ALGORITHMES DE TRIS

Situation La recherche par dichotomie d'un élément dans un tableau est plus rapide qu'une recherche classique mais celle-ci n'est possible que si ce tableau était trié.
Il est donc naturel de se demander s'il existe une procédure efficace pour trier des données.

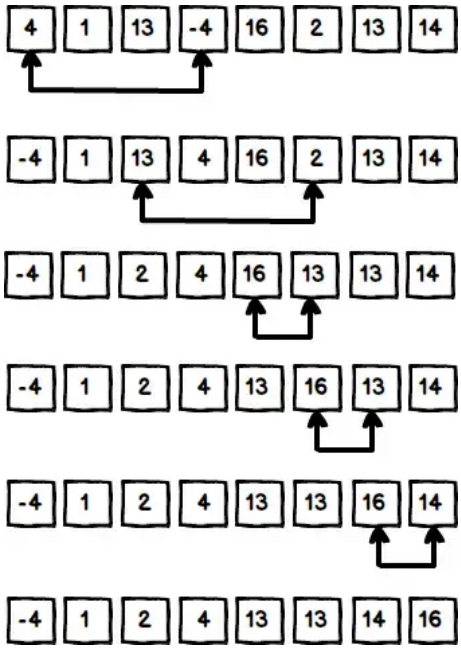
On suppose donc que l'on cherche à trier un tableau de N entiers par ordre croissant.

Pour pouvoir comparer l'efficacité des algorithmes, il faut calculer leurs complexités. Il existe deux types de complexité : la complexité spatiale (espace en mémoire) et la complexité temporelle. On choisira de s'intéresser à la complexité temporelle c'est à dire de comparer les nombres d'affectations et de comparaison (opération plus longue) que nécessite le tri.

• Tri par sélection ou tri naïf

•• Principe

On recherche le minimum de la liste, et le placer en début de liste puis recommencer sur la suite du tableau.



•• Exemple à compléter

❝ **Notation** : *Soit T un tableau, on note T[0 :i] la portion du tableau d'indice 0 à i-1*

On cherche à trier le tableau `T=[7,12,5,8,9,6,13,3]`

Etape	Action	Etat du tableau après modification
Init.		7 12 5 8 9 6 13 3
0	on cherche le minimum sur T[0 :8] et on l'échange avec T[0]	**3** 12 5 8 9 6 13 **7**
1	on cherche le minimum sur T[1 :8] et on l'échange avec T[1]	3 **5** **12** 8 9 6 13 7
2	on cherche le minimum sur ... et on l'échange avec ...	...
3	on cherche le minimum sur ... et on l'échange avec ...	...
4	on cherche le minimum sur ... et on l'échange avec ...	...
5	on cherche le minimum sur ... et on l'échange avec ...	...
7	on cherche le minimum sur ... et on l'échange avec ...	...

•• Algorithme

```
procédure tri_selection(tableau t)
    n = longueur(t)
    pour i de 0 à n - 2
        min = i
        pour j de i + 1 à n - 1
            si t[j] < t[min], alors min = j
        fin pour
        si min != i, alors échanger t[i] et t[min]
    fin pour
fin procédure
```

•• Complexité temporelle

– **Nombre de comparaisons :**
Pour rechercher le premier minimum on exécute $N-1$ comparaisons, pour le second $N-2$, ...

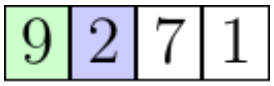
Le nombre total de comparaisons est de $(N-1)+(N-2)+...+1=\frac{N(N-1)}{2}$ donc en $O(N^2)$.

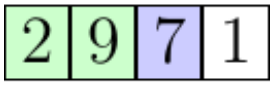
– **Nombre d'affectations :**
pour chaque échange, on execute 3 affectations donc 3N, donc $O(N)$

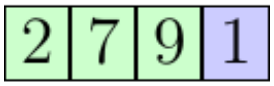
• Tri par insertion

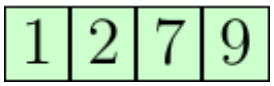
•• Principe

Le tri par insertion considère chaque élément du tableau et l'insère à la bonne place parmi les éléments déjà triés. Ainsi, au moment où on considère un élément, les éléments qui le précèdent sont déjà triés, tandis que les éléments qui le suivent ne sont pas encore triés.









•• Exemple à compléter

Insérer successivement chaque élément T[i] dans la portion du tableau T[0 :i] déjà triée.
Exemple sur T=[4, 12, 5, 8, 9, 6, 13, 3].

Etape	Action	Tableau
Init.	Le tableau T[0 :1]=[4] est déjà trié.	**4** 12 5 8 9 6 13 3
1	on cherche placer T[1]=12 dans T[0 :1]=[4]	**4 12** 5 8 9 6 13 3
2	on cherche placer T[2]=5 dans T[0 :2]=[4, 12]	**4 5 12** 8 9 6 13 3
3	on cherche placer ..... dans	
4	on cherche placer ..... dans	
5	on cherche placer ..... dans	
6	on cherche placer ..... dans	
7	on cherche placer ..... dans	

•• Algorithme

```
procédure tri_insertion(tableau T)
n = taille de T
pour i de 1 à n - 1
    # mémoriser T[i] dans x
    val = T[i]

    % décaler les éléments T[0]..T[i-1] qui sont plus grands que x, en partant de T[i-1]
    j = i
    tant que j > 0 et T[j - 1] > val
        T[j] = T[j - 1]
        j = j - 1
    % placer x dans le "trou" laissé par le décalage
        T[j] = val
    fin tant que
fin procédure
```

•• Compléxité

Une fois de plus, le coût en mémoire de ce tri est constant.
Évaluons sa complexité temporelle :

- pour l'étape 1, on effectue 1 comparaison ;
- pour l'étape 2, on effectue entre 1 comparaison (si le tableau est déjà trié), et 2 comparaisons ...
- pour l'étape 3, on effectue entre 1 comparaison (si le tableau est déjà trié), et 3 comparaisons ...
- ...
- pour le dernier placement, on exécute au mieux 1 comparaison (si le tableau est déjà trié), au pire N-1 comparaisons

Le nombre total de comparaisons est donc compris entre $N-1$ comparaisons donc $O(N)$ (temps linéaire) et $1+2+...+N-1=\frac{N(N-1)}{2}$ comparaisons donc $O(N^2)$

- si le tableau était déjà classé, le temps est linéaire $O(n)$
- si le tableau était rangé par ordre décroissant le temps est quadratique $O(n)$.