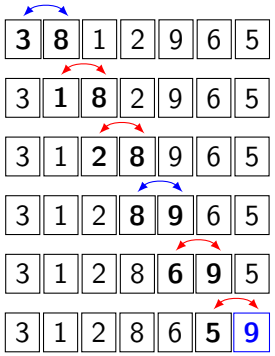
Des algorithmes de tri

Lycée Blaise Pascal

Décembre 2015

Le tri à bulles

Le tri à bulles par l'exemple : 3 8 1 2 9 6 5



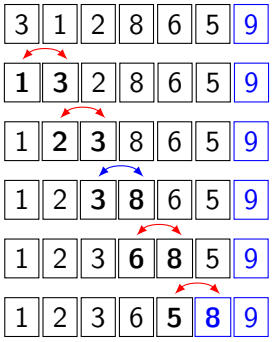
Première étape : on parcourt la liste du rang 0 au rang $n-1$.

- On compare deux termes voisins.
- si besoin, on échange ces deux termes pour disposer du plus grand à droite.

A l'issue de cette étape, le plus grand nombre est en queue de liste.

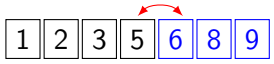
Le tri à bulles

Deuxième étape :

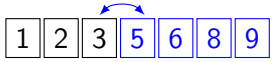


Et ensuite...

A l'issue de la troisième étape :



De la quatrième étape :



Et finalement :

Le tri à bulles de complexité quadratique

On dénombre :

- $(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}$ comparaisons.
- De 0 à $\frac{n(n-1)}{2}$ échanges.

Le tri à bulles est en $O(n^2)$.

Implémentation en Python

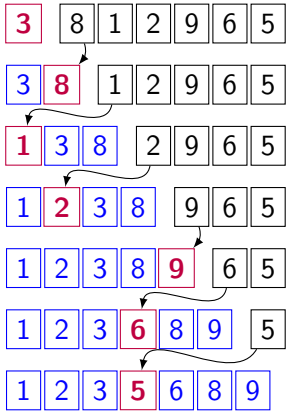
On obtient :

```
def echange(tab, i, j):

    tab[i], tab[j] = tab[j], tab[i]


def tri_bulles(liste):

    for i in range(len(t)-1, 0, -1):
        for j in range(i):
            if liste[j] > liste[j + 1]:
                echange(liste, j, j+1)
    return liste
```


Le jeu de cartes

Le tri par insertion

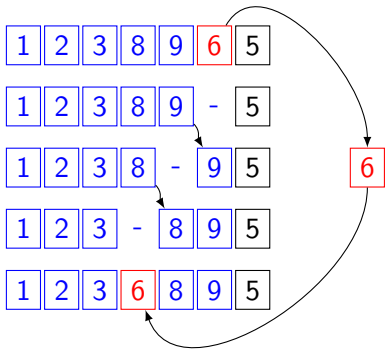
Le tri par insertion par l'exemple : 3 8 1 2 9 6 5



Légende :

- En queue de liste (à droite, en noir) : les éléments à trier.
- En tête de liste (à gauche, en bleu) : les éléments triés.
- Le premier élément à trier est **inséré** dans la partie triée (en gras rouge).

Coût d'une insertion dans une liste



- La valeur à déplacer est affectée à une variable auxiliaire.
- On compare un terme de la liste à la valeur auxiliaire.
- Si besoin, on déplace le terme comparé et on retourne au point ci-dessus.
- La variable auxiliaire est affectée à sa place dans la liste.

L'insertion à sa place du terme de rang i est en $O(i)$.

Algorithmes d'insertion et de tri par insertion

Procédure insertion(`liste, i`) :
  Affecter à `aux` la valeur `liste[i]`
  Tant que `aux < liste[i-1]` et `i > 0` **faire :**
    Affecter à `i` la valeur `i-1`
    Affecter à `liste[i+1]` la valeur `liste[i]`
  Affecter à `liste[i]` la valeur `aux`

La complexité de l'insertion est en $O(i)$.

Procédure tri_insertion(`liste`) :
  Affecter à `n` la valeur de la longueur de la `liste`
  Pour `i` **variant de** 0 **à** `n-1` **faire :**
    Exécuter `insertion(liste, i)`

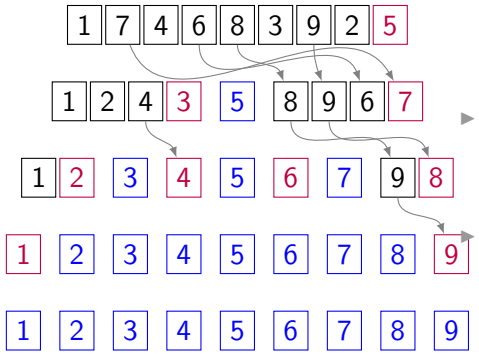
La complexité du tri par insertion est en $O(n^2)$.

