ALGORITHMES DE TRI

Les premières suites d'instructions permettant d'effectuer un tri ont été étudiées dès les années 1940 sur les premiers ordinateurs. Durant cette période, l'américaine Betty Holberton était une de six programmatrices de l'ENIAC, le premier ordinateur entièrement électronique. C'est à ce moment qu'elle développe le premier algorithme de tri. Elle consacra ensuite sa carrière au développement des langages Fortran et Cobol

1 – TRIER DES DONNEES

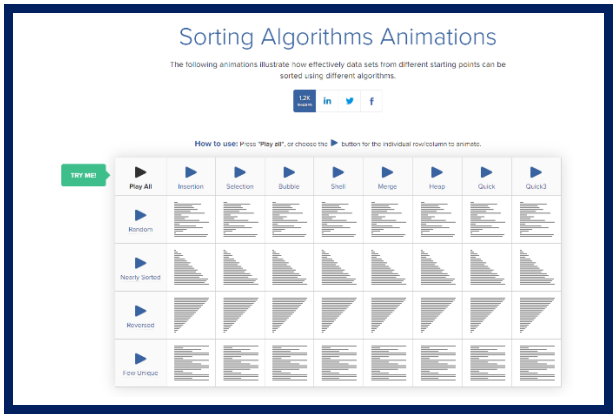
Trier des données cela consiste à les ranger dans un ordre prédéfini. Il est possible de trier des nombres dans un ordre croissant ou décroissant. L'ensemble des nombres {3, 4, 2, 5, 1} triés dans l'ordre croissant nous donne l'ensemble {1, 2, 3, 4, 5}. Mais il est également possible de trier des lettres ou des mots dans l'ordre alphabétique par exemple : {a, d, e, c, b} --> {a, b, c, d, e}.

Le tri est historiquement un problème majeur en informatique pour plusieurs raisons :

- On a souvent besoin de trier des données (notes, noms, photos ...).
- Les algorithmes de tri sont des sous-programmes indispensables à de nombreuses applications (gestionnaires de fenêtres graphiques ...) ou programmes (compilateurs).
- La diversité des algorithmes de tri qui ont été développés, présente un intérêt pédagogique dans l'apprentissage de l'algorithmique.

Il existe un certain nombre d'algorithmes de tri dont les plus connus sont, le **tri par insertion**, le **tri sélection**, le **tri bulle** (Bubble sort), le **tri shell**, le **tri fusion** (Merge sort), le **tri par tas** (Heap sort), le **tri rapide** (Quick sort)...

Vous trouverez, sur le site ci-contre, des animations vous permettant de comparer l'efficacité de ces algorithmes dans le cas de 4 jeux de données différents.



> **Question 1** : Lancer l'animation dans le cas où le jeu de données est aléatoire. Evaluer quel est l'algorithme le plus efficace et celui qui est le moins efficace. Réaliser le même travail dans le cas où les données sont presque triées et lorsque les données sont inversées.

Dans la suite de l'activité, nous nous intéresserons à deux types d'algorithme de tri :

- Algorithme de **tri sélection**.
- Algorithme de **tri par insertion**.

Pour terminer, pour aller plus loin, nous étudierons le **tri bulle**.

2 – ALGORITHMIQUE DE TRI SELECTION

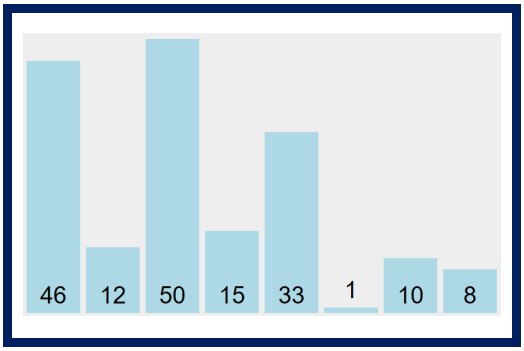
2.1 – Présentation de l'algorithme de tri sélection

Le tri sélection (appelé aussi tri par extraction) est un **algorithme de tri par comparaisons**. Cet algorithme est simple, mais considéré comme peu efficace.

