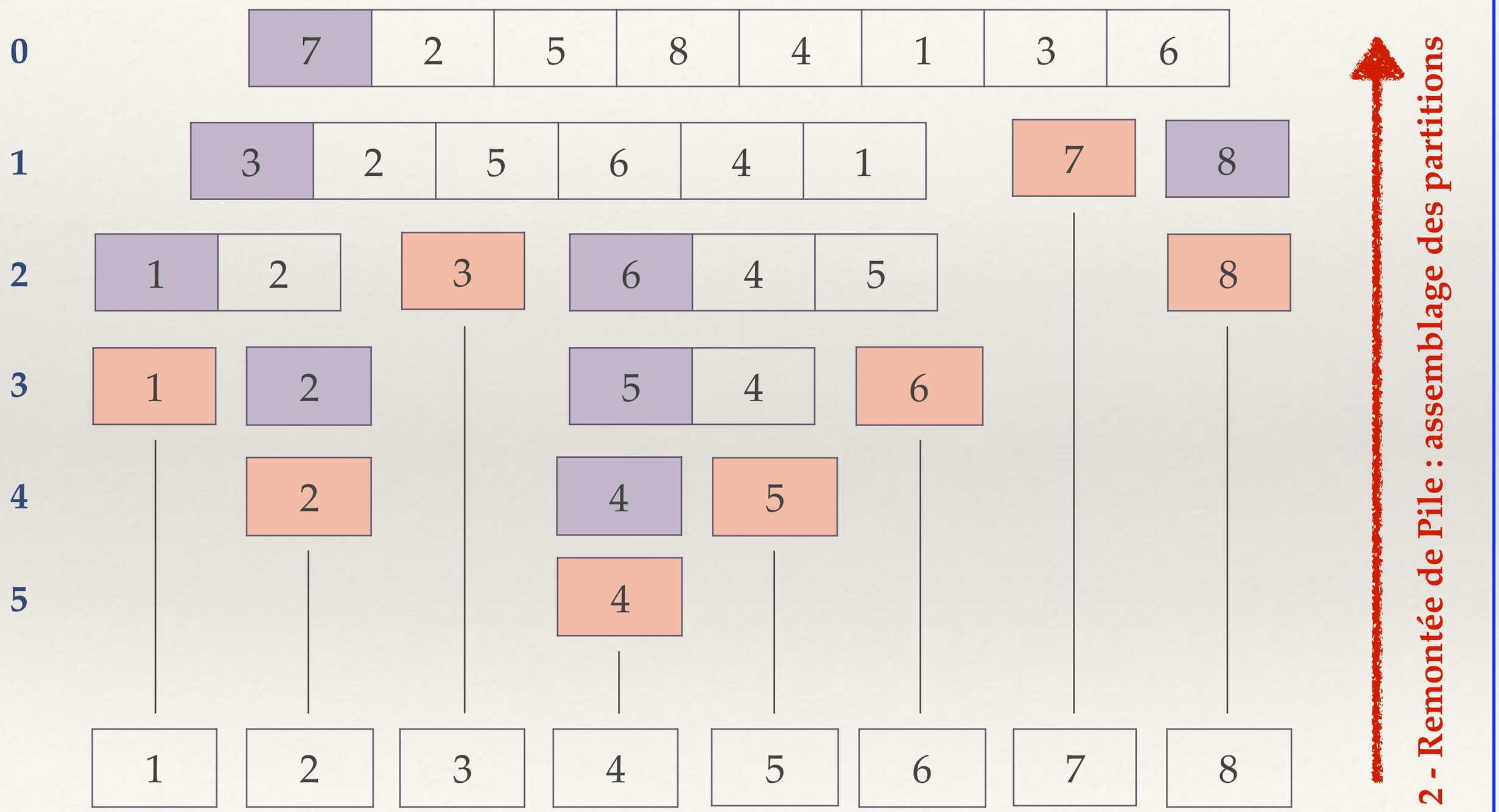
INFORMATIQUE

Tri avancé : QuickSort

4 - Le Tri Rapide

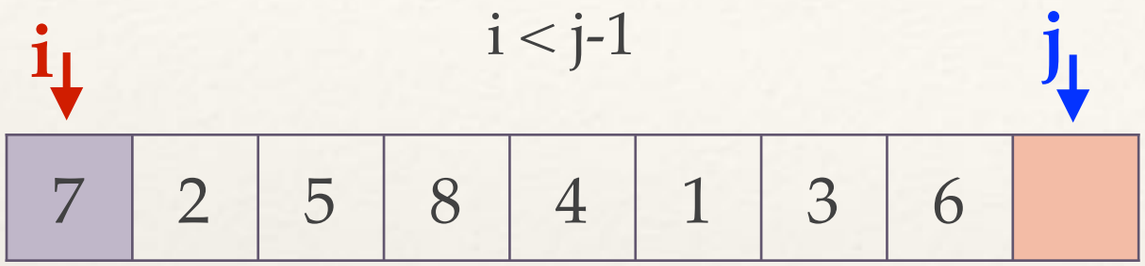
« quick-sort »

