

# Tri rapide

December 13, 2021

## 1 Principe

Le tri rapide est basée sur un élément de la liste appelé pivot. Cet élément pivot a deux fonctions :  
- il est placé à sa place finale dans la liste à chaque appel. - il divise la liste en deux sous listes qui sont triées de manière récursive par la même fonction.

On commence par choisir au hasard un élément pivot  $p$  par exemple le premier élément de la liste initiale  $L$  avec  $p = L[0]$ .

On a donc pour une liste de taille  $n$ , une liste à trier  $L = [p, L[1], L[2], \dots, L[n-1]]$ .

Le pivot se trouve à sa place finale quand tous les éléments qui le précèdent lui sont inférieurs, et tous les éléments qui le succèdent lui sont supérieurs. Pour arriver à ce résultat, on compare  $p$  avec tous les autres éléments  $L[i]$  allant de  $L[1]$  jusqu'à  $L[n-1]$ . Si  $L[i]$  est inférieur à  $p$ , alors on le déplace avant  $p$ , si  $L[i]$  est supérieur à  $p$ , alors on le laisse après  $p$ .

On a donc une liste  $L = [L[a], L[b], \dots, p, L[\alpha], L[\beta], \dots]$  avec tous les éléments  $L[a], L[b], \dots \leq p \leq L[\alpha], L[\beta], \dots$ .

Cette liste peut être séparée en deux autour du pivot comme  $L = [L_1, p, L_2]$ , comme tous les éléments de  $L_1$  sont inférieurs à  $p$  qui est inférieur à tous les éléments de  $L_2$ , il n'y a pas de fusion à faire  $L_1, p$ , et  $L_2$  sont bien ordonnées. Par contre il faut trier  $L_1$  et  $L_2$ , pour cela la fonction de tri rapide s'appelle elle même pour trier  $L_1$  et  $L_2$ .

On réalise un appel récursif, il faut donc prévoir un cas d'arrêt. Ici si  $L$  est de taille 1 ou 0, on a rien à faire la liste est déjà triée ou vide.

Il faut aussi prévoir la terminaison de l'algorithme. Ici la taille de la liste à trier décroît d'au moins 1 à chaque appel récursif jusqu'à atteindre le cas d'arrêt 0 ou 1. En effet le pivot est retiré de la liste à trier à chaque appel ce qui diminue la taille de la liste restante d'au moins 1. Et la taille  $n$  d'une liste est un entier positif donc on atteint bien le cas d'arrêt et en un nombre fini d'appel.

## 2 Exemple de tri rapide

Déroulons les opérations successives du tri rapide sur un exemple  $L = [14, 1, 4, 3]$ :

- on choisit le pivot  $p = L[0] = 14$
- on compare  $p$  et  $L[1] = 1$ ,  $1 < 14$ , donc on réarrangera  $L$  en  $[1, 14, \dots]$
- on compare  $p$  et  $L[2] = 4$ ,  $4 < 14$  donc on réarrangera  $L$  en  $[1, 4, 14, \dots]$
- on compare  $p$  et  $L[3] = 3$ ,  $3 < 14$  donc on réarrangera  $L$  en  $[1, 4, 3, 14]$
- on obtient  $L = [1, 4, 3, 14]$ , on divise  $L$  en  $[L_1, p, L_2]$  avec  $L_1 = [1, 4, 3]$  et  $L_2 = []$
- $L_2$  est vide donc est déjà triée c'est le cas d'arrêt
- on choisit le pivot  $p_1 = L_1[0] = 1$

- on compare  $p\_1$  et  $L\_1[1] = 4$ ,  $1 < 4$  donc on laissera  $[1, 4, \dots]$
- on compare  $p\_1$  et  $L\_1[2] = 3$ ,  $1 < 3$  donc on laissera  $[1, 4, 3]$
- on obtient  $L\_1 = [1, 4, 3]$ , on divise  $L\_1$  en  $[L\_1\{11\}, p\_1, L\_1\{12\}]$  avec  $L\_1\{11\} = []$  et  $L\_1\{12\} = [4, 3]$
- $L\_1\{11\}$  est vide donc déjà triée c'est le cas d'arrêt
- on choisit le pivot  $p\_{12} = L\_{12}[0] = 4$
- on compare  $p\_{12}$  et  $L\_{12}[1] = 3$ ,  $3 < 4$  donc on réangerra  $L\_{12}$  en  $[3, 4]$
- on obtient  $L\_{12} = [3, 4]$ , on divise  $L\_{12}$  en  $[L\_{121}, p\_{12}, L\_{122}]$  avec  $L\_{121} = [3]$  et  $L\_{122} = []$
- $L\_{122}$  est vide c'est le cas d'arrêt et  $L\_{121}$  est déjà triée donc c'est le deuxième cas d'arrêt.
- on dépile la pile d'appel récursif:
  - $L\_{121} = [3]$ ,  $p\_{12} = 4$ ,  $L\_{122} = []$  donc  $L\_{12} = [3, 4]$
  - $L\_{11} = []$ ,  $p\_1 = 1$ ,  $L\_{12} = [3, 4]$  donc  $L\_1 = [1, 3, 4]$
  - $L\_1 = [1, 3, 4]$ ,  $p = 14$ ,  $L\_2 = []$  donc  $L = [1, 3, 4, 14]$

