Dans ce chapitre, on désire trier éléments d'un tableau donné dans l'ordre croissant.

1 Tri par sélection

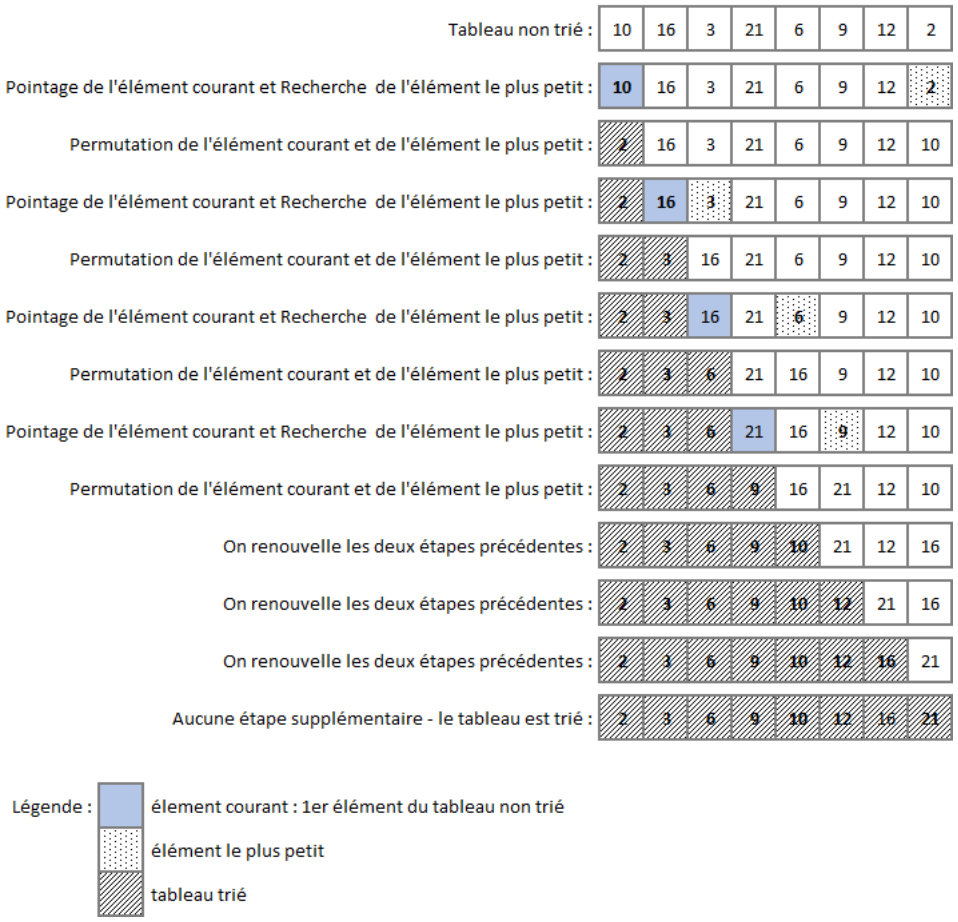
C'est la méthode de tri la plus intuitive.
Le tableau à trier est « divisé » en deux parties : la $1^{ère}$ constituée des éléments triés (initialisée avec seulement le 1^{er} élément) et la seconde constituée des éléments non triés (initialisée du $2^{ème}$ au dernier élément)

- Le premier élément constitue, à lui tout seul, un tableau trié de longueur 1.
- On recherche le plus petit élément dans la partie non triée du tableau et on l'échange avec le dernier élément de la partie triée du tableau. À la première étape, le plus petit élément du tableau est donc mis au début. On obtient alors un sous-tableau trié de longueur 1.
- On augmente de 1 la taille du sous-tableau trié, en y incluant le deuxième élément du tableau. On recherche le plus petit élément dans la partie non triée du tableau, en commençant par le troisième élément, et on l'échange avec le dernier élément de la partie triée du tableau. On obtient alors un sous-tableau trié de longueur 2. Et ainsi de suite...
- Le principe du tri par sélection est donc d'échanger à la n^{ieme} itération le dernier élément de la partie triée du tableau avec le plus petit élément de la partie non triée du tableau.

