INFORMATIQUE

ALGORITHMES DE TRI

I Introduction

La séquence précédente portait sur l'étude comparative des algorithmes de tri à coût quadratique. Rappelons qu'un coût quadratique signifie qu'un doublement de la dimension de l'objet à traiter multiplie par 4 le nombre d'opérations c'est à dire le temps de traitement. Dans le cadre de cette première étude, le coût calculatoire est estimé à partir d'un ensemble de réalisations dans le but de réduire la variance de l'estimation. Le résultat est finalement comparé à son expression théorique et complété par une estimation de la densité de probabilité associée à la variable aléatoire temps d'exécution. Cette nouvelle séquence a pour objectif l'étude de deux algorithmes de tri à coût quasi-linéaire soit en $O(nlogn)$. Un protocole d'étude semblable au TP 10 sera mis en place.

II Du « déjà vu » mais en version récursive : le tri par sélection

Un algorithme peut s'écrire aussi bien en itératif qu'en récursif, toutefois la procédure récursive est rarement plus efficace que le traitement itératif mais ce n'est pas la seule raison, … Illustrons ce postulat sur l'exemple du tri par sélection.

Rappelons le principe du tri par sélection :

1- Déterminer l'élément minimum du tableau T qui est échangé avec le premier élément (du tableau).
2- On recommence 1- avec les éléments restants d'indices 2 à n puis avec les éléments d'indice 3 à n … jusqu'aux éléments d'indices $n-1$ puis n.

La récursivité apparait alors clairement.

Lors du précédent TP, vous vous êtes assurés que cette fonction modifie la liste T en place passée en argument par effet de bord, un « return » n'est donc pas nécessaire puisque la liste T apparaîtra triée dans l'espace des variables.

Concernant la version récursive, rappelons d'une part que si T est une liste on obtient la liste privée de l'élément $T[m]$ avec la syntaxe $T[:m]+T[m+1:]$ et que d'autre part, la clause d'arrêt est prioritaire dans le traitement, elle correspond à la liste vide ou de taille 1 passée en argument.

APPLICATION

Q1. Proposer une version récursive de l'algorithme du tri par sélection.
La liste initiale est-elle modifiée par effet de bord comme c'est le cas pour la version itérative ?

Soit une liste de taille N, on note c_N un majorant du coût calculatoire de l'algorithme récursif pour traiter cette liste.

APPLICATION

Q2. Vérifier que le coût calculatoire suit la récurrence suivante : $c_N = c_{N-1} + N - 1$.
En déduire que le coût de la version récursive est quadratique comme sa version itérative.

III Algorithmes de tri quasi-linéaire

3-1 Le tri fusion

3-1-1 Principe et illustration

Le principe général de la méthode de tri fusion pour un tableau T est la suivante :

1- On coupe en deux parties les données à trier.

2- Les données de chaque partie sont triées par la méthode de tri fusion.

3- Les deux parties triées sont triées.

L'algorithme est récursif. Il fait partie des algorithmes qui reposent sur le paradigme « diviser pour régner ». La récursivité s'arrête lorsque la liste terminale d'une branche de l'arbre est constituée d'un seul élément. Ce principe est illustré par la figure 1 ci-dessous :

