Algorithme de tri rapide

I. Fonctionnement

On considère un tableau $t = [t_0; t_1; \dots t_{n-1}]$ de taille n. Le but de l'algorithme est de trier le tableau t **en place**, en effectuant uniquement des permutations d'éléments, sans allouer de nouveau tableau.

I. 1. Fonctionnement général

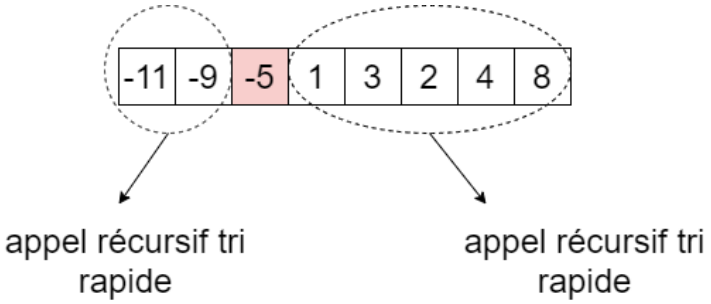
L'algorithme est globalement récursif mais fait appel à une sous-fonction itérative. L'algorithme travaille en place, directement dans le tableau, en effectuant uniquement des échanges de valeurs (transpositions, cf chapitre de mathématiques à venir sur le groupe symétrique).
Comme dans l'algorithme de recherche dichotomique, la récursivité s'opère sur la taille de la fenêtre de travail. Celle-ci est délimitée par la donnée de deux indices : `l` (indice gauche) et `r` (indice droit strict). Les éléments du tableau sur lesquels on travaille sont compris dans l'intervalle d'indices $[l, r-1]$. La taille de la fenêtre de travail est donc $r-l$ et on a toujours $0 \leq l \leq r \leq n$.

Tripartition : On suppose que l'on dispose d'une fonction de tripartition qui permet de répartir les éléments d'un tableau en trois segments par rapport à une valeur de référence, appelée **pivot** :

- **un segment gauche** qui ne contient que des valeurs strictement inférieures au pivot,
- **un segment central** qui contient toutes les valeurs égales au pivot,
- **un segment droit** qui ne contient que des valeurs strictement supérieures au pivot.

Initialisation. On choisit une valeur du tableau comme pivot et on active l'algorithme de tripartition sur l'ensemble du tableau

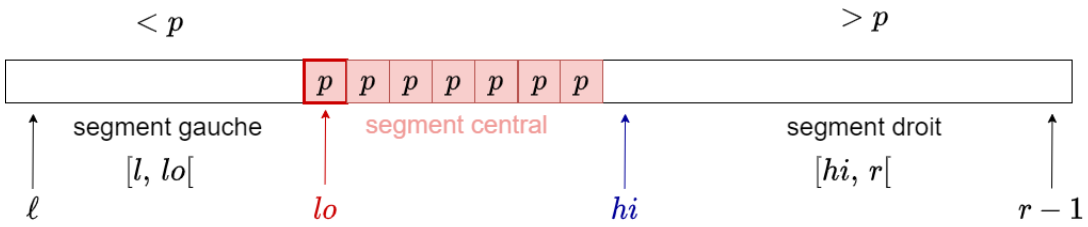
Récursivité. On appelle ensuite récursivement l'algorithme de tri rapide sur le segment gauche et sur le segment droit.



Cas de base. L'imbrication des appels récursifs cesse lorsque la taille de la fenêtre de travail est inférieure ou égale à 1 (aucun ou un seul élément dans le tableau).