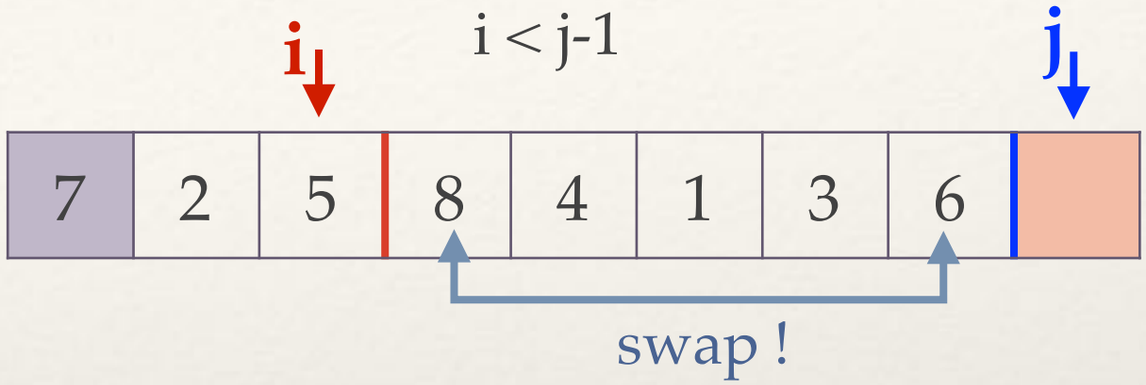
1 - Partition à la descente & tri recursifs

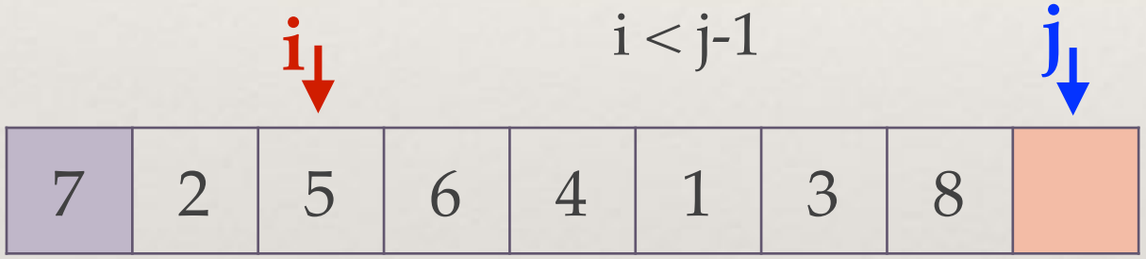


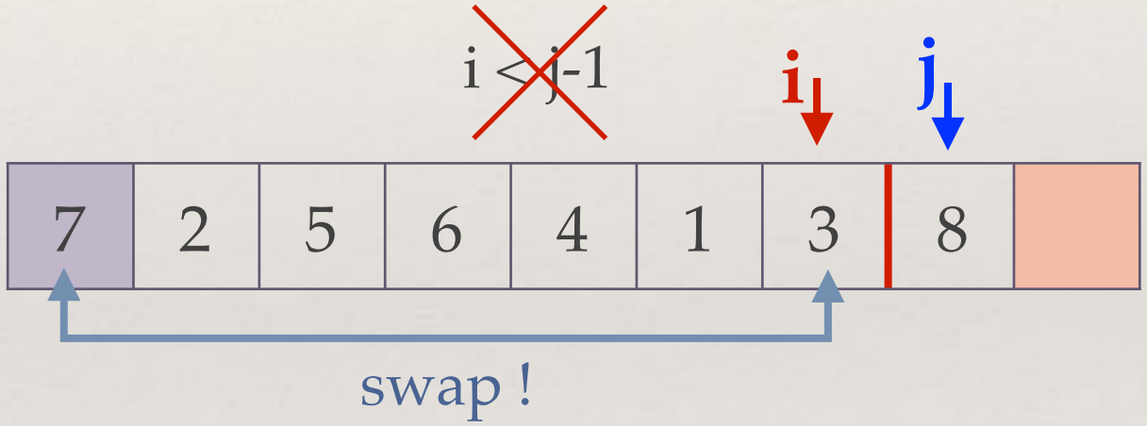
Algorithme de partition :

Etage 0 : Premier balayage de tableau => coût linéaire !