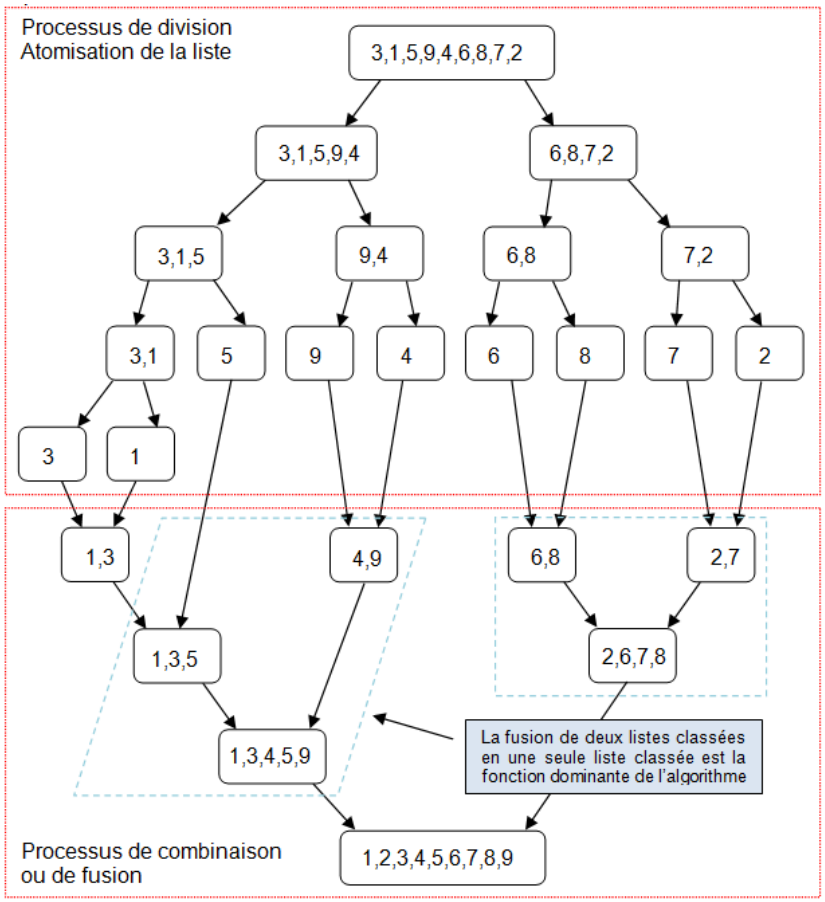


Figure 1 : principe du tri fusion

Le problème se décline désormais en un problème de trie de listes déjà triées. Comment peut-on trier simplement des listes déjà triées ?

Considérons deux tas qui représentent deux listes triées. Il s'agit de prendre la plus petite valeur placée aux sommets des deux tas et de construire un troisième tas (en rouge sur la figure 2). Lorsque l'un des deux tas est vide, le reste est retourné puis placé sur ce troisième tas.

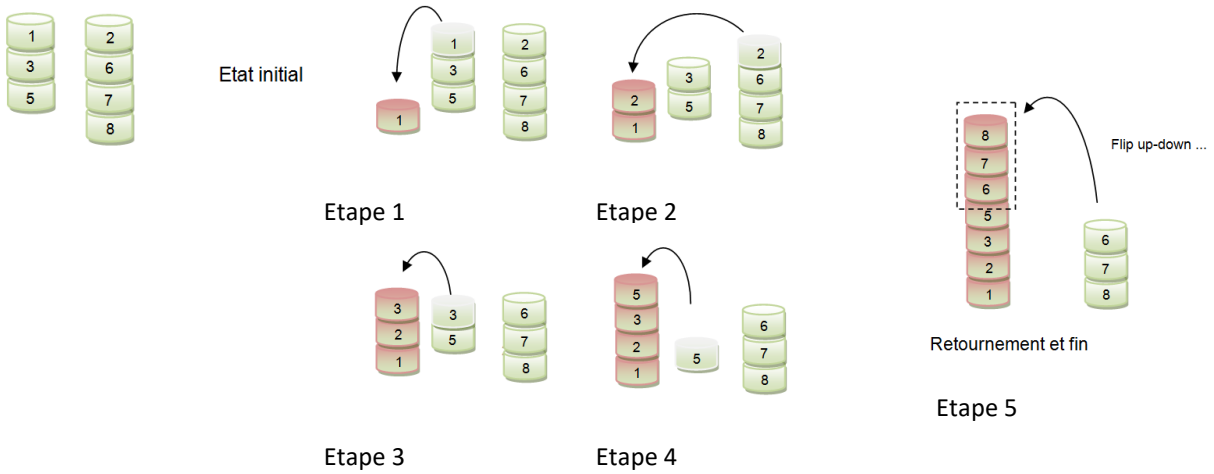


Figure 2 : principe de tri de deux listes triées

3-1-2 Modèle et pseudocode

L'algorithme de tri fusion se décline en deux fonctions, une que l'on peut qualifier de principale nommée Tri. Elle fait appel à une seconde nommée Fusion qui porte l'algorithme de fusion de deux listes triées présenté précédemment.

Algorithme : fusion
{Cet algorithme permet de fusionner deux listes triées}
Arguments passés en paramètre
Un tableau T de n éléments à trier : $T = [e_0, e_1, \dots, e_{n-1}]$
p, q, r qui sont les indices p, q et $q+1, r$ du tableau T des deux listes triées qui sont à fusionner.
La function fusion retourne la fusion en place, dans la liste T entre les indices p et r, des deux listes triées.

	Pseudo-code	$\times fois$	commentaires
1	$Fusion(T, p, q, r)$		
2	**Début**		
3	$n_1 = q - p + 1$	1	# Longueur de la liste $T[a_p, \dots, a_q]$
4	$n_2 = r - q$	1	# Longueur de la liste $T[a_{q+1}, \dots, a_r]$
5	# La longueur de la liste est $n_1 + n_2 = r - p + 1$		
6	**pour** $i = 0 \text{à} n_1 - 1$ **faire** :	n_1	*# recopie* de la liste $T[a_p, \dots, a_q]$ dans L
7	$L[i] = T[p + i - 1]$		
8	**pour** $j = 0 \text{à} n_2 - 1$ **faire** :	n_2	*# recopie* de la liste $T[a_{q+1}, \dots, a_r]$ dans R
9	$R[j] = T[q + j]$		
10	$L[n_1 + 1] = \infty$ **et** $R[n_2 + 1] = \infty$	1	# test d'arrêt
11	$i = j = 1$	1	# initialisation de deux compteurs
12	**pour** $k = p, \dots, r$	$r - p + 1$	# trie des deux listes classées
13	**si** $L[i] < R[j]$		
14	$T[k] = L[i]$ et $i = i + 1$		
15	**sinon**		
16	$T[k] = R[j]$ et $j = j + 1$		
17	**fin**		