I. 2. Détail de l'algorithme de tripartition

Le but de l'algorithme de tripartition est de séparer les éléments d'une fenêtre de travail $[l, r-1]$ en trois segments autour d'une valeur pivot p comme ceci :



De façon assez classique, on choisit la première valeur de la fenêtre comme pivot :

$$p = t[\ell]$$

mais n'importe quelle valeur de pivot peut être choisie et l'algorithme s'adapte facilement. Le pivot reste le même tout au long de l'algorithme de tripartition.
Pour effectuer la tripartition, on fait évoluer deux indices séparateurs lo et hi et un indice i correspondant à la valeur en cours de traitement, de sorte que, à chaque étape, on ait $\ell \leq lo < i \leq hi \leq r$:

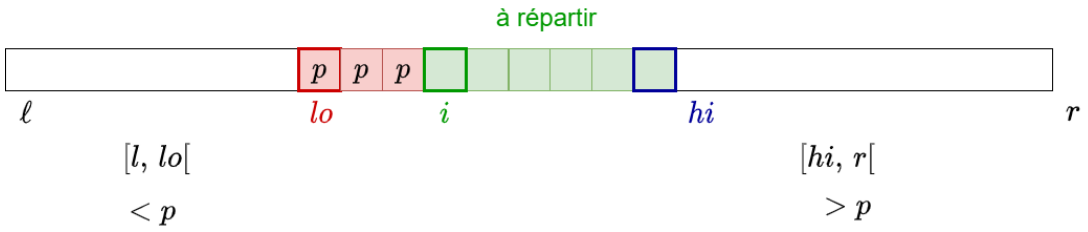
dans le segment gauche correspondant aux indices $[\ell, lo[$: tous les éléments strictement inférieurs à p

dans le segment central correspondant aux indices $[lo, i[$: tous les éléments égaux à p

dans le segment correspondant aux indices $[i, hi[$: tous les éléments restants à traiter

dans le segment droit correspondant aux indices $[hi, r[$: tous les éléments strictement supérieurs à p

L'algorithme de tripartition fonctionne de manière itérative : une boucle `while` sur l'indice i de la valeur à analyser nous permet à chaque itération de ranger la nouvelle valeur $t[i]$ dans le bon segment et de mettre en place les indices i, lo et hi pour la prochaine itération.



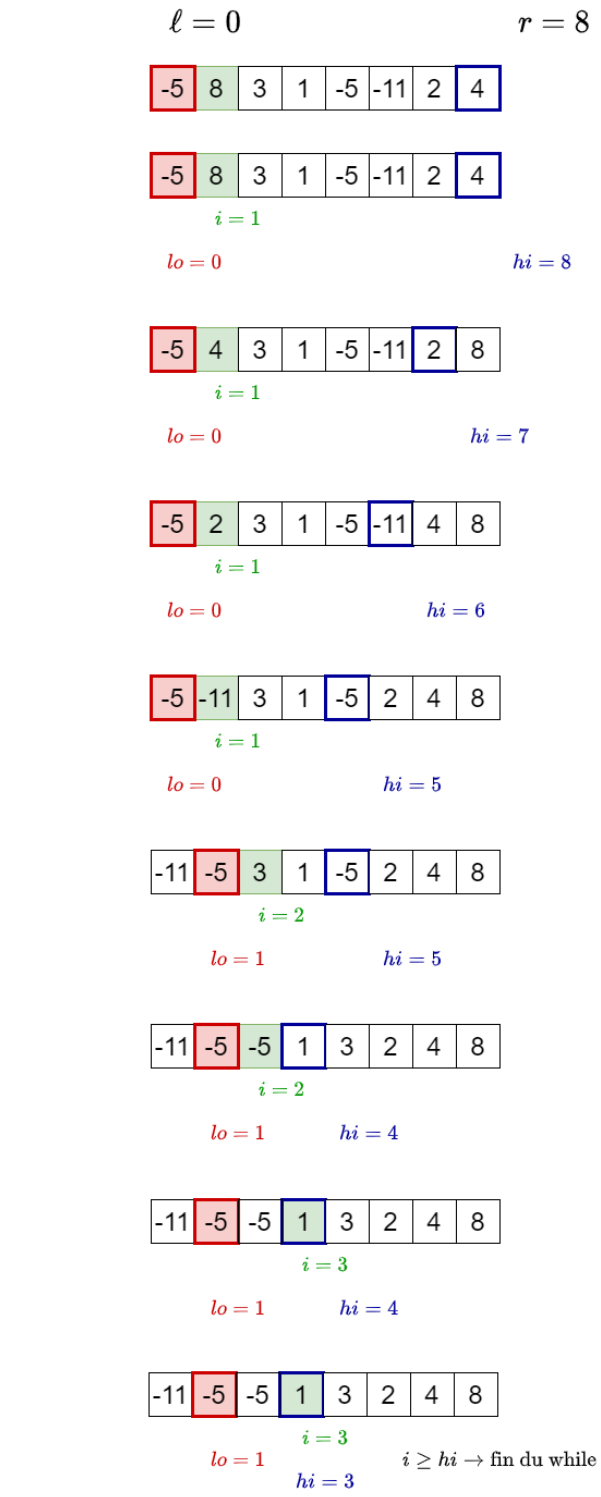
Plus précisément, à chaque itération, tant que i reste contenu dans l'intervalle des valeurs à répartir :

