École Supérieure de Technologie et d'Informatique
Département d'Informatique Appliquée
__***__

Examen de la session principale en

Algorithmique Avancée & Complexité

