

Algorithme de tri

Il existe énormément d'algorithmes de tri, certains plus performant que d'autres (en termes de complexité). Nous allons travailler sur 2 algorithmes "classiques" : le tri par insertion et le tri par sélection.

Tri par insertion

C'est le tri que font la plupart des gens lorsqu'ils trient un jeu de carte. On insère chaque nouvelle carte à sa place. Etudions le truc : Prenons 5 cartes au hasard dans un.



cartes triées

cartes à trier



Froidement, je ne m'intéresse d'abord qu'à la première carte. Elle est « triée » avec elle-même.

cartes triées

cartes à trier



Je m'intéresse ensuite à la seconde carte et je la trie avec la première. Ça donne ce résultat.

cartes triées

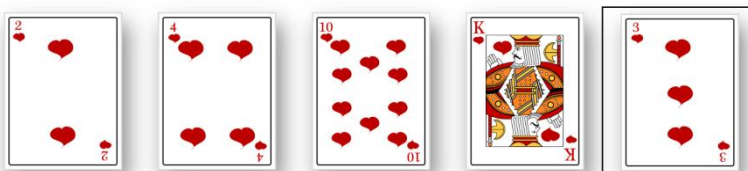
cartes à trier



Je m'intéresse maintenant à la troisième carte que je vais classer avec les deux premières. Ici, elle ne bouge pas, vu que le roi, c'est le king, c'est plus grand qu'un 4 et qu'un pauvre 2.

cartes triées

carte à trier



Et je continue, je prends la carte suivante (le 10) et je l'insère à sa place.

cartes triées



Et ainsi de suite, je m'occupe donc de la carte suivante, le 3, que je j'insère à sa place.

Et voilà, mes cartes sont triées 😊 Vous ne faites pas comme ça lorsque vous triez vos cartes ? Vous avez une autre méthode ? On en parle tout à l'heure, pour l'instant, on va écrire l'algorithme de cette méthode.

On ne va pas demander à notre programme de trier des cartes, mais plus sûrement des tableaux ou des listes.
Mettons donc nos cartes dans un tableau **t** :

4	2	13	10	3
---	---	----	----	---

Rappel sur les tableaux : ici, $t[0]=4$, $t[1]=2$, $t[2]=13$, $t[3]=10$, $t[4]=3$

Réfléchissons bien à ce que nous faisons lorsqu'on fait notre tri, et à ce qu'on va avoir besoin de faire comme opérations algorithmiques :

- Parcourir le tableau
- Faire des comparaisons
- « décaler » les valeurs pour pouvoir insérer la carte à la bonne place.

Lançons-nous dans l'écriture d'un algorithme trivial avant de faire plus tard quelque chose de magnifique :

Var t en tableau d'entier

Var k en entier // ça sera ma valeur tampon, pour ne pas casser mon tableau (cf tp image...)

k=t[1] // on prend la seconde carte pour l'insérer.

Si k<t[0]

t[1]=t[0]

t[0] = k

fin si // et fin du premier tri

k=t[2] // on prend la troisième carte pour l'insérer

// On se rend compte que c'est un peu moins simple, la, parce qu'il ne suffit plus de la comparer à une carte mais à deux cartes... Si j'analyse de près ce que je fais quand je trie cette carte, je vois que je fais ça : je parcours les cartes triées et des que je tombe sur une carte plus grande, je place ma carte juste avant. Continuons dans la trivialité pour exprimer cela :

Si k<t[0]

t[2]=t[1]

t[1] = t[0]

t[0] = k

sinon

Si k<t[1]

t[2]=t[1]

t[1] = k

fin si

// Trions maintenant la carte suivante

k=t[3] // on prend la troisième carte pour l'insérer

Si k<t[2]

t[3]=t[2]

t[2]=k

sinon

Si k<t[1]

t[3]=t[2]

t[2]=t[1]

t[1] = k

sinon

Si k<t[0]

t[3]=t[2]

t[2]=t[1]

t[1] = t[0]

t[0] = k

fin si

*On comprend vite (j'espère..) qu'il y a peut être moyen de faire une boucle, non ? Cessons la trivialité et réfléchissons. Ce qu'on fait, c'est : tant que ma carte est plus petite, on « avance vers la gauche » en décalant les autres. Des qu'elle est plus grande, on l'insère.
On peut écrire cela comme ça :*

j=rang de la carte à trier

k=t[j]

i = rang de la carte juste avant celle à trier (donc $i = j-1$)

Tant que $k < t[i]$ et que $i >= 0$

$t[i+1]=t[i]$

$i=i-1$

fin tant que

$t[i]=k$

On a presque fini. On a déjà fait l'algorithme pour trier une carte. Pour les trier toutes, il suffit de boucler autant de fois qu'il y a de cartes :

VAR t : tableau d'entiers
VAR i, j, k : nombre entier

DEBUT

j ← 1

tant que j < longueur(t): //boucle 1 : boucle pour prendre en main chaque carte une à une pour l'insérer.

i ← j-1

k ← t[j]

tant que k < t[i] et que i >= 0 //boucle 2 : boucle pour placer la carte qu'on a en main.