si $t[i] < p$, on échange $t[i]$ et $t[lo]$, on incrémente lo car le segment gauche a gagné une valeur, et on incrémente i pour le placer sur la prochaine valeur à répartir ;

si $t[i] = p$, $t[i]$ est donc déjà bien placée dans le segment central des valeurs égales au pivot. On incrémente simplement i pour le placer sur la prochaine valeur à répartir ;

si $t[i] > p$, on échange $t[i]$ et $t[hi - 1]$, et on décrémente hi de sorte que l'ancienne valeur $t[i]$ devient la valeur la plus à gauche du segment droit. Nul besoin de modifier i car la valeur maintenant située à l'indice i a été échangée avec une valeur à répartir, et sera traitée au prochain tour de boucle sans qu'il soit nécessaire de modifier i.

La figure ci-dessous montre un exemple de fonctionnement de l'algorithme de tripartition :


$\ell = 0$
$r = 8$
-5 8 3 1 -5 -11 2 4
-5 8 3 1 -5 -11 2 4
$i = 1$
$lo = 0$
$hi = 8$
-5 4 3 1 -5 -11 2 8
$i = 1$
$lo = 0$
$hi = 7$
-5 2 3 1 -5 -11 4 8
$i = 1$
$lo = 0$
$hi = 6$
-5 -11 3 1 -5 2 4 8
$i = 1$
$lo = 0$
$hi = 5$
-11 -5 3 1 -5 2 4 8
$i = 2$
$lo = 1$
$hi = 5$
-11 -5 -5 1 3 2 4 8
$i = 2$
$lo = 1$
$hi = 4$
-11 -5 -5 1 3 2 4 8
$i = 3$
$lo = 1$
$hi = 4$
-11 -5 -5 1 3 2 4 8
$i = 3$
$lo = 1$
$hi = 3$
$i \geq hi \rightarrow$ fin du while
-11 -5 -5 1 3 2 4 8
appel récursif tri rapide
$\ell = 0$
$r = lo = 1$
appel récursif tri rapide
$\ell = hi = 3$
$r = 8$
cas de base
$r - l \leq 1 \rightarrow$ fin des appels récursifs

II. Preuve de correction : grandes lignes

Au départ, $l = 0$ et $r = n$.
L'algorithme `quicksort` est structuré en 3 phases :

- tripartition des valeurs situées dans la fenêtre de travail $[l, r[$, qui renvoie les indices permettant de délimiter les trois segments lo et hi
- appel récursif `quicksort` sur le segment de gauche $[l, lo[$
- appel récursif `quicksort` sur le segment de droite $[hi, r[$

II. 1. Preuve de l'algorithme global

On suppose dans un premier temps que l'algorithme de tripartition est correct. L'algorithme global est un algorithme récursif dichotomique : on prouve sa correction par récurrence forte sur la taille $k = r - l$ de la fenêtre de travail.