### 3 Réalisation

Ecrivons l'algorithme de tri en le décomposant en chaque fonction élémentaire.

La première fonction consiste à choisir et placer le pivot dans la portion de liste entre les indices debut et fin.

Cette fonction renvoie un tuple avec la nouvelle position du pivot et la liste avec le pivot positionné.

```
[2]: def positionnement_pivot(L,debut,fin):
    p = L[debut]
    j = debut
    for i in range(debut,fin):
        if p > L[i] :
            L = L[:j]+[L[i]]+L[j:i]+L[i+1:]
            j +=1

    return j,L
```

On peut par exemple positionner le pivot de  $L = [14, 1, 4, 3]$  avec:

```
[3]: L = [14, 1, 4, 3]
    positionnement_pivot(L,0,len(L))
```

```
[3]: (3, [1, 4, 3, 14])
```

La deuxième fonction constitue l'appel récursif du tri rapide.

Elle effectue le tri rapide complet sur la portion de tableau allant de debut à fin.

```
[4]: def tri_rapide_rec(L,debut,fin):
    if debut+1>=fin :
        return L
    else :
        j,L = positionnement_pivot(L,debut,fin)
        debut_1 = debut
        fin_1 = j
```

```

    L = tri_rapide_rec(L,debut_1,fin_1)
    debut_2 = j+1
    fin_2 = fin
    L = tri_rapide_rec(L,debut_2,fin_2)
    return L

```

On peut par exemple l'appeler pour [14, 1, 4, 3] entre 0 et 4

```

[5]: L = [14, 1, 4, 3]
     tri_rapide_rec(L,0,4)

```

```

[5]: [1, 3, 4, 14]

```

La fonction tri rapide est donc simplement l'exécution de l'appel récursif sur l'ensemble du tableau

```

[6]: def tri_rapide(L) :
     L = tri_rapide_rec(L,0,len(L))
     return L

```

```

[7]: L = [14, 1, 4, 3]
     tri_rapide(L)

```

```

[7]: [1, 3, 4, 14]

```

par exemple on peut visualiser les différentes étapes du tri de la liste [14 1 4 3] avec les lignes de commandes suivantes

```

[6]: def positionnement_pivot_visualisation(L,debut,fin):
     p = L[debut]
     j = debut
     for i in range(debut,fin):
         if p > L[i] :
             L = L[:j]+[L[i]]+L[j:i]+L[i+1:]
             j +=1
     print(L)
     return j,L

def tri_rapide_rec_visualisation(L,debut,fin):
    if debut+1>=fin :
        return L
    else :
        j,L = positionnement_pivot_visualisation(L,debut,fin)
        print(L)
        debut_1 = debut
        fin_1 = j
        L = tri_rapide_rec(L,debut_1,fin_1)
        debut_2 = j+1
        fin_2 = fin
        L = tri_rapide_rec(L,debut_2,fin_2)
        return L

```

```
def tri_rapide_visualisation(L) :
    L = tri_rapide_rec_visualisation(L,0,len(L))
    print(L)
    return L
```

```
L = [14, 1, 4, 3]
tri_rapide_visualisation(L)
```

```
[14, 1, 4, 3]
[1, 14, 4, 3]
[1, 4, 14, 3]
[1, 4, 3, 14]
[1, 4, 3, 14]
[1, 3, 4, 14]
```

[6]: [1, 3, 4, 14]

## 4 Complexité temporelle

Le positionnement du pivot dépend de la liste initiale, on se retrouve à nouveau dans le cas d'une étude de la complexité qui peut varier selon que l'on est dans le pire des cas ou le meilleur des cas.