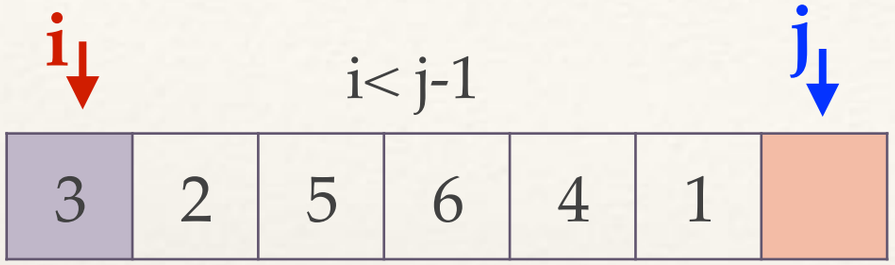


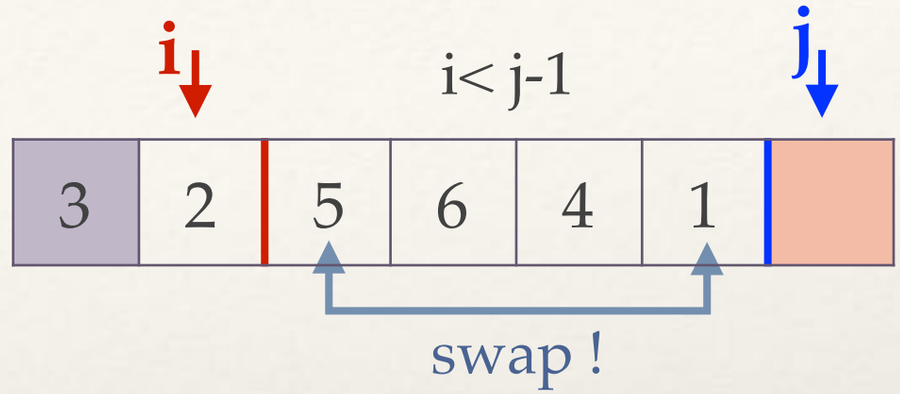


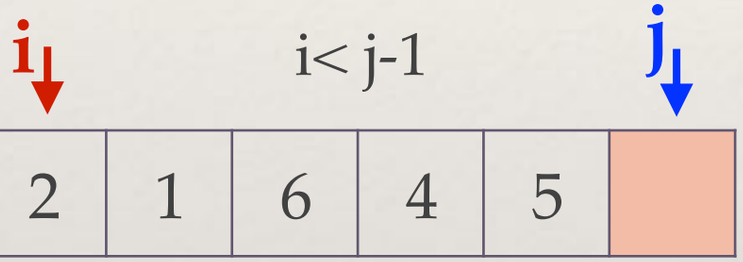


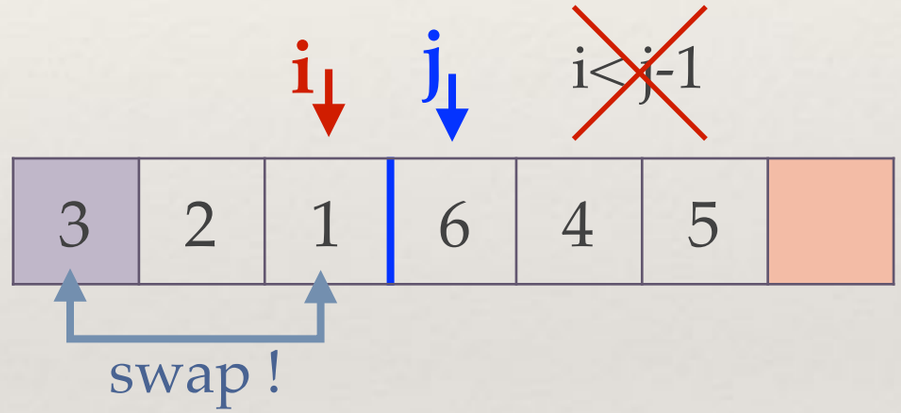
Finalisation de l'étape 1:

Etage 1 : second balayage de tableau => coût linéaire !









Finalisation de l'étape 2 :

Codage

```
def triRapide(tab):

    N=len(tab)
    if (N<=1):   #Gestion terminaison
        return

    indPivot=0;       valPivot=tab[indPivot]    #Pivot en 0
    i=0
    j=N
    while( i < j-1):            #RQ : 1ere condition empêche index error de tab[i+1]
          while ( i+1 < j and tab[i+1] <= valPivot):  #curseur i monte vers la droite
              i += 1
          while ( i < j-1 and  tab[j-1] > valPivot):      #curseur j descend vers la gauche
              j  -= 1

          if (i+1<j):
             tab[i+1], tab[j-1]=  tab[j-1], tab[i+1]

tab[0], tab[i] = tab[i], tab[0]       #On remet le pivot à sa place

triRapide(tab[0:i])           #Le tri se fait IN-PLACE => on ne renvoie rien
triRapide(tab[(i+1):])        #mais la liste a bien été triée.
```

Complexités

Meilleur des cas / Pire des cas

Coût de la partition d'un étage :

Quel que soit le nombre de sous-tableaux dans un étage, il faut passer en revue tous les éléments de chacun des sous-tableaux. Soit ~ N comparaisons

=> La partition d'un étage a toujours un coût linéaire O(N)

Nombre d'étages :

Meilleur des cas : Le pivot tombe toujours au milieu
On coupe en 2 exactement à chaque étape : Log2(N) étages.

$$\mathbb{C} = O(N.\log(N))$$

Pire des cas : Tableau déjà trié => le pivot est le 1er élément
Il y aura alors ~N étages.

$$\mathbb{C} = O(N^2)$$

Preuve

Tableau de taille 0 ou 1 : nécessairement trié

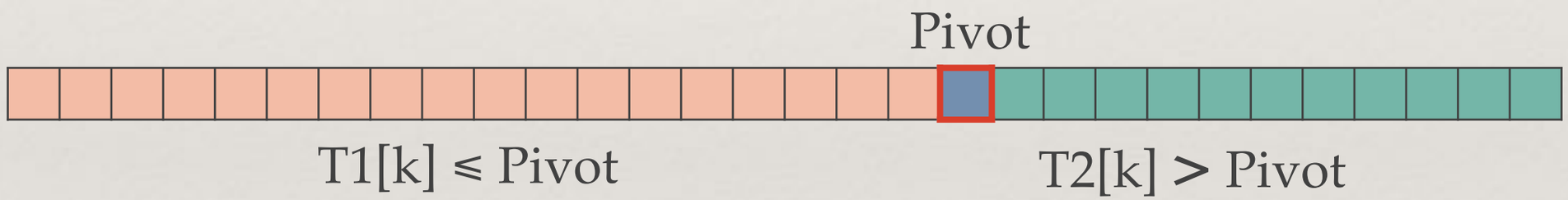
On suppose que l'algorithme fonctionne pour tout tableau de taille inférieure à n

Soit un tableau de taille n+1 :

La partition sépare le tableau en trois parties :

- un tableaux T1 de taille entre 0 et n, d'éléments inférieurs ou égaux au pivot
- le pivot seul
- un tableaux T2 de taille entre 0 et n, d'éléments supérieurs au pivot

[Rq : taille 0 et n si le pivot est le plus petit ou le plus grand élément]



Or par hypothèse le triRapide sait trier T1 < Pivot et T2 > pivot car leur taille est au plus n. Le résultat sera donc un tableau trié de taille n+1.