Le principe de l'algorithme de tri sélection est le suivant :

- rechercher le plus petit élément du tableau, et l'échanger avec l'élément d'indice 0 ;
- rechercher le second plus petit élément du tableau, et l'échanger avec l'élément d'indice 1 ;
- continuer de cette façon jusqu'à ce que le tableau soit entièrement trié.

L'animation ci-contre vous présente le fonctionnement du tri sélection.



La vidéo ci-contre vous présente également le fonctionnement du tri sélection :

2.2 – Programmation de l'algorithme de tri sélection

L'algorithme tri sélection est le suivant :

```
VARIABLES
tab : tableau d'entiers
n : entier
temp : entier
i,j : entier
DEBUT
   n <-- taille tableau tab
   Pour i de 0 à n-2
       min <-- i
       Pour j de i+1 à n-1
          Si tab[j] < tab[min]
             min <-- j
       Si min ≠ i
          temp <-- tab[i]
          tab[i] <-- tab[min]
          tab[min] <-- temp
FIN
```

Question 2 : Editer la fonction **`tri_selection`**. Tester son fonctionnement sur le tableau suivant :

```
t = [1, 10, 3, 5, 7, 30, 15, 8, 9, 12, 22, 5, 10, 5, 18]
```

Question 3 : Analyser le fonctionnement du programme sur Python Tutor.

2.3 – Complexité de l'algorithme de tri sélection

Quel que soit l'ordre du tableau initial, le **nombre de comparaisons reste le même**, **ainsi que le nombre d'échanges**.

L'algorithme est constitué de deux boucles for imbriquées. À chaque itération **i**, **j** prend les valeurs de $\mathbf{i + 1}$, à $\mathbf{n - 1}$ c'est-à-dire $\mathbf{n - i + 1}$ valeurs. Une comparaison est effectuée pour chaque valeur de **j**. Donc pour chaque valeur **i**, il y a $\mathbf{n - i + 1}$ **comparaisons**.

Donc au total, cela nous fait $(\mathbf{n - 1}) + (\mathbf{n - 2}) + (\mathbf{n - 3}) + \ldots + \mathbf{2} + \mathbf{1}$ **comparaisons**.

$$(\mathbf{n}-\mathbf{1})+(\mathbf{n}-\mathbf{2})+(\mathbf{n}-\mathbf{3})+\ldots+\mathbf{2}+\mathbf{1}=\sum_{\mathbf{i}=\mathbf{1}}^{\mathbf{n}-\mathbf{1}}\mathbf{n}-\mathbf{i}=\frac{\mathbf{n}(\mathbf{n}-\mathbf{1})}{\mathbf{2}}$$

Dans le **pire des cas** (entiers du tableau sont initialement donnés dans l'ordre décroissant) le **nombre d'échanges** est de $\mathbf{n - 1}$.

Le coût de l'algorithme de tri sélection est de l'**ordre** $\mathbf{n^2}$. Cet algorithme est donc de **complexité quadratique**.

2.4 – Terminaison de l'algorithme de tri sélection

L'algorithme est constitué de deux boucles imbriquées. Le nombre de passages dans ces deux boucles est parfaitement déterminé et il est évidement fini. La **terminaison est donc vérifiée**.

2.5 – Conclusion

L'algorithme de tri sélection est un **algorithme simple à programmer mais peu efficace**. La complexité de cet algorithme est de l'ordre $\mathbf{n^2}$, c'est-à-dire une **complexité quadratique**.

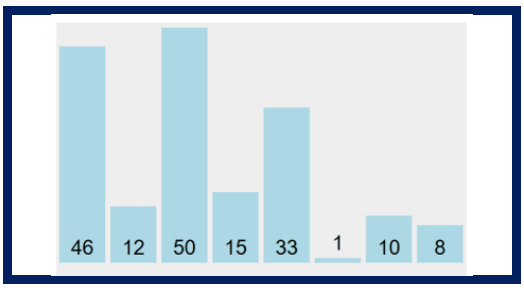
3 – ALGORITHMIQUE DE TRI PAR INSERTION

3.1 – Présentation de l'algorithme de tri par insertion

C'est le tri du joueur de cartes. On fait comme si les éléments à trier étaient donnés un par un, le premier élément >constituant, à lui tout seul, un tableau trié de longueur 1. On range ensuite le second élément pour constituer un tableau trié de longueur 2, puis on range le troisième élément pour avoir un tableau trié de longueur 3 et ainsi de suite...