↪ ALGORITHME DU TRI PAR SÉLECTION

```
1  TriSelection (S : Tab)
   Entrée :
       S : tableau non trié d'entiers
   Sorties :
       S : tableau trié
   Variables locales :
       i : entier - compteur pour boucle
       j : entier - compteur pour boucle
       indice : entier - indice de l'élément le plus petit
2  début
3      pour i=1 à S.Fin-1 faire
4          - initialisation de l'indice de l'élément le plus petit avec l'indice du dernier élément de la
             partie du tableau trié
5          indice ← i;
6          - recherche de l'élément le plus petit de la partie du tableau non triée
7          pour j=i+1 à S.Fin faire
8              si S[j]<=S[indice] alors
9                  - Mise à jour de l'indice de l'élément le plus petit de la partie du tableau non trié
10                 indice ← j;
11             fin
12         fin
13         - permutation de cet élément le plus petit avec le 1^er élément de la partie du tableau non triée
             qui devient le dernier élément de la partie du tableau trié.
14         S[i] ↔ S[indice]
15     fin
16 fin
```

↪ UN EXEMPLE POUR ILLUSTRER



↪ Un lien pour voir fonctionner le processus : `http://lwh.free.fr/pages/algo/tri/tri_selection.html`

↪ LA COMPLEXITÉ DE L'ALGORITHME

1. Combien de comparaisons et d'échanges vont être effectués dans tous les cas ?

2. Si le tableau contient 20 millions de valeurs, combien y aura-t-il de comparaisons et d'échanges dans tous les cas ?

ADMIS

La complexité de la méthode de tri par sélection d'un tableau contenant n éléments est $O(n^2)$.

2 Tri par insertion

Le tri par insertion le tri « naturel » du joueur de cartes.
Le tableau à trier est « divisé » en deux parties : la $1^{ère}$ constituée des éléments triés (initialisée avec seulement le 1^{er} élément du tableau) et la seconde partie constituée des éléments non triés du tableau (initialisée du $2^{ème}$ au dernier élément). On procède comme si les éléments d'un tableau à trier étaient donnés un par un.