Par récurrence le triRapide trie tout tableau de taille n, ce qui prouve l'algorithme !

Terminaison

Soit Tn la taille des tableaux à travers les étages : $T_n \geq 0$

- Si la taille du tableau est 0 ou 1 : triRapide retourne le tableau !

- Si la taille du tableau est Tn = N : la partition donne 2 tableaux T1 et T2 dont la taille est comprise entre 0 et N-1 : $0 \leq T_{n+1} < N-1$

Ainsi pour toute descente de l'arbre de récursivité la suite (Tn) des tailles de tableaux est strictement décroissante et bornée par le bas en 0 inclus.
La descente de l'arbre de récursivité conduit inévitablement à un tableau de taille 0 ou 1 pour lequel l'algorithme termine.

Rq : Le nombre de descentes est égal au nombre de feuilles de l'arbre et est borné par N.

Dans tous les cas l'algorithme termine !

Les différences clefs entre Tri Fusion et Tri Rapide

Différences de principe

Fusion : Récursion puis Fusion
Le tri se fait lors de la remontée par Fusion

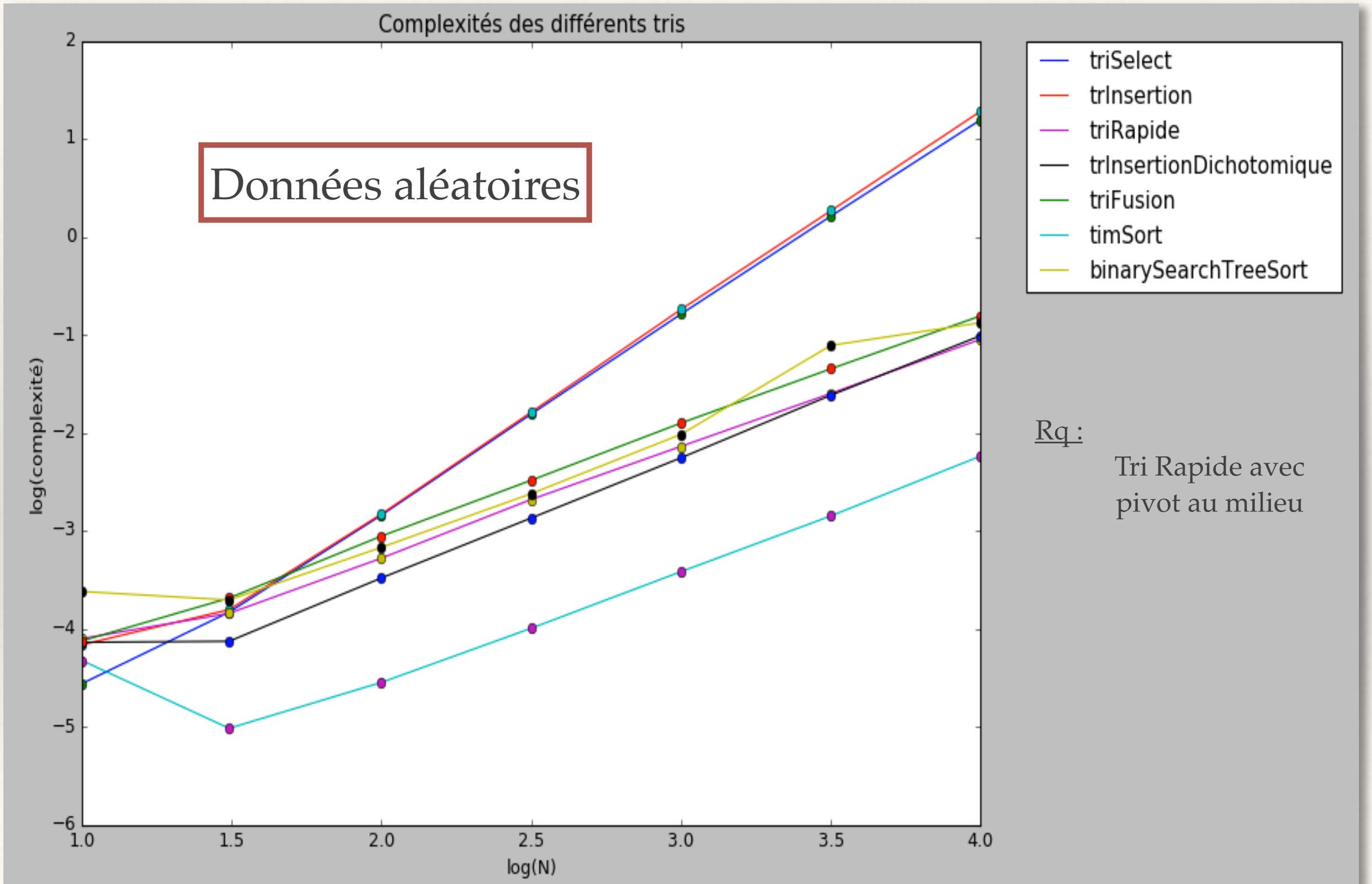
Rapide : Partition puis Récursion
Le tri se fait lors de la descente.

Différences pratiques

Fusion : **Le tri se fait par copie des valeurs dans un second tableau**
Coût spatial => deux fois la mémoire à trier
La fusion procède par « glissement » [type tri-insertion]
Le tri Fusion est stable !

Rapide : **Le tri se fait au sein du tableau « IN-PLACE »**
Pour cela on fait des commutations [type tri-sélection]
Le tri Rapide n'est pas « stable ».