Le principe du tri par insertion est donc d'insérer à la nième itération le nième élément à la bonne place.

L'animation ci-contre vous présente le fonctionnement du tri par insertion.



La vidéo ci-contre vous présente également le fonctionnement du tri sélection.

3.2 – Programmation de l'algorithme de tri par insertion

L'algorithme tri par insertion est le suivant :

```
VARIABLES
tab : tableau d'entiers
n : entier
x : entier
i,j : entier

DEBUT
   n <-- taille tableau tab
   Pour i de 0 à n-1
       x <-- tab[i]
       j <-- i
       Tant que j >0 et tab[j-1] > x
          tab[j] <-- tab[j-1]
          j <-- j - 1
       tab[j] <-- x
FIN
```

Question 4 : Editer le programme de la fonction **`tri_insertion`**. Tester son fonctionnement sur le tableau suivant :

```
t = [1, 10, 3, 5, 7, 30, 15, 8, 9, 12, 22, 5, 10, 5, 18]
```

Question 5 : Analyser le fonctionnement du programme sur Python Tutor.

3.3 – Complexité de l'algorithme de tri par insertion

Le nombre de comparaisons peut être différent selon le tableau à trier.

Dans le **meilleur des cas**, le tableau initial est trié et on effectue alors une comparaison à chaque itération. Le nombre de comparaisons est donc dans ce cas : $\mathbf{n-1}$.

Dans le **pire des cas**, les entiers du tableau sont initialement donnés dans l'ordre décroissant. Dans ce cas, on, comme pour le tri sélection, le nombre de comparaisons est de :

$$\sum_{i=1}^{n-1} n - i = \frac{n(n-1)}{2}$$

Dans ce cas le nombre d'échanges est également de :

$$\sum_{i=1}^{n-1} n - i = \frac{n(n-1)}{2}$$

Le coût de l'algorithme de tri par insertion est de l'**ordre n^2** . Cet algorithme est donc de **complexité quadratique**.

3.4 – Terminaison de l'algorithme de tri par insertion

Pour vérifier la terminaison, il nous faut étudier les deux boucles. La boucle externe est une boucle for dont **le nombre de passages dans la boucle `for` est déterminé et fini**. La boucle interne est une boucle **`while`**. Les valeurs prises par **`j`** constituent une suite décroissante dont les valeurs sont comprises entre **`i – 1` et `0`**. Le nombre de passage dans la boucle **`while`** est donc un **nombre fini**. Le **variant de boucle** est **`j – 1`**.

La **terminaison de l'algorithme est donc prouvée**.

3.5 – Conclusion

L'algorithme du tri par insertion est **simple et relativement intuitif**, même s'il a une complexité quadratique (**ordre n^2**).

Cet algorithme de tri reste toutefois très utilisé à cause de ses facultés à s'exécuter en temps quasi linéaire (**ordre n**) sur des **entrées presque triées**.

4 – TRIER D'AUTRES TYPES D'OBJETS

Question 6 : Vérifier le fonctionnement des algorithmes de tri sélection et par insertion sur des caractères et des chaînes de caractères.

```
t1=['P','R','a','B','s','d','c','L','0','l','Z','I','3','j','g','X','t','h','E','A']
```

Question 7 : Préciser comment est effectué le tri.

Question 8 : Vérifier le fonctionnement des algorithmes de tri sélection et par insertion sur des chaînes de caractères.

```
t2=['Jeanne','Marie','Paul','Jean','Léa','Rose','Pierre','Charles','Robert','Choé']
```

5 – ET PYTHON DANS TOUT CA

L'algorithme de tri implémenté dans le langage python est le l'algorithme de **tri timsort**. Il s'agit d'un algorithme de **tri efficace** dérivé du tri fusion et du tri par insertion.