Diviser pour régner

Le tri rapide

Quicksort

Le tri rapide par l'exemple



Sur le principe de

"diviser pour régner" :

- On choisit un pivot : par exemple le dernier terme de la liste (ici en rouge).
- On laisse à gauche les termes inférieurs au pivot, et on place à droite du pivot ceux qui lui sont supérieurs.
- Sur les sous-listes gauche et droite, on réapplique les deux points ci-dessus.

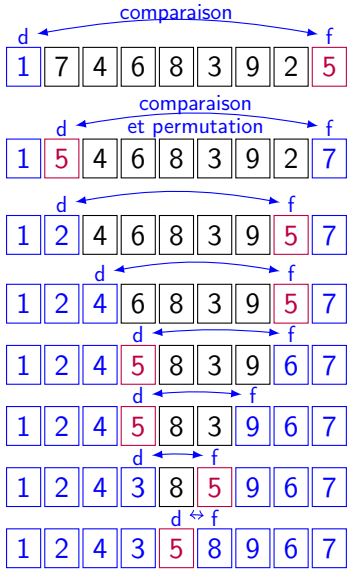
Une première implémentation en Python

L'élégance de la concision ...

```
def tri_rap(t):

  if (liste == []):
    return []
  else:
    return (tri_rap([i for i in t[1:] if (i<t
       [0])])\
      + [t[0]]\
      + tri_rap([i for i in t[1:] if (i>=t[0])])
         )
```

... prend parfois beaucoup de place !

Une façon de partitionner en place



- On dispose de deux curseurs placés en tête et queue de la partie de la liste à partitionner.
- On compare les deux valeurs situées aux curseurs.
- Si besoin, on les échange pour les ordonner.
- L'un des curseurs ne pointe pas sur le pivot. S'il s'agit de `d`, on l'incrémente, s'il s'agit de `f`, on le décrémente.
- On réitère les trois points précédents.

Complexité du partitionnement

Pour une liste de n éléments, le partitionnement nécessite :

- $n-1$ comparaisons.
- Au plus $n-1$ permutations.

Ainsi le partitionnement est en $O(n)$.

Dans la suite, on ne compte plus que les comparaisons...

Complexité du tri rapide : le pire cas

Le pire des cas :
A chaque partition, on choisit comme pivot, le plus grand (ou le plus petit) élément : ce qui arrive sur une liste déjà triée.

5 4 3 2 1

Dans ce cas, la première partition nécessite $n-1$ comparaisons,

1 | 5 4 3 2

La deuxième partition nécessite $n-2$ comparaisons,

1 2 | 5 4 3

La troisième partition nécessite $n-3$ comparaisons...

Au total, $\frac{n(n-1)}{2}$ comparaisons. La complexité de ce tri est en $O(n^2)$.

Complexité du tri rapide : le meilleur cas

Le meilleur des cas :
Le pivot partitionne la liste en deux sous-listes de même taille (à 1 près). (Voir l'exemple introductif)

Si on note c_n la complexité, on a au mieux :

$$\forall n \in \mathbb{N}^*, \quad c_n = n - 1 + 2 \times c_{\lfloor \frac{n}{2} \rfloor}$$

Simplifions un peu en supposant qu'il existe un entier p tel que : $n = 2^p$. La relation précédente devient :

$$\forall p \in \mathbb{N}, \quad c_{2^p} = 2^p - 1 + 2 \times c_{2^{p-1}}$$

Complexité du tri rapide : le meilleur cas

$$\begin{aligned}
\forall p \in \mathbb{N},\ c_{2^p} &= 2^p - 1 + 2c_{2^{p-1}} \\
&= 2^p - 1 + 2\left(2^{p-1} - 1 + 2c_{2^{p-2}}\right) \\
&= 2^p - 1 + 2^p - 2 + 2^2 c_{2^{p-2}} \\
&= 2^p - 1 + 2^p - 2 + 2^2\left(2^{p-2} - 1 + 2c_{2^{p-3}}\right) \\
&= 2^p - 1 + 2^p - 2 + 2^p - 2^2 + 2^3 c_{2^{p-3}} \\
&= \dots \\
&= 2^p - 1 + 2^p - 2 + 2^p - 2^2 + \cdots + 2^p - 2^{p-1} + 2^p c_{2^0} \\
&= p \times 2^p - \frac{1 - 2^p}{1 - 2} + 2^p c_1 \\
c_{2^p} &= p \times 2^p + 1 - 2^p + 2^p c_1
\end{aligned}$$

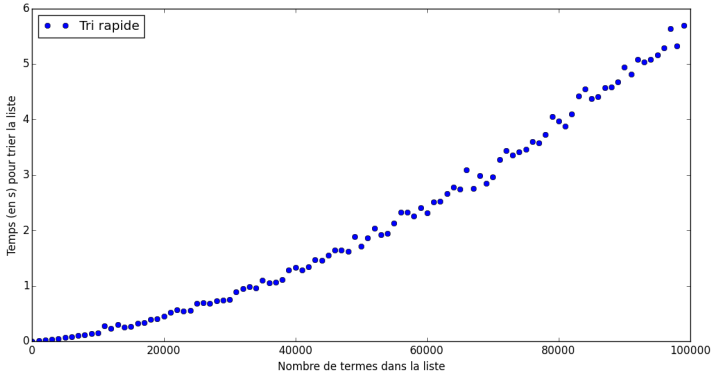
D'où :

$$\forall n \in \mathbb{N}^*,\ c_n = \log_2(n) \times n + 1 - n$$

La complexité de ce tri est en $O(n \ln(n))$.

