

Tris

1 Tri par sélection

Dans le prolongement de la recherche d'un maximum (ou d'un minimum dans une liste on va étudier un premier algorithme de tri, le **tri par sélection**

Regardons sur un exemple, la liste $T = [L,I,G,N,E]$ comment fonctionne le tri par sélection :

i	min	0	1	2	3	4
0	4	L	I	G	N	E
1	2	E	I	G	N	L
2	2	E	G	I	N	L
3	4	E	G	I	N	L
4	4	E	G	I	L	N

1. D'abord on parcourt toute la liste pour **sélectionner la première position du plus petit élément de la liste**, dans notre exemple la position 4 pour la lettre E (en rouge)
2. Ensuite on échange de place cet élément avec l'élément de la liste repéré par $i = 0$.
3. Ensuite la lettre E (en vert) est bien placée et on recommence ce que l'on a fait précédemment **mais cette fois ci sur la partie du tableau de l'indice 1 jusqu'à la fin**
4. On parcourt le tableau de l'indice 1 jusqu'à la fin pour **sélectionner la première position du plus petit élément de la liste**, dans notre exemple la position 2 pour la lettre G (en rouge)
5. Ensuite on échange de place cet élément avec l'élément de la liste repéré par $i = 1$.
6. etc...

D'où l'algorithme

Algorithme 1 : Tri par sélection

Données : Un tableau T ayant $n \geq 2$ éléments comparables

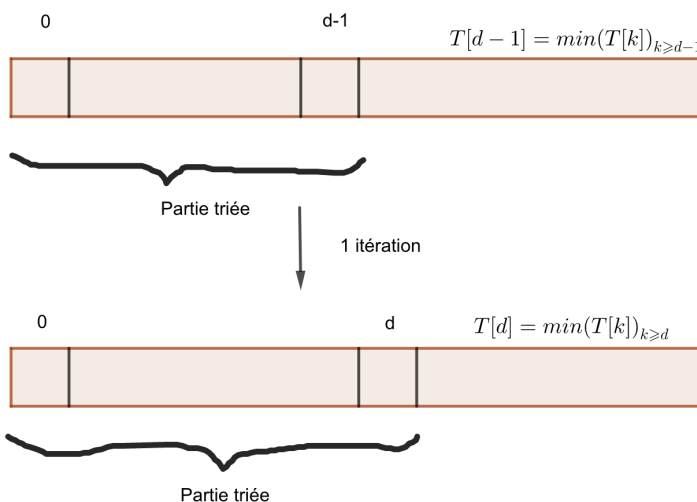
Résultat : T est trié dans l'ordre croissant

```
1 début
2   pour  $debut \in \llbracket 0; n - 1 \rrbracket$  faire
3     //Sélection de la position du plus petit élément dans l'intervalle
4      $\llbracket debut; n \llbracket$ 
5      $min \leftarrow debut$ 
6     pour  $i \in \llbracket debut + 1; n \llbracket$  faire
7       si  $T[i] < T[min]$  alors
8          $min \leftarrow i$ 
9     fin
10    //On échange les valeurs en position min et deb
11     $temp \leftarrow T[debut]$ 
12     $T[debut] \leftarrow T[min]$ 
13     $T[min] \leftarrow temp$ 
14  fin
15 fin
```

Voir ici une animation <http://www.sorting-algorithms.com/selection-sort>
Comment être sûr que ce programme renvoie toujours un tableau trié pour n'importe quel tableau d'éléments comparables ?

1.1 Preuve du programme

Dans un premier temps on remarque que l'algorithme s'arrête toujours.



L'invariant de boucle est en deux parties :

Pour d appartenant à $[1, n - 1]$ où n est la longueur du tableau

1. La partie du tableau d'indice compris entre 0 et $d-1$ est triée
2. $T[d-1] = \min(T[k])_{k \geq d-1}$

Cette propriété est vraie après le premier tour de boucle pour $d = 1$.

Supposons que cette propriété est vraie après plusieurs tours de boucle, donc

1. $T[0] \leq T[1] \leq \dots \leq T[d-1]$
2. $T[d-1] \leq T[k]$ pour tout $k \geq d-1$ donc $T[d-1] \leq T[d]$

Après un tour de boucle on va sélectionner l'indice min de la première occurrence de plus petit élément entre les indices d et $n-1$, puis échanger les valeurs $T[\min]$ et $T[d]$, donc

1. $T[0] \leq T[1] \leq \dots \leq T[d-1] \leq T[d]$
2. $T[d] \leq T[k]$ pour tout $k \geq d$

L'invariant de boucle est vrai après un tour de boucle et reste vrai entre deux tours de boucle, donc cette propriété est vraie pour $d = n - 1$, autrement dit à la fin de l'algorithme le tableau est trié.

1.2 Complexité du tri par sélection

Pour simplifier l'évaluation de la complexité on va regrouper toutes les affectations dans l'échange (lignes 11 à 13) pour un coût de c_1 . Ensuite on majore les comparaisons (ligne 6) par un coût c_2

Combien d'échanges ?