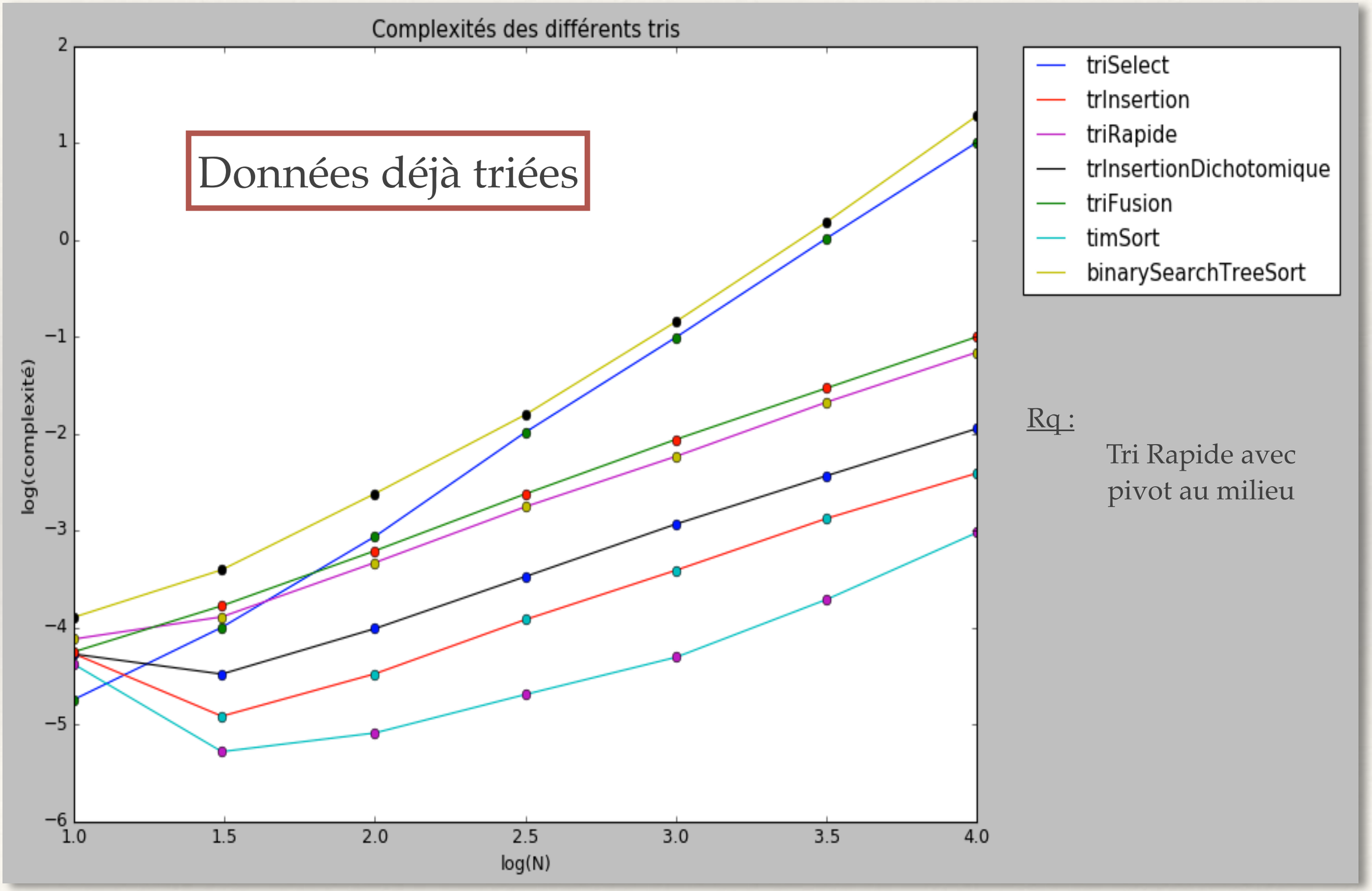
Benchmark test des tris de base



Rq :