Complexité du tri rapide : en moyenne

On pourra lire avec profit la page :
`http://imss-www.upmf-grenoble.fr/prevert/Prog/Tris/triRapideEval.html`



pour l'heure une image nous convaincra qu'en moyenne la complexité du tri rapide est en $O(n \ln(n))$.

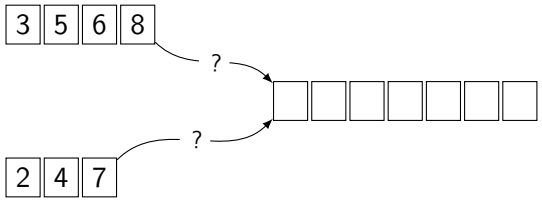
Un tri qui ne tient pas en place

Le tri fusion

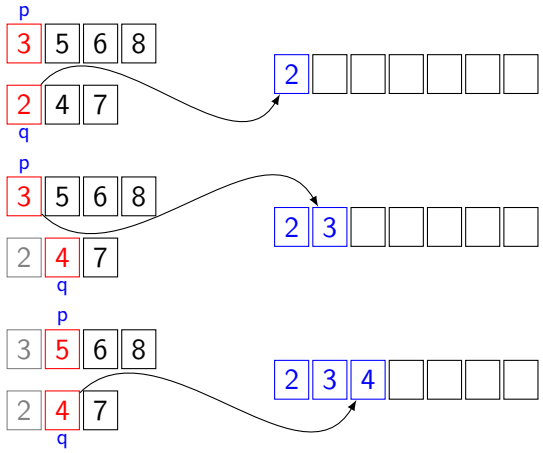
Merge sort

La fusion par l'exemple : 2 4 7 et 3 5 6 8

On considère deux listes triées et on veut les fusionner pour en faire une seule liste triée.

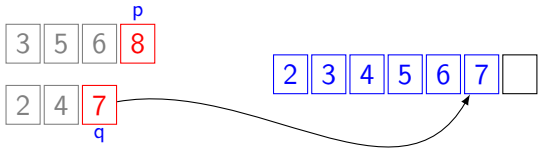


La fusion en action

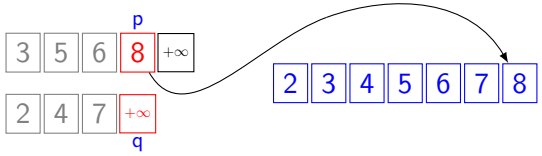


etc. . .

La fusion : gérer une liste vide



Ici, la deuxième liste est vide. Où placer le pointeur q ?



Une idée consiste à ajouter en bout de liste un nombre plus grand que tous les autres. En python on pourra utiliser : `float('inf')`.

Le tri fusion : un exemple de fonction récursive

Fonction tri_fus(`liste`) :
  Si (longueur(`liste`) $\leqslant 1$) **alors :**
    renvoyer `liste`
  Fin du si
  Sinon :
    Affecter à `m` la valeur `len(liste)//2`
    Affecter à `lgauche` la sous-liste
      de `liste` du rang 0 au rang `m`
    Affecter à `ldroite` la sous-liste
      de `liste` du rang `m+1` à la fin de la liste
    renvoyer `fusion(tri_fus(lgauche), tri_fus(ldroite))`
  Fin du sinon

Complexité de la fusion

3	5	6	8

2	4	7

2	3	4	5	6	7	8

La fusion de deux listes de longueur i et j est de complexité $O(i+j)$.

Complexité du tri fusion

Si on note c_n la complexité du tri fusion, on a :

$$\forall n \in \mathbb{N}^*, \quad c_n = c_{\lfloor \frac{n}{2} \rfloor} + c_{\lfloor \frac{n}{2} \rfloor} + n$$

C'est à dire :

$$\forall n \in \mathbb{N}^*, \quad c_n = 2 \times c_{\lfloor \frac{n}{2} \rfloor} + n$$

Ce qui est (presque) la même relation que dans l'étude du meilleur cas de la complexité du tri rapide.

Ainsi, la complexité du tri fusion est en $O(n \ln(n))$.

Trier c'est bien, mais...

Pourquoi trier ?

- Une recherche dichotomique dans une liste triée est de complexité $O(\ln n)$.
- Une recherche dans une liste non triée est de complexité $O(n)$.

Alors trions !

Le tri cantine

canteen sort