Dans la suite on note d la variable *debut*. La variable d parcourt l'intervalle de 0 à $n-2$ donc il y a $n-1$ échanges

Combien de comparaisons ? Lorsque $d = 0$, i varie de 1 à $n-1$ donc $n-1$ comparaisons, lorsque $d = 1$, i varie de 2 à $n-1$ donc $n-2$ comparaisons puis finalement $d = n-2$ et i varie de $n-1$ à $n-1$ donc une seule comparaison

En tout il y a $(n-1) + (n-2) + \dots + 2 + 1 = \frac{(n-1)n}{2} = \frac{n^2}{2} - \frac{n}{2}$ comparaisons

Donc $T(n) \leq \left(\frac{n^2}{2} - \frac{n}{2}\right)c_2 + (n-2)c_1 = \frac{n^2}{2}c_2 + n\left(c_1 - \frac{c_2}{2}\right) - 2c_1$

Donc quand n devient "très grand" le terme en n^2 l'emporte sur celui en n qui l'emporte sur le terme constant, on va donc dire que la complexité du tri par sélection est **dans tous les cas** proportionnelle à n^2 ou **quadratique**

2 Tri par insertion

L'idée et le nom du tri par insertion vient des jeux de cartes. Oublions les couleurs et considérons que les hauteurs sont classées ainsi dans l'ordre croissant $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, V, D, R]$ Imaginons que l'on vous distribue 5 cartes et que la première est V donc votre main est pour l'instant $[V]$ puis la deuxième est 4 alors puisque $4 < V$ on insère 4 à sa place et votre main devient $[4, V]$, puis la troisième carte distribuée est 7 donc insère 7 à sa place donc votre main devient $[4, 7, V]$ etc...

D'où un algorithme possible (comment insérer une carte dans la main à sa place?)

Algorithme 2 : Tri par insertion

Données : Un tableau T ayant $n \geq 2$ éléments comparables

Résultat : T est trié dans l'ordre croissant

```
1 début
2   pour  $i \in \llbracket 1; n \rrbracket$  faire
3     //on insère la carte  $T[i]$  à sa place dans  $T[:i]$  triée
4      $j \leftarrow i - 1$ 
5      $temp \leftarrow T[i]$ 
6     tant que  $j > 0$  et  $T[j] > temp$  faire
7        $T[j + 1] \leftarrow T[j]$ 
8        $j \leftarrow j - 1$ 
9     fin
10     $T[j + 1] \leftarrow temp$ 
11  fin
12 fin
```

2.1 Exercices -Tris par sélection et insertion

Ex 1

Soit $S(n) = 1 + 2 + 3 + \dots + n - 1 + n = n + n - 1 + \dots + 3 + 2 + 1$

1. Montrer que $2S(n) = n(n + 1)$
2. Que vaut $S(n)$?
3. Que vaut $1 + 2 + 3 + 4 + \dots + 1000$?

Ex 2

Faire tourner l'algorithme du tri par sélection sur le tableau $T = [3,1,4,1,5]$ avec un tableau comme dans le cours

Ex 3

Traduire l'algorithme du tri par sélection en une fonction Python. Pour plus de lisibilité utiliser deux fonctions `exchange(tab, i, j)` qui ne renvoie rien mais qui dans le tableau `tab` échange les cellules d'indices `i` et `j`

Ex 4

Faire tourner l'algorithme du tri par insertion sur le tableau $T = [3,1,4,1,5]$ avec un tableau comme dans le cours

Ex 5

Ecrire en Python et en Javascript l'algorithme du tri par insertion

Ex 6

Quel algorithme (tri par insertion ou sélection) sera le plus rapide sur un tableau tel que :

1. Toutes les valeurs sont identiques
2. Le tableau est déjà trié
3. Le tableau est trié dans l'ordre descendant

Ex 7

Faire une preuve du programme du tri par insertion

Ex 8

Montrer que la complexité du tri par insertion est quadratique dans le pire des cas (tableau trié dans l'ordre descendant) et linéaire dans le meilleur des cas (tableau déjà trié)

Ex 9

Voici un autre algorithme du tri par insertion qui propose une autre méthode pour insérer la carte dans la main à sa place.

Vérifier que cet algorithme est correct.

Algorithme 3 : Tri par insertion

Données : Un tableau T ayant $n \geq 2$ éléments comparables

Résultat : T est trié dans l'ordre croissant

```
1 début
2   pour  $i \in \llbracket 1; n \llbracket$  faire
3     pour  $j \in \llbracket 0; i \llbracket$  faire
4       si  $T[j] < T[j-1]$  alors
5         //On échange les valeurs en position  $j$  et  $j-1$ 
6          $temp \leftarrow T[j]$ 
7          $T[j] \leftarrow T[j-1]$ 
8          $T[j-1] \leftarrow temp$ 
9       sinon
10        | sortir de la boucle
11      fin
12    fin
13  fin
14 fin
```