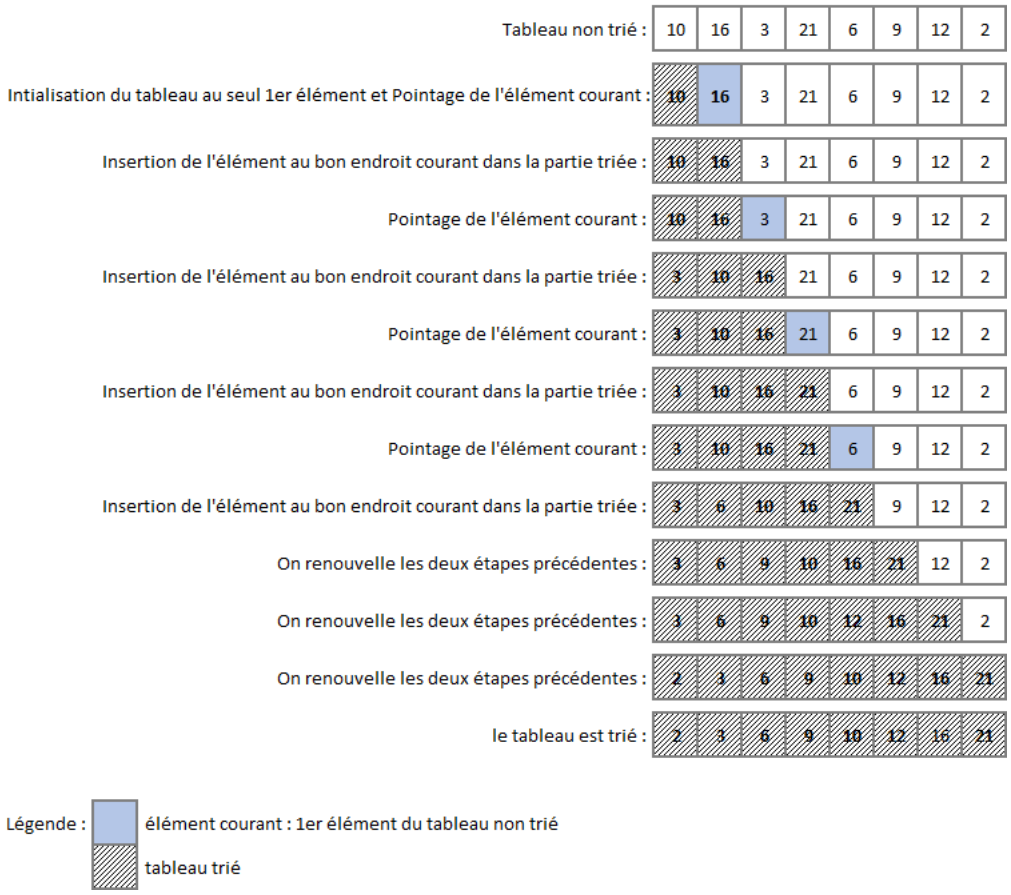
- Le premier élément constitue, à lui tout seul, un tableau trié de longueur 1.
- On range ensuite le second élément « à sa place » pour constituer un tableau trié de longueur 2, puis on range le troisième élément pour avoir une tableau trié de longueur 3 et ainsi de suite...
- Le principe du tri par insertion est donc d'insérer à la n^{ieme} itération le n^{ieme} élément à la « bonne » place.

↪ ALGORITHME DU TRI PAR INSERTION

```
1  TriInsertion (S : Tab)
   Entrée :
       S : tableau non trié d'entiers
   Sorties :
       S : tableau trié

   Variables locales :
       i : entier - compteur pour boucle
       j : entier - compteur pour boucle
       valeur : entier - valeur de l'élément à déplacer par insertion
       indice : entier - indice futur de l'élément à déplacer par insertion
2  début
3      - le tableau est constitué de deux parties : la 1ère constituée des éléments triés
         (initialisée avec seulement le 1er élément du tableau) et la seconde partie constituée
         des éléments non triés du tableau (initialisée du 2ème au dernier élément)
4      pour i=1 à S.Fin-1 faire
5          - mémorisation du 1er élément de la partie du tableau non trié que nous allons
             déplacer
6          valeur ← S[i + 1]
7          - recherche de l'indice que doit prendre ce 1er élément dans la partie du tableau trié
8          indice ← 1
9          tant que S[indice]<valeur faire
10             indice ← indice + 1
11         fin
12         - décalage des éléments compris entre le dernier élément de la partie du tableau trié
             et l'emplacement trouvé précédemment (parcours décroissant)
13         pour j=i à indice en décroissant faire
14             S[j + 1] ← S[j]
15         fin
16         - Déplacement par insertion du 1er élément de la partie du tableau non trié à l'indice
             trouvé ... qui devient un élément trié
17         S[indice] ← valeur
18     fin
19 fin
```

↪ UN EXEMPLE POUR ILLUSTRER



↪ Un lien pour voir fonctionner le processus : `http://lwh.free.fr/pages/algo/tri/tri_insertion.html`

↪ LA COMPLEXITÉ DE L'ALGORITHME

1. Combien de comparaisons et d'échanges vont être effectués dans le meilleur cas ?

2. Combien de comparaisons et d'échanges vont être effectués dans le pire cas ?

3. Si le tableau contient 20 millions de valeurs, combien y aura-t-il de comparaisons et d'échanges dans le pire cas ?

ADMIS

La complexité de la méthode de tri par insertion d'un tableau contenant n éléments est $O(n^2)$.