Tri Rapide avec pivot au milieu

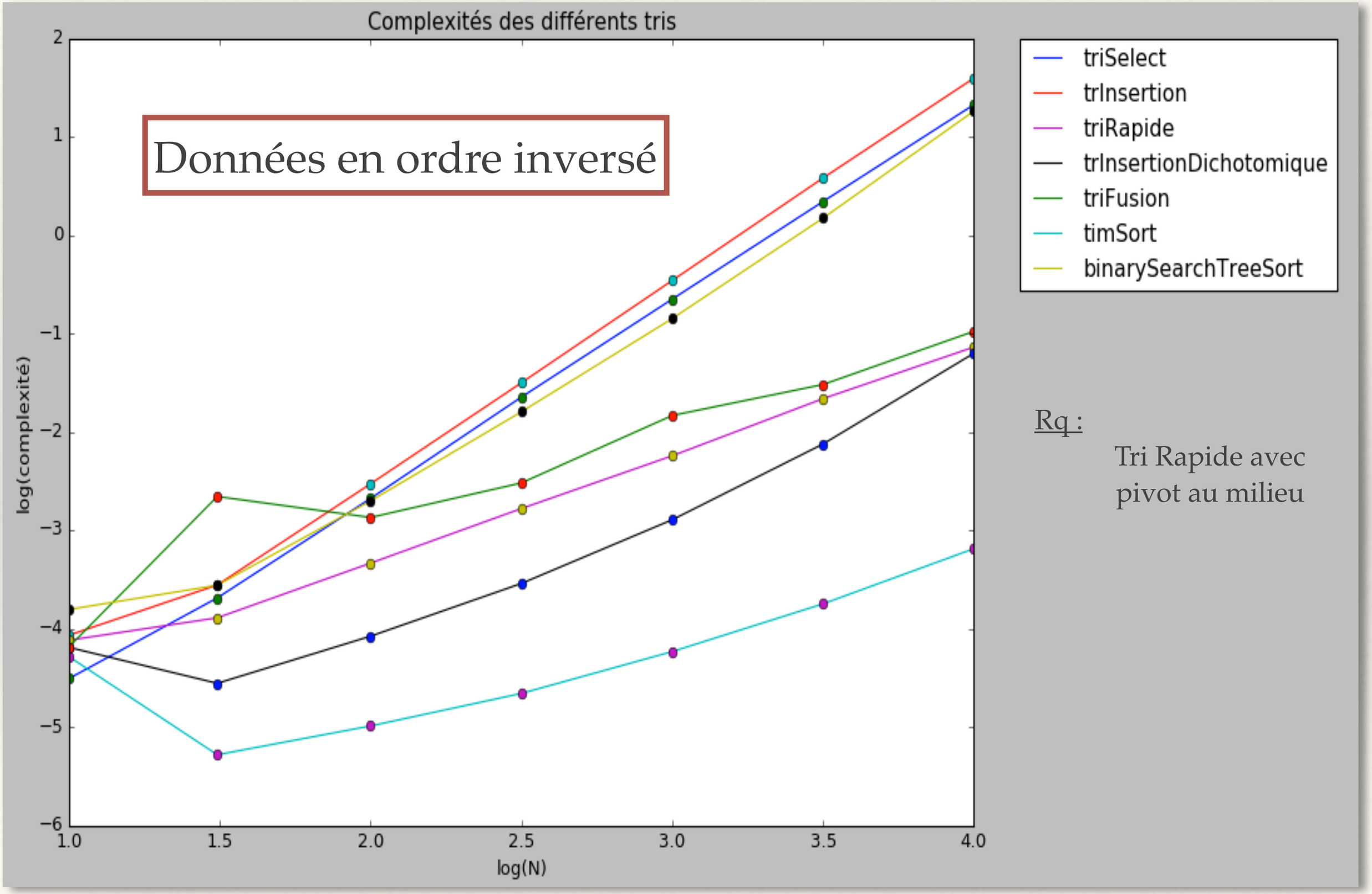
Benchmark test des tris de base



Rq :

Tri Rapide avec pivot au milieu

Benchmark test des tris de base



Rq :

Tri Rapide avec pivot au milieu

Randomized Quick-Sort

Le choix d'un Pivot est la botte secrète du Tri Rapide car il permet de mettre en place une stratégie « diviser pour régner » mais c'est aussi son talon d'Achille !

Il est paradoxalement plus rapide de commencer par mélanger le tableau pour s'assurer que le pivot partagera toujours le tableau en deux parties de même taille :

- On peut faire une première étape de mélange de coût linéaire.
- On peut choisir le pivot de façon aléatoire au lieu du premier élément.

Randomized Quick-Sort : Complexité en moyenne

Le coût global du tri correspond au nombre de comparaisons réalisées lors du tri :

Pour avoir un **coût moyen** on considère :

- Que le tableau est fixé et de taille N. C'est un tableau de valeurs quelconques.
- Que le choix des pivots (Pi) est une séquence aléatoire σ parmi celles possibles Ω.

On appelle donc ici «**complexité en moyenne**» le nombre moyen de comparaisons pour trier un tableau fixé de taille N, lorsque l'on varie aléatoirement la séquence des pivots.

Le tableau ne change pas, mais on montre que le coût ne dépend que de la taille N du tableau et pas de son contenu :