### 4.1 Calcul de la complexité dans le meilleur des cas

Le meilleur des cas est une liste pour laquelle le pivot se place au centre de la liste de taille  $n$  et la divise en deux listes de taille  $n/2$ .

Le nombre d'opérations à effectuer est alors de  $n - 1$  pour le positionnement du pivot puis toutes les opérations pour trier les deux listes de taille  $n/2$ , soit  $2 \times C(n/2)$ .

On a donc une relation de récurrence  $C(n) = 2C(n/2) + n - 1$ , comme pour le tri par fusion.

Dans le meilleur des cas, nous avons de même que pour le tri fusion une complexité  $C(n) = O(n \log n)$ , soit une complexité quasi-linéaire.

### 4.2 Calcul de la complexité dans le pire des cas

Le pire des cas est une liste pour laquelle le pivot se place en début ou en fin de la liste initiale et la divise en deux listes de taille 0 et  $n$ .

Le nombre d'opération à effectuer est alors de  $n - 1$  pour le positionnement du pivot puis toutes les opérations pour trier une liste de taille  $n-1$ , soit  $n - 1$ .

On a donc une relation de récurrence  $C(n) = n - 1 + C(n - 1)$ . Si on développe le calcul de la relation de récurrence  $C(n) = n - 1 + C(n - 1) = n - 1 + n - 2 + C(n - 2) = n - 1 + n - 2 + n - 3 + \dots + 1 + C(1) = \sum_{i=n-1}^1 i = \frac{(n-1)(n-2)}{2}$ .

Dans le pire des cas nous avons de même que pour le tri par insertion une complexité  $C(n) = O(n^2)$ , soit une complexité quadratique.

## 5 Exercices

### 5.1 exercice : meilleur et pire des cas

- Dérouler chaque étape de l'algorithme de tri rapide sur la liste [15, 4, 2, 9, 55, 16, 0, 1]
- Ecrire cette même liste où les éléments sont ordonnés dans le meilleur des cas et dérouler chaque étape de son tri
- Ecrire cette même liste où les éléments sont ordonnés dans le pire des cas et dérouler chaque étape de son tri

### 5.2 exercice : calcul numérique de la complexité temporelle

- à l'aide de tirage au sort successif de liste de différentes tailles, mesurer numériquement le temps mis pour l'exécution du tri rapide sur ces listes et tracer le temps d'exécution en fonction de la taille des listes à trier.
- est-ce cohérent avec le calcul de complexité effectué précédemment ?

### 5.3 exercice : complexité spatiale

- calculer la complexité spatiale d'un algorithme de tri rapide

### 5.4 exercice : tri rapide avec pivot en sentinelle

Dans cet exercice nous allons programmer une version du tri rapide différent de celui présenté plus haut.

Un pivot est toujours choisit dans la liste, mais nous allons considérer une autre méthode pour trier les deux sous listes d'élément inférieur et supérieur au pivot.

Au cours de l'organisation des éléments autour du pivot la liste est orientée comme suit :

$L = [p, L[1], L[2], \dots, L[i-1], L[i], \dots, L[j-1], L[j], \dots]$

avec  $p$  le pivot,

tous les éléments de  $L[1]$  à  $L[i-1]$  sont inférieurs à  $p$ ,

tous les éléments de  $L[i]$  à  $L[j-1]$  sont encore inconnus,

tous les éléments de  $L[j]$  jusqu'à la fin de la liste sont supérieurs à  $p$ .

Considérons l'algorithme suivant prenant en entrée 1 une liste.

- (0) Choisir pour pivot le premier élément de la liste. Initialiser judicieusement les variables  $i$  et  $j$ .
  - (1) Augmenter  $i$  tant que ça ne modifie pas l'organisation de la liste.
  - (2) Diminuer  $j$  tant que ça ne brise pas l'organisation de la liste.
  - (3) Si la zone de points d'interrogation est vide, arrêter l'algorithme et renvoyer  $i$ .
  - (4) Échanger  $L[i]$  et  $L[j]$ , augmenter  $i$  de 1 et diminuer  $j$  de 1.
  - (5) Retourner à l'étape (1).
- Écrire une fonction implémentant cet algorithme en Python.

- Pourquoi l'étape (4) de l'algorithme conserve toujours la bonne organisation de la liste ?
- Quelle est la complexité de cette fonction ?
- Utiliser cette fonction pour écrire une fonction permettant d'implémenter le tri rapide comme vu plus haut.