$\mathcal{P}(k)$: à la sortie de l'appel `quicksort(t, n, l, r)`, le sous-tableau $[t_l; \ldots; t_{r-1}]$ est trié par ordre croissant.

Cette propriété se démontre aisément par récurrence forte en prenant soin de bien justifier tous les cas lors de la recomposition des deux segments triés.

II. 2. Preuve de l'algorithme de tripartition

L'algorithme de tripartition est itératif. On utilise donc un invariant de boucle pour montrer sa correction. On fait, on a plusieurs invariants qui sont vrais au début et à la fin de chaque tour de boucle. Ils se formulent ainsi :
$\mathcal{P}$: Soit p la valeur pivot.

- $l \leq lo < i \leq hi \leq n$
- $\forall k \in [l, lo[,\ t[k] < p$
- $\forall k \in [lo, i[,\ t[k] = p$
- $\forall k \in [hi, r[,\ t[k] > p$
- $[t_l; \ldots t_{r-1}]$ est une permutation des valeurs entre l et r du tableau initial.

III. Analyse de complexité

On évalue la complexité temporelle en termes de nombres de tests effectués.
La complexité de la boucle de tripartition est évidemment linéaire en la taille de la fenêtre n : on effectue $n - 1$ tests lors de la boucle `while`.
On en déduit, pour l'algorithme global, les complexités temporelles suivantes :

Meilleur cas : tableau ne contenant qu'une valeur, la même dans toutes les cases. Les premiers appels récursifs se font sur des tableaux vides et la récursivité s'arrête. La complexité globale de l'algorithme de tri rapide est alors linéaire en la taille du tableau.

Pire cas : tableau déjà trié (par ordre croissant ou décroissant). Si on choisit toujours le premier élément du tableau comme pivot, à chaque appel récursif, l'appel récursif sur le segment gauche se fait avec un tableau vide mais l'autre appel récursif sur le tableau droit se fait sur un tableau dont la taille a seulement diminué de 1 élément. On a donc un algorithme quadratique en la taille du tableau

Cas favorable : équilibre des segments gauche et droit après la tripartition. Nous l'étudions ci-dessous.

III. 1. Analyse de la complexité dans le cas parfaitement équilibré.

Pour simplifier cette analyse, et dans la mesure où c'est la complexité asymptotique qui nous intéresse, on considère que l'algorithme de tripartition a un coût temporel égal à $r - l$, la taille de la fenêtre de travail.
Si on note $C(n)$ la complexité temporelle de l'algorithme de tri rapide pour un tableau de taille n, on a la relation de récurrence suivante :

$$C(n) = \underbrace{n}_{\text{coup de la tripartition sur le tableau entier de taille } n} + \underbrace{2 \times C(\lfloor \frac{n}{2} \rfloor)}_{\text{2 appels récursifs équilibrés}}$$

Ainsi :

$$\begin{cases} C(0) & = 0 \\ C(1) & = 0 \\ C(n) & = n + 2 \times C(\lfloor \frac{n}{2} \rfloor) \end{cases}$$

Limitons nous au cas où n est une puissance de 2 : $n = 2^p \Leftrightarrow p = \log_2(n)$.

$$\begin{aligned} C(2^p) &= 2^p + 2 \times C(2^{p-1}) = 2^p + 2 \times (2^{p-1} + 2 \times C(2^{p-2})) \\ &= 2^p + 2^p + 4 \times (2^{p-2} + 2 \times C(2^{p-3})) \\ &= 3 \times 2^p + 2^3 \times C(2^{p-3}) = \dots \\ &= p \times 2^p + 2^p C(1) = \Theta(p2^p) = \Theta(\log_2(n)n) \end{aligned}$$

On peut ensuite généraliser ce résultat pour n'importe quelle valeur de $n\mathbb{N}$ et montrer que, dans le cas favorable équilibré, la complexité temporelle est toujours en $\Theta(n\log_2(n))$, quelque soit la taille n du tableau initial.