$$\mathbb{C} = O(N.\log(N))$$

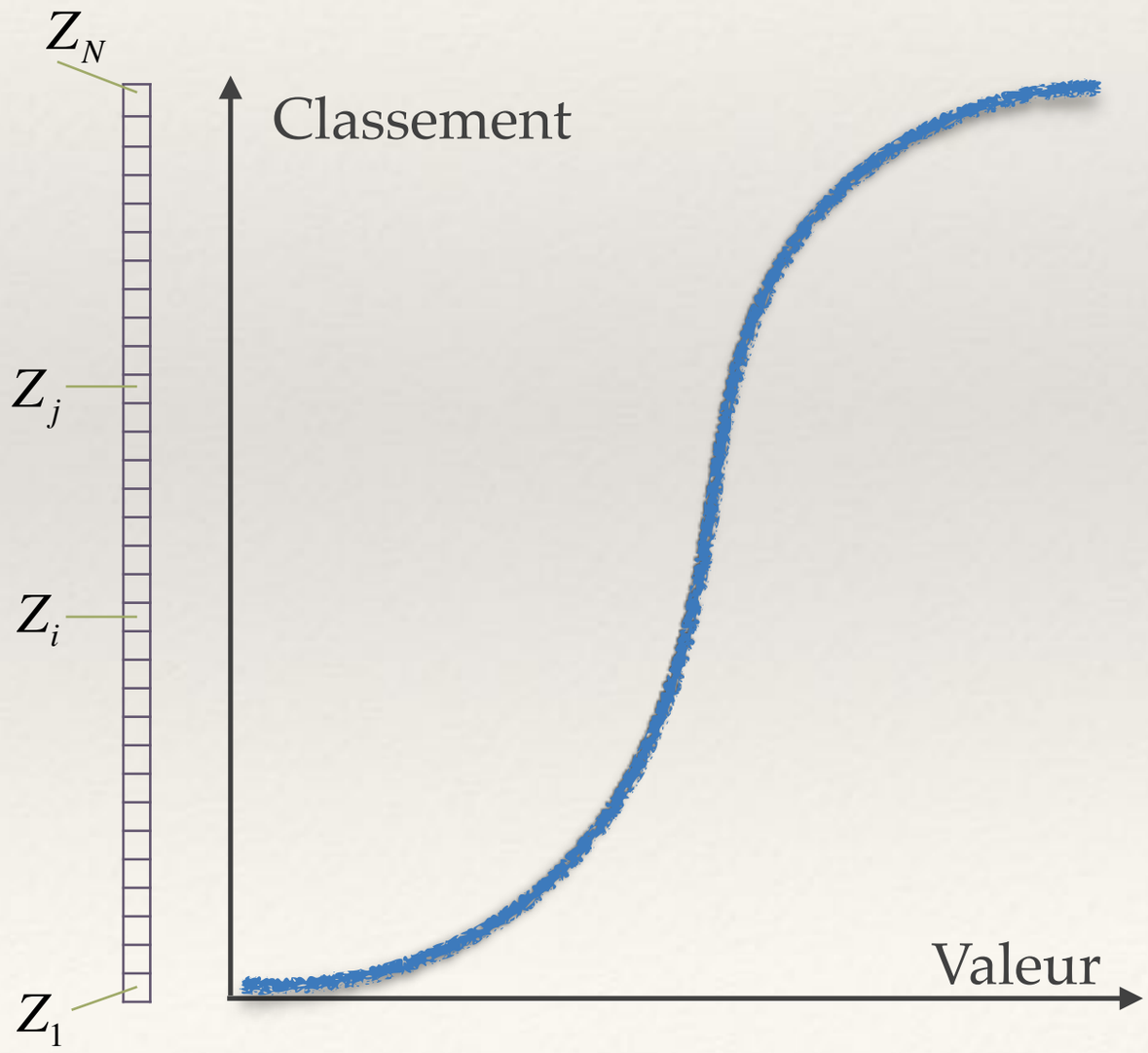
Représentation dans le tableau trié

Les indices ou positions dans le tableau n'ont pas une importance cruciale, car le pivot est aléatoire, et les éléments sont brassés à chaque récursion. Le raisonnement s'appuiera ici sur le classement des éléments tels qu'ils seraient rangés dans le tableau une fois trié :

On note pour cela Zi le i-ième plus petit élément du tableau par sa valeur, indépendamment de sa position dans le tableau :

Ex :

Z_8							Z_1
10	3	9	11	8	6	7	2
					Z_3		



Rq :
Notez bien que dans le vrai tableau les Zi sont répartis aléatoirement.
Et leur répartition change à chaque récursion.

Calcul du nombre de comparaisons

Soit i < j : on pose Xij[σ] le nombre de comparaison de Zi avec Zj lors d'une séquence aléatoire $\sigma \in \Omega$.

On ne peut comparer Zi et Zj qu'une fois au plus : **Xij[σ] vaut 0 ou 1** $\forall \sigma$

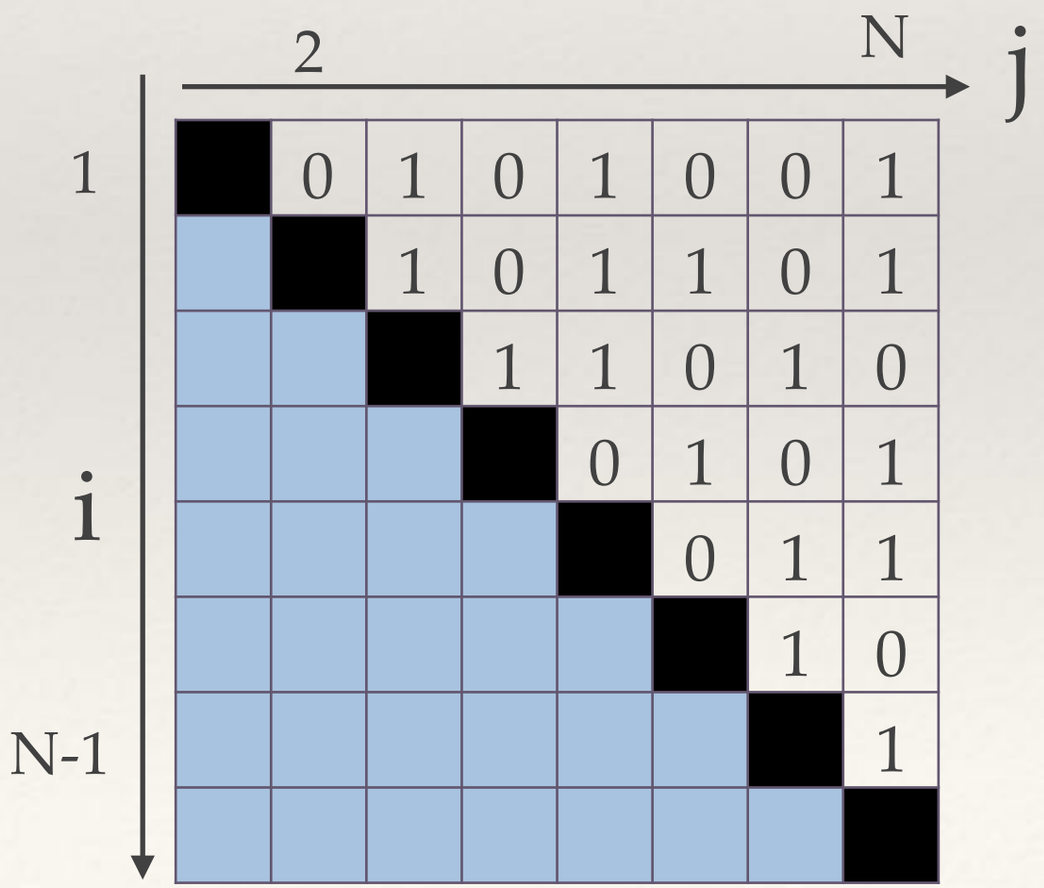
En effet seuls les pivots sont comparés :

- Soit Zi devient pivot le premier et sera éventuellement comparé à Zj, si Zj n'à pas été envoyé dans un autre paquet par un pivot précédent : auquel cas il ne seront jamais comparés !
- Soit c'est Zj qui devient pivot le premier mais le résultat est le même.

