Tris par sélection et par insertion

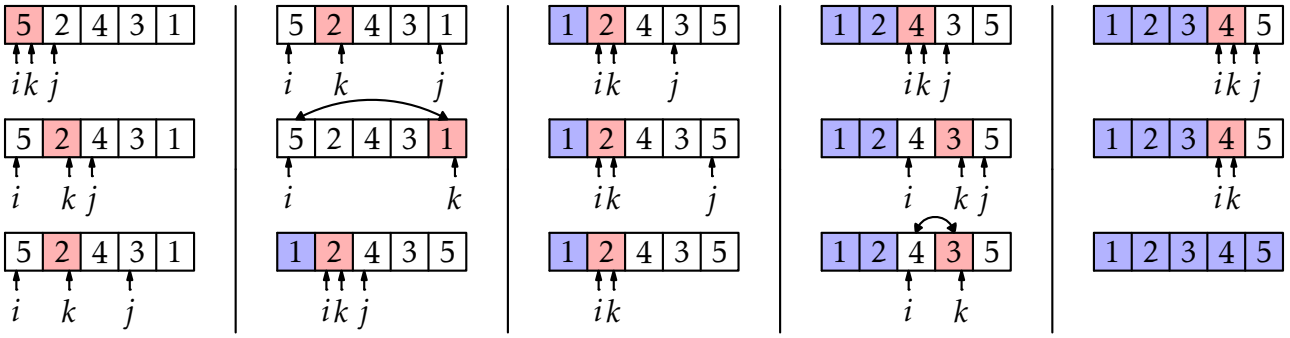
Le tri par sélection

L'algorithme de tri à bulles, est simple à programmer, mais peu efficace car il effectue un grand nombre de comparaisons et d'échanges.

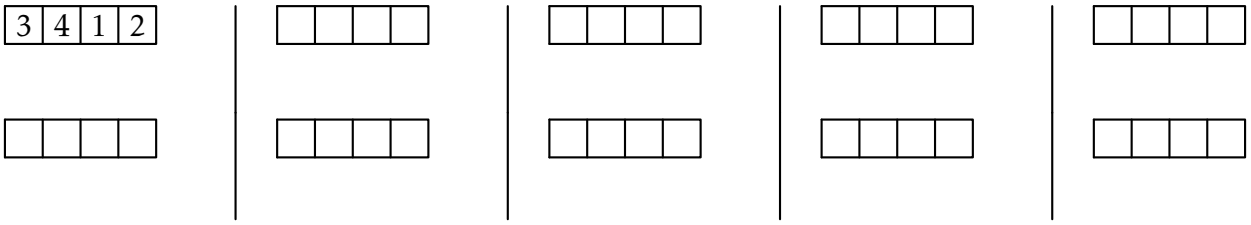
L'algorithme de **tri par sélection** lui, cherche à minimiser le nombre d'échanges nécessaires. L'idée est de chercher le plus petit élément et de le mettre en première position, puis le deuxième plus petit et le mettre en deuxième position et ainsi de suite.
L'algorithme est ci-contre. La boucle externe est composée de 2 parties. Tout d'abord, on cherche la position k du plus petit élément entre i et n. Une fois trouvé, on inverse les éléments en position i et k. Voici un exemple d'exécution :

Algorithme du tri par sélection :

Pour i allant de 1 à $n-1$ **:**
    $k \leftarrow i$
    Pour j allant de $i+1$ à n **:**
        Si $T[j] < T[k]$ **:**
            $k \leftarrow j$
    Échanger($T[i]$, $T[k]$)



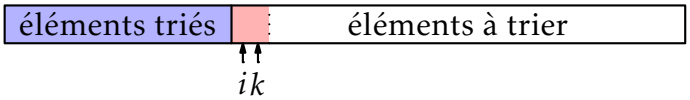
Exercice 1 : Donner les étapes correspondant au tri du tableau ci-dessous :



Correction et complexité du tri par sélection

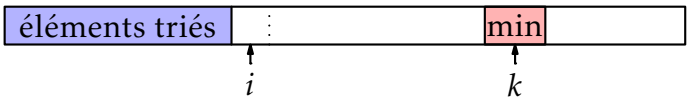
La terminaison de l'algorithme ne pose pas de problème puisque les boucles sont toutes les deux bornées.
Pour la correction, on remarque qu'avant chaque tour de la boucle externe, les éléments en position 1 à $i-1$ sont triés et en bonne position. Ce sera notre invariant de boucle.