Considérons une liste T de n éléments indexés de 1 à n et les deux partitions triées adjacentes indexées de p à q et de $q+1$ à r :

$$T = \left[a_1, \dots, a_{p-1}, \underbrace{a_p, \dots, a_q}_{listetriée}, \underbrace{a_{q+1}, \dots, a_r}_{listetriée}, a_{r+1}, \dots, a_N\right]$$

$Fusion(T,p,q,r)$ est une fonction qui fait un tri « en place » des deux listes triées indexées de p à q et de $q+1$ à r et qui retourne la fusion de ces deux listes triées en place entre les index p et r dans T. Il faut remarquer une petite « subtilité » de codage à la ligne 10 qui évite les tests de longueur de liste. Ainsi, lorsque la première des deux listes est vide, l'élément suivant est codé ∞ ce qui permet le basculement vers la liste restante et son reversement, empilement sur le troisième tas.

$Tri(T,p,r)$ est la fonction d'appel récursive de $Tri(T,p,q)$ et $Tri(T,q+1,r)$ jusqu'à la liste triviale d'un élément à trier puis qui fusionne les listes triées lors de son processus de backtracking.

Algorithme : Tri fusion
{Cet algorithme est la fonction principale de la l'algorithme de tri}
Arguments passés en paramètre
Un tableau T de n éléments à trier : $T = [e_0, e_2, \dots, e_{n-1}]$
p, r qui sont les index du tableau T. Initialement, $p = 0$ et $r = n-1$.

	Pseudo-code	$\times fois$	commentaires
	$Tri(T,p,r)$	$C(r-p+1)$	# Si le coût de la fonction Tri est $C(N)$,…
	Début		
	si $p < r$		# test d'arrêt
	$q = \lfloor (p+r)/2 \rfloor$		# division de la liste en deux listes
	$Tri(T,p,q)$	$C(\lfloor (r-p+1)/2 \rfloor)$	# … alors le tri de la demie liste est $C(\frac{N}{2})$
	$Tri(T,q+1,r)$	$C(\lfloor (r-p+1)/2 \rfloor)$	# idem pour la demie liste supérieure
	$Fusion(T,p,q,r)$	$r-p+1$	# fusion des deux demies listes triées en une liste triée avec un coût de N.
	fin		
	fin		

Complexité en temps : Rappelons que le coût de cet algorithme est solution de la récurrence $C(N) = 2C(N/2) + N$. Sous réserve de l'hypothèse d'une liste de N éléments telle que $N = 2^p$, le coût est en $\mathcal{O}(Nlog_2(N))$.

3-2 Tri rapide

3-2-1 Principe et illustration

Considérons la liste $T = [3,2,5,9,4,6,8,7,1]$

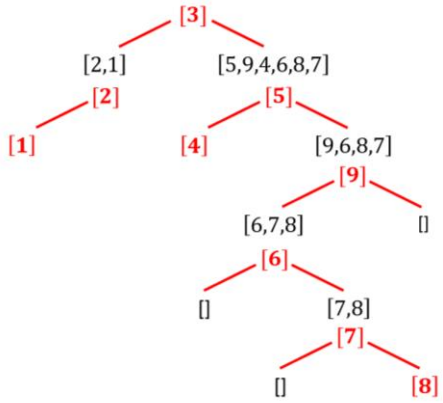
1- Choisissons comme pivot, par exemple, le premier élément soit 3.

2- La liste T est réordonnée de la façon suivante : $T = [2,1,3,5,9,4,6,8,7]$

Les éléments à gauche du pivot lui sont inférieurs alors qu'à droite, ils lui sont supérieurs.

Objectif : trier maintenant les deux sous-listes pour trier complètement la liste.