Cet algorithme est utilisé par la méthode **`t.sort()`** qui permet de trier des tableaux ou par la fonction **`sorted()`** qui permet de trier tout type d'objet itérable.

```
t1 = [1, 10, 3, 5, 7, 30, 15, 8, 9, 12, 22, 5, 10, 5, 18]
t1.sort()
t1
```

```
[1, 3, 5, 5, 5, 7, 8, 9, 10, 10, 12, 15, 18, 22, 30]
```

```
chaine = 'azertyuiopqsdfghjklmwxcvbn'
chaine_triee = sorted(chaine)
print(chaine_triee)
```

```
['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v'
,'w','x','y','z']
```

> **Question 9** : A l'aide de **Python Tutor**, indiquer quel type objet est généré par la fonction **`sorted()`**.

`t.sort()` et **`sorted()`** acceptent un paramètre nommé **reverse** avec une valeur booléenne qui permet de choisir un tri croissant ou descendant.

```
t1 = [1, 10, 3, 5, 7, 30, 15, 8, 9, 12, 22, 5, 10, 5, 18]
t1.sort(reverse=True)
t1
```

```
[30, 22, 18, 15, 12, 10, 10, 9, 8, 7, 5, 5, 5, 3, 1]
```

```
chaine = 'azertyuiopqsdfghjklmwxcvbn'
chaine_triee = sorted(chaine, reverse=True)
print(chaine_triee)
```

```
['z','y','x','w','v','u','t','s','r','q','p','o','n','m','l','k','j','i','h','g','f','e'
,'d','c','b','a']
```

6 – POUR ALLER PLUS LOIN

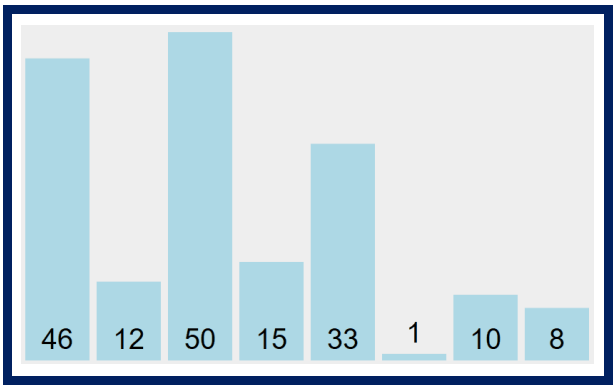
6.1 – Tri sélection et tri par insertion dans l'ordre décroissant

> **Question 10** : Modifier les fonctions **`tri_selection`**() et **`tri_insertion()`** pour obtenir un tri décroissant.

6.2 – Algorithme de tri bulle

Le principe du **tri bulle** (bubble sort ou sinking sort) est de **comparer deux à deux les éléments e1 et e2** consécutifs d'un tableau et d'effectuer une **permutation si e1 > e2**. On continue de trier jusqu'à ce qu'il n'y ait plus de permutation.

L'animation ci-contre vous présente le fonctionnement du tri sélection.



La vidéo ci-contre vous présente également le fonctionnement du tri sélection.

L'algorithme tri bulle est le suivant :

```
VARIABLES
tab : tableau d'entiers
n : entier
temp : entier
tableau_trie : booleen
i,j : entier

DEBUT
    n <-- taille tableau tab
    Pour i de n-1 à 1
        tableau_trie <-- vrai
        pour j de 0 à i-1
            si tab[j+1] < T[j]
                échanger T[j+1] et T[j]
                tableau-trie = faux
        si tableau_trie = vrai
            return tab
FIN
```

Question 11 : Editer le programme de la fonction **`tri_bulle`**. Tester son fonctionnement sur le tableau suivant :

```
t = [1, 10, 3, 5, 7, 30, 15, 8, 9, 12, 22, 5, 10, 5, 18]
```

Question 12 : Indiquer quel est le meilleur des cas et le pire des cas pour l'analyse de la complexité. Estimer le nombre de comparaisons effectuées dans le pire des cas. En déduire l'ordre de la complexité.

Question 13 : Justifier que la terminaison de l'algorithme de tri bulle est vérifiée.