Il ne peut pas y avoir 2 comparaisons des mêmes éléments car l'un est pivot !

Le nombre total de comparaisons peut s'écrire donc facilement sur la base des Xij[σ] :

$$C[\sigma] = \sum_{i=0}^{i=N-1} \sum_{j=i+1}^{N} X_{ij}[\sigma]$$



Rq : On passe en revue toutes les paires (i,j) avec i < j.

Passage à la moyenne et calculs de probabilités

La séquence de pivots σ étant une variable aléatoire, on calcul l'espérance de C[σ] qui par linéarité de la sommation s'écrit :

$$\overline{C} = E(C[\sigma]) = \sum_{i=0}^{i=N-1} \sum_{j=i+1}^{N} E(X_{ij}[\sigma])$$

A partir de la définition de l'espérance en termes de probabilité, montrer que :

$$\overline{C} = \sum_{i=0}^{i=N-1} \sum_{j=i+1}^{N} P(X_{ij})$$

Où P(Xij) est la probabilité que Zi et Zj soient comparés lors de la séquence σ.

Comment évaluer la probabilité que Zi et Zj soient comparés lors de la séquence σ ?

On peut montrer que le problème se réduit à l'étude du segment [Zi, Zi+1, ….. , Zj] et on raisonne à nouveau dans l'espace des Zi.

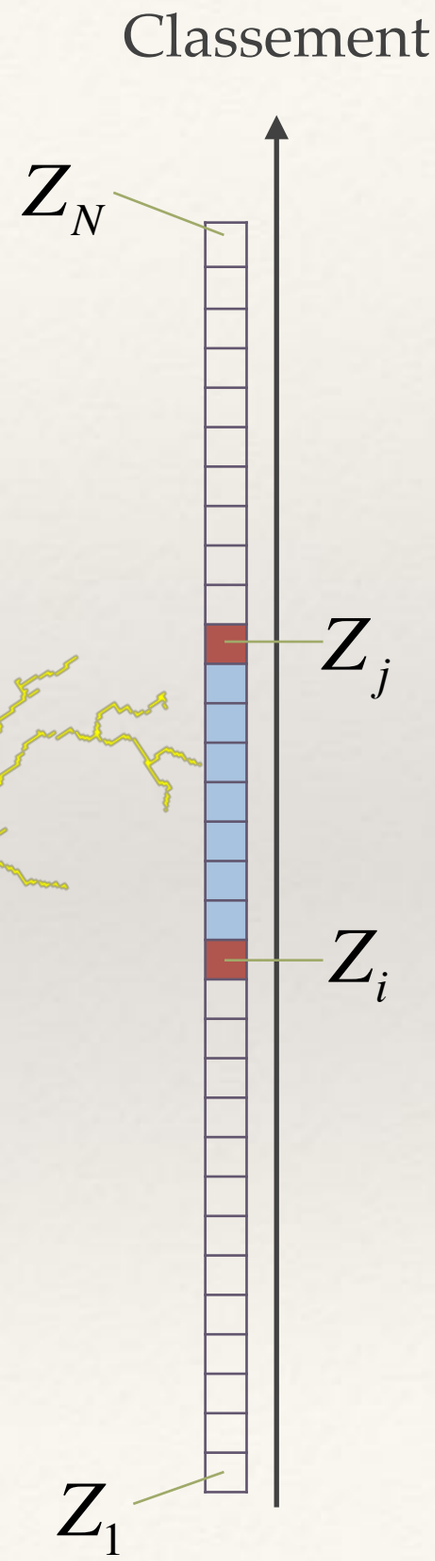
On sait en effet que le choix du pivot va tomber dans [Zi, Zj] tôt ou tard. On ne sait pas quand exactement, mais nous allons voir que cela ne détermine en rien la probabilité qu'ils soient comparés :

Tant que le pivot tombe hors de [Zi, Zj] on ne compare pas Zi et Zj.
=> Zi et Zj restent dans le même paquet. Ils ne sont pas pivot

<u>Soit (Pi) le premier pivot qui tombe dans de [Zi, Zj] :</u>

- Soit le pivot est Zi ou Zj : il y aura comparaison
- Soit le pivot n'est ni Zi ni Zj : il n'y aura jamais de comparaison

(Ils sont séparés par le pivot.)



Enfin, quel que soit le tableau qui contient les [Zi, Zj] à cet instant, ils y sont tous, et sont tous des pivots équiprobables.

Point clef à retenir :

Tout se joue au moment ou la foudre tombe entre Zi et Zj
Ce qui se passe avant ou après ni changera rien

Quelle est donc la probabilité qu'il y ait comparaison <u>sachant que</u> le pivot est dans l'intervalle [Zi, Zj] ?

Finalisation

En majorant la somme des inverses par la fonction ln(), obtenir le résultat escompté.

$$\mathbb{C} = O(N.\log(N))$$

La majoration est en N.Log(N) comme le meilleur des cas, donc la complexité moyenne est en N.log(N).