Exercice 1.

1. Prouver par récurrence la propriété $\mathcal{P}(k) : \forall n \in \mathbb{N}$ tels que $2^k \leq n < 2^{k+1}$, on a :

$$k2^k \leq C(n) \leq 2^{k+1}(k+1)$$

2. Conclure sur la complexité temporelle du tri rapide dans le cas favorable équilibré

III. 2. Analyse de la complexité moyenne.

Soit $n \in \mathbb{N}$. La probabilité d'obtenir un doublon dans un tableau de taille n est négligeable si on a le choix entre une infinité de valeurs pour chaque case.
On ne considère donc que des tableaux sans doublon : le segment central est donc forcément de taille 1 (une seule valeur égale au pivot dans le tableau)
On note T_n l'ensemble des tableaux de taille n sans valeurs doublons.
Pour $k \in [\![0; n-1]\!]$, on note $T_{n,k}$ la classe d'équivalence contenant tous les tableaux de taille n sur lesquels l'algorithme de tripartition décrit ci-dessus aboutit à un segment gauche de taille k et un segment droit de taille $n-1-k$. Ces classes d'équivalences forment une partition de T_n :

$$\bigcup_{k=0}^{n-1} T_{n,k} = T_n$$

On fait l'**hypothèse de modélisation probabiliste** suivante : un tableau de taille k sans doublon, formé de valeurs indépendantes piochées au hasard a autant de chance d'appartenir à chacune des n classes d'équivalences $(T_{n,k})_{k\in[\![0;n-1]\!]}$. Autrement toutes les classes d'équivalences sont équiprobables :

$$\mathbb{P}(T_{n,k}) = \frac{1}{n}$$

On considère maintenant le variable aléatoire X_n qui associe, à chaque tableau t de taille n, la complexité temporelle évaluée en nombre de comparaisons résultant de l'algorithme de tri rapide.
Soit $t = [t_0; t_1; \ldots; t_{n-1}] \in T_{n,k}$. On note $X_{n,k}$ la restriction de X_n à la classe d'équivalence $T_{n,k}$. On a alors, d'après l'algorithme :

$$X_{n,k}(t) = \underbrace{n-1}_{\text{nombre de tests dans la tripartition}} + \underbrace{X_k([t_0; t_1; \ldots; t_{k-1}])}_{\text{coût appel récursif segment gauche}} + \underbrace{X_{n-1-k}([t_{k+1}; t_{k+2}; \ldots; t_{n-1}])}_{\text{coût appel récursif segment droit}}$$

La complexité moyenne correspond à l'espérance de la variable aléatoire X_n. Comme le nombre de tableaux de taille n que l'on peut donner en entrée de l'algorithme est infini, on calcule cette espérance en réalisant une somme sur les classes d'équivalences, qui sont, elles, en nombre fini (il y en a n) :

$$C_{\text{moy}}(n) = E[X_n] = \sum_{k=0}^{n-1} \mathbb{P}(T_{n,k}) E[X_{n,k}]$$

Or, par linéarité de l'espérance :

$$E[X_{n,k}] = \underbrace{E[n-1]}_{=n-1 \text{ car la moyenne d'une constante est... cette constante}} + E[X_k] + E[X_{n-1-k}]$$

$$= n - 1 + C_{\text{moy}}(k) + C_{\text{moy}}(n-1-k)$$

On a donc :

$$C_{\text{moy}}(n) = E[X_n] = n - 1 + \sum_{k=0}^{n-1} \frac{1}{n} \left(C_{\text{moy}}(k) + C_{\text{moy}}(n-1-k)\right)$$

$$C_{\text{moy}}(n) = n - 1 + \frac{1}{n} \sum_{k=0}^{n-1} \left(C_{\text{moy}}(k) + C_{\text{moy}}(n-1-k)\right)$$