État du tableau au début d'un tour de la boucle externe :



Pour démontrer que cette propriété est bien un invariant de boucle, il faut également remarquer qu'à la fin de l'exécution de la boucle interne, le minimum des éléments entre i et n est en position k. Lorsqu'on permute les éléments en position i et k, on s'assure que l'élément en position i est bien placé pour la prochaine itération.

État du tableau à la fin de la boucle interne :



Pour la complexité, comme pour le tri à bulles, le nombre de tests à effectuer est :

$$(n-1)+(n-2)+\ldots+1=\frac{n(n-1)}{2}$$

La complexité de cet algorithme de tri est donc quadratique. Par contre, le nombre d'échanges, lui est toujours $n-1$. Il y a donc un nombre linéaire d'échanges. Cet algorithme de tri est avantageux si l'échange de valeurs est coûteux en temps ou en mémoire.
Il est également possible de l'améliorer en rajoutant un test pour s'assurer que $i \neq k$ avant de faire l'échange. Ainsi, si le minimum est déjà en bonne position, pas besoin de faire d'échange. Dans ce cas, le nombre d'échanges va de 0 en $n-1$ en fonction du tableau.

Exercice 2 : On suppose qu'on a rajouté l'optimisation pour éviter de faire l'échange si $i = k$. Donner des exemples de tableaux nécessitant le nombre maximum d'échanges, avec 2, 3, 4 et n éléments.

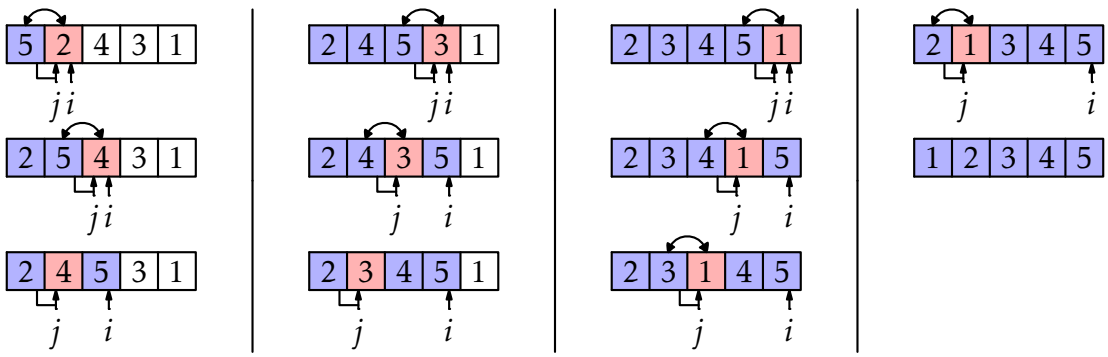
Le tri par insertion

Le **tri par insertion** a pour but de minimiser le nombre de comparaisons nécessaires. L'idée consiste à parcourir la liste de gauche à droite en plaçant le nouvel élément au bon endroit parmi ceux de gauche. En gros, cela revient à trier les éléments au fur et à mesure qu'on les voit.