t[i+1] ← t[i]

i ← i-1

fin tant que

t[i+1] ← k

j ← j+1

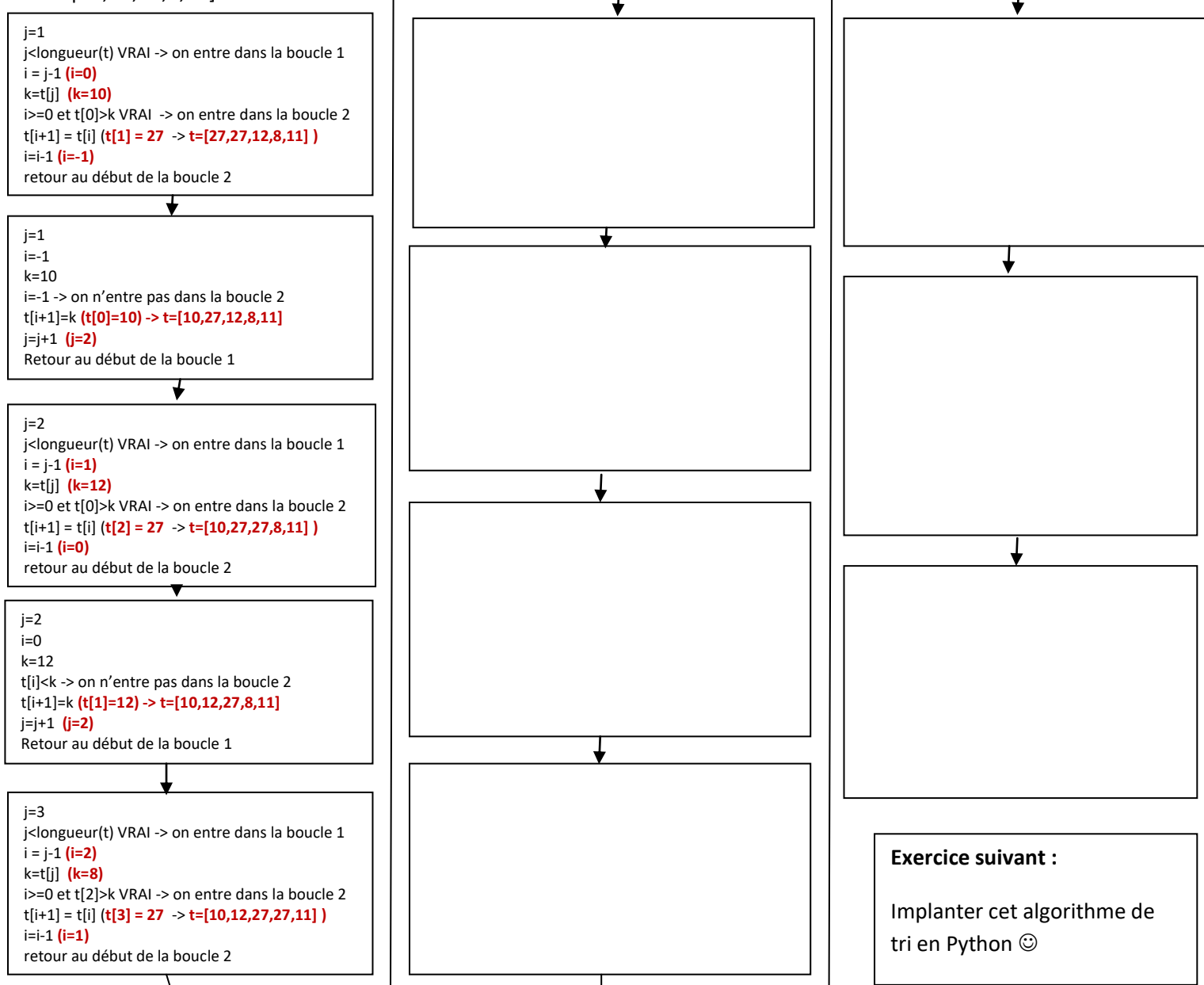
fin tant que

FIN

Si ça ce n'est pas magnifique... ☺

Pour bien comprendre cet algorithme, on va le dérouler à la main en prenant un exemple de tableau à trier. Votre travail : Poursuivez le travail commencé ci-dessous (attention de bien donner l'état du tableau à chaque étape)

T = [27, 10, 12, 8, 11]



Complexité de cet algorithme :

Vous vous en souvenez, pour trouver la complexité asymptotique temporelle, on ne s'embête pas avec les coefficients et les constantes, on y va « à la louche »

On s'aperçoit déjà que cet algorithme semble plus complexe que ceux qu'on avait étudiés de complexité $O(n)$. Pour ceux-ci, on ne parcourait que quelques fois la liste, ce « quelque fois » étant indépendant du nombre de valeurs contenu dans la liste.

Ici, on s'aperçoit que plus le nombre de valeurs sera grand, plus le nombre de fois qu'on parcourra la liste sera grand lui aussi.

A la louche, on pourrait dire qu'on parcourt **n fois la liste**.

On peut donc dire, toujours « à la louche », qu'on fera **$n \cdot n$** opérations élémentaires, soit **n^2** opérations élémentaires (peut être $4n^2$ ou $6n^2$, mais le coefficient, on s'en moque).

On dira donc ici que la complexité s'écrit **$O(n^2)$**

Entre nous, on dira que la complexité est en **n^2**

On dit aussi qu'elle est **quadratique**