L'arbre suivant représente les appels successifs de la fonction qui met à sa place définitive le pivot dans la liste à trier :



3-2-2 Modèle et pseudocode

Algorithme : Tri rapide

{Cet algorithme permet de trier un tableau T par une méthode de tri rapide}

Argument passé en paramètre

Un tableau T de n éléments à trier : $T = [e_0, e_1, \dots, e_{n-1}]$

	Pseudo-code	commentaires
2	**Début**	
	$n \leftarrow len(T)$	
	si $len(T) \leq 1$ **faire**	# test d'arrêt
	$retourne\ T$	# Processus de backtracking
	sinon	
	$L1, L2 \leftarrow [\ \ ]$	# Init des deux liste qui encadreront le pivot
	$pivot \leftarrow T[0]$	# Le premier élément est choisi pivot
3	**pour** $i = 0, \dots, len(T) - 1$	# Les éléments restants sont placés tels que, …
4	**si** $T[i] \leq pivot$	
5	$L1 \leftarrow T[i]$	# Dans $L1$ ils sont pivot
	sinon	
6	$L2 \leftarrow T[i]$	# Dans $L2$ ils sont pivot
	$T \leftarrow [Tri\ rapide(L1),\ [pivot], Tri\ rapide(L2)]$	# Le pivot est en place, recursion sur $L1$ et $L2$
	retourne T	
7	**fin**	

Le coût de cet algorithme est meilleur que le précédent mais toutefois en $\mathcal{O}(Nlog_2(N))$. Dans une situation de pire cas il est en $O(n^2)$.

Complexité en temps : Rappelons que le coût de cet algorithme est en moyenne solution de la récurrence $C(N) = 2C(N/2) + N$. Sous réserve de l'hypothèse d'une liste de N éléments telle que $N = 2^p$, le coût est en $\mathcal{O}(Nlog_2(N))$ mais dans le pire cas il est quadratique. Cette situation, contre toute attente, correspond à une liste triée (cours).

APPLICATION

Q3. Pour chacun des pseudocodes présentés, écrivez les fonctions associées $Tri(T,p,r)$, $Fusion(T,p,q,r)$ et $tririapide(T)$.
Ces fonctions ont a minima pour argument d'entrée un tableau T constitué de n éléments de type entier et retournent ce tableau passé en entrée trié.

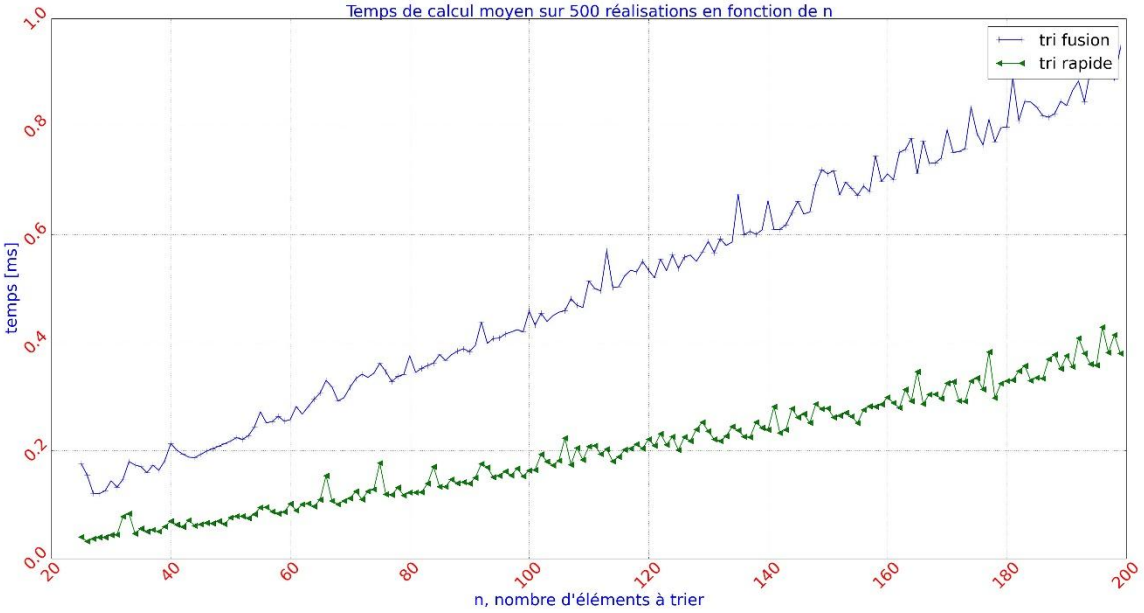


Figure 3 : Temps d'exécution moyen comparé obtenu pour 500 réalisations

L'estimation du temps d'exécution d'un algorithme de tri est une fonction de l'arrangement des éléments dans la liste passé en paramètre. Afin de réduire la variance de l'estimation, il est donc souhaitable de moyenner un grand nombre de réalisations.

Attention aux effets de bord, …

APPLICATION

Q4. Ecrivez une fonction qui permet d'estimer le coût calculatoire moyen obtenu à partir « d'un grand nombre de réalisations » pour chacun des algorithmes avec une entrée liste de dimension $n = 20, \ldots, 200$. Représenter les résultats sur un même plot comme représenté sur la figure 3.

APPLICATION

Q5. Comparer ces résultats avec ceux obtenus pour les algorithmes à coût quadratique puis conclure.
Il serait judicieux de les représenter sur un même plot.