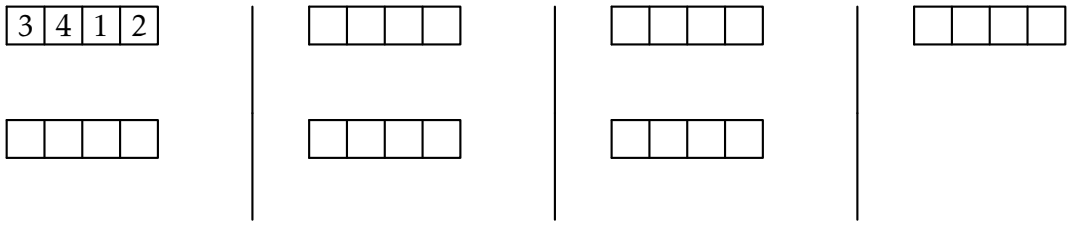
Algorithme du tri par insertion :

Pour i allant de 2 à n **:**
    $j \leftarrow i$
    Tant que $j > 1$ et $T[j-1] > T[j]$ **:**
        Échanger($T[j-1]$, $T[j]$)
        $j \leftarrow j-1$

Contrairement aux tris à bulles ou à sélection, les éléments déjà parcourus ne sont pas forcément dans leur position finale. Les prochains éléments viendront s'intercaler au fur et à mesure.
Voici un exemple d'utilisation :



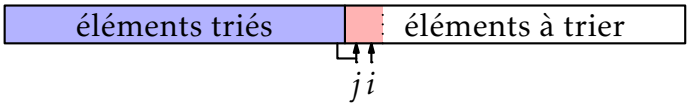
Exercice 3 : Donner les étapes correspondant au tri du tableau ci-dessous :



Correction et complexité du tri par insertion

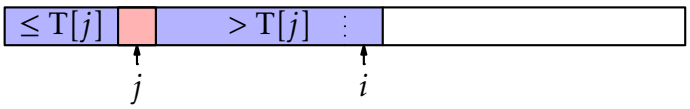
Puisqu'il y a une boucle non bornée, il faut s'assurer qu'elle termine forcément. Dans le cas du tri par sélection, puisque j décroit à chaque itération et que la boucle s'arrête dès qu'il est inférieur ou égal à 1, la boucle se termine forcément.
Pour la correction, il faut remarquer qu'avant chaque itération de la boucle externe, tous les éléments en position 1 à $i-1$ sont classés dans l'ordre croissant, même s'ils ne sont pas forcément en position définitive.

État du tableau au début d'un tour de la boucle externe :



Pour s'assurer que cette propriété est bien un invariant de boucle, il faut s'assurer que la boucle interne met l'élément en position i au bon endroit parmi les premiers éléments. Si le début du tableau est bien trié, cet élément se déplacera vers la droite en s'arrêtant soit en première position si c'est le minimum, soit après le premier élément qui est inférieur. Il est lui-même inférieur à tous les éléments à droite qui sont déjà triés. La liste des i premiers éléments est donc bien triée.

État du tableau à la fin de la boucle interne :



La complexité de cet algorithme est beaucoup plus difficile à estimer que pour les autres, à cause de la boucle bon bornée. Dans le pire des cas, on peut remarquer qu'il y a, encore une fois, $\frac{n(n-1)}{2}$ tests et échanges. Pour déterminer précisément ce nombre, il suffit de compter pour chaque élément du tableau combien sont supérieur et se trouvent avant lui. Le nombre d'échanges nécessaires est la somme de ces nombres. Le nombre de tests est lui égal à la somme du nombre d'échanges et de $n-1$.
Au minimum, si la liste est déjà triée, il y a donc $n-1$ tests et aucun échanges. En fait, si le tableau est "presque trié", le coût est linéaire, ce qui en fait un des meilleurs algorithmes de tris dans ces cas là.
La complexité de cet algorithme varie donc entre linéaire et quadratique selon les cas.

Exercice 4 :

1) Déterminer le nombre d'échanges à faire pour le tableau suivant : `[7, 9, 1, 15, 3, 47]`
2) Déterminer le nombre d'échanges à faire pour le tableau suivant : `[87, 11, 7, 4, 8, 22]`
3) Quel type de tableaux correspond au pire des cas pour cet algorithme ?