6 | **pour** j *allant de* $i+1$ *à* $n-1$ **faire**
7 | | **si** $T[j] < T[i_{MIN}]$ **alors**
8 | | | $i_{MIN} \leftarrow j$
9 | | **fin**
10 | **fin**
11 | *permuter le plus petit de* $[i:]$ *avec* $[i]$
12 | *ce qui le place au bout des triés*
13 | **si** $i_{MIN} \neq i$ **alors**
14 | | échanger $T[i_{MIN}]$ et $T[i]$
15 | **fin**
16 **fin**



Il s'agit d'un **tri en place**, c'est à dire que le tableau soumis au tri est lui-même modifié, ce n'est pas une copie triée qui est produite. Au besoin, on pourrait créer une copie du tableau et trier cette copie.

Complexité, correction

- Nous n'avons que des boucles `pour`, donc l'algorithme **termine**.
- La boucle `pour` en ligne 2 est exécutée $n-1$ fois. La boucle `pour` en ligne 6 dépend de la taille de la zone à trier. Lors du premier passage, elle fait n répétitions, puis $n-1$ puis $n-2$ puis... On a donc une complexité en $n+(n-1)+\cdots 2$ soit quelque chose en $\frac{n(n+1)}{2}$.
  L'ordre de grandeur asymptotique (pour n grand) est donc n^2. Cela signifie qu'un tableau $10\times$ plus grand prendra $100\times$ plus de temps à être trié.
- Voyons la correction.
  - Premier point, nous voulons vérifier que la zone triée ne contient jamais d'élément plus grand qu'un élément de la zone à trier. C'est vrai au début puisque la zone triée est vide. C'est vrai ensuite car quand on ajoute un élément de la zone non triée, c'est le plus petit d'entre eux qui n'est donc pas plus grand que les autres.
  - Deuxième point, la zone triée est toujours triée. C'est vrai puisque d'après le point précédent, un élément ajouté à la zone triée est toujours au moins aussi grand que tous ceux déjà dans la zone triée. En le mettant à la suite de la zone triée on respecte l'ordre.
  - Dernier point, à chaque boucle, la zone triée augmente de 1 item. Donc la zone triée fini par occuper la totalité du tableau.

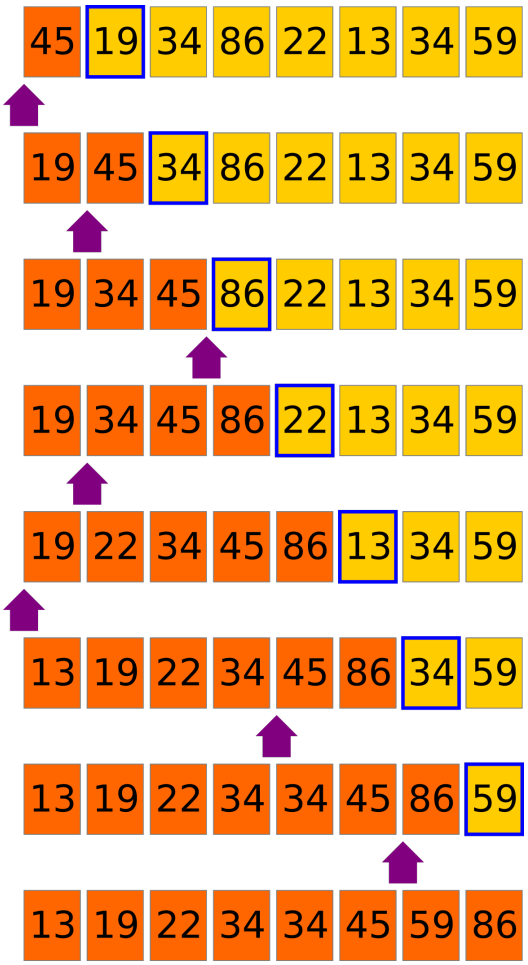
Tri par insertion

À mesure de la progression de l'algorithme, une zone triée se constitue au début du tableau. Une boucle prend le premier élément qui vient dans la zone à trier et l'**insère** à sa place parmi les triés.

Exemple ci-contre avec
`[45, 19, 34, 86, 22, 13, 34]`

```
   Entrées : T : tableau à trier
1  n ← longueur de T
2  pour i allant de 1 à n − 1 faire
3  |  les éléments [: i] sont triés
4  |  on prend celui qui suit : T[i]
5  |  x ← T[i]
6  |  pour insérer x = T[i] à sa place parmi les
   |    triés
7  |  il faut décaler ceux plus grands que lui
8  |  j ← i
9  |  tant que j > 0 et T[j − 1] > x faire
10 |  |  T[j] ← T[j − 1]
11 |  |  j ← j − 1
12 |  fin
13 |  j est la position d'insertion
14 |  T[j] ← x
15 fin
```

Il s'agit là encore d'un **tri en place**.



Complexité, correction

- La boucle `pour` termine. Dans la boucle `tant que`, j diminue de 1 à chaque répétition et la condition est $j > 0$. Cette boucle termine donc. Donc l'algorithme **termine**.
- Cette fois c'est la recherche du point d'insertion qui prend le plus de temps. Cette recherche devient plus longue à mesure que la zone triée grandit. Au pire il faudra compter $1+2+\cdots+(n-1)$ ce qui produit le même ordre que pour le tri par sélection.
- La correction semble plus évidente ici puisque l'on insère chaque nouvel item à sa place dans la zone triée.