Il s'agit maintenant d'utiliser des techniques sur la manipulation de sommes pour aboutir à une expression explicite de $C_{\text{moy}}(n)$.
Un manipulation très simple d'indice sur le deuxième terme de la somme à droite permet d'écrire (symétrie) :

$$C_{\text{moy}}(n) = n - 1 + \frac{2}{n} \sum_{k=0}^{n-1} C_{\text{moy}}(k)$$

On utilise ensuite une technique d'élimination par combinaisons linéaires pour essayer de faire disparaître tous le termes $C_{\text{moy}}(k)$ intermédiaires.

On multiplie par n l'égalité ci-dessus :

$$n \times C_{\text{moy}}(n) = n(n-1) + 2\sum_{k=0}^{n-1} C_{\text{moy}}(k)$$

On la réécrit au rang $n-1$ (en substituant $n \leftarrow n-1$) :

$$(n-1) \times C_{\text{moy}}(n-1) = (n-1)(n-2) + 2\sum_{k=0}^{n-2} C_{\text{moy}}(k)$$

Puis on soustrait les deux expressions obtenues, ce qui permet de se débarrasser de tous les termes intermédiaires $C_{\text{moy}}(k)$ pour $0 \leq k \leq n-2$:

$$nC_{\text{moy}}(n) - (n-1)C_{\text{moy}}(n-1) = n(n-1) - (n-1)(n-2) + 2C(n-1)$$

$$\Leftrightarrow nC_{\text{moy}}(n) - (n+1)C_{\text{moy}}(n-1) = 2(n-1)$$

Divisons cette égalité par $n(n+1)$:

$$\Leftrightarrow \frac{C_{\text{moy}}(n)}{n+1} - \frac{C_{\text{moy}}(n-1)}{n} = \frac{2(n-1)}{n(n+1)}$$

On commence à apercevoir une possibilité de somme télescopique, écrivant la relation pour $n \to k$:

$$\frac{C_{\text{moy}}(k)}{k+1} - \frac{C_{\text{moy}}(k-1)}{k} = \frac{2(k-1)}{k(k+1)}$$

$$\sum_{k=3}^{n} \frac{C_{\text{moy}}(k)}{k+1} - \frac{C_{\text{moy}}(k-1)}{k} = \sum_{k=3}^{n} \frac{2(k-1)}{k(k+1)}$$

$$\Leftrightarrow \frac{C_{\text{moy}}(n)}{n+1} - \frac{C_{\text{moy}}(2)}{3} = \sum_{k=3}^{n} \frac{2(k-1)}{k(k+1)}$$

Nous allons maintenant décomposer la fraction rationnelle $F(X) = \dfrac{2(X-1)}{X(X+1)}$ en éléments simples :

$$\frac{2(X-1)}{X(X+1)} = \frac{4}{X+1} - \frac{2}{X}$$

et on a évalué XF en 0 pour trouver le coefficient associé à la valeur interdite 0 et $(X+1)F$ en (-1) pour trouver le coefficient associé à la valeur interdite -1.
Nous avons donc montré que :

$$\Leftrightarrow \frac{C_{\text{moy}}(n)}{n+1} - \frac{C_{\text{moy}}(2)}{3} = \sum_{k=3}^{n} \frac{4}{k+1} - \frac{2}{k}$$

et on a $C_{\text{moy}}(2) = 1$ car un tableau à deux éléments nécessite une seule comparaison pour effectuer la tripartition.

$$\frac{C_{\text{moy}}(n)}{n+1} = \frac{1}{3} + 4\sum_{k=3}^{n} \frac{1}{k+1} - 2\sum_{k=3}^{n} \frac{1}{k} \sim 2\ln(n)$$

et on a reconnu deux sommes harmoniques à droite.
Ainsi, on a :

$$C_{\text{moy}}(n) \sim 2n\ln(n) \approx 1.39\, n \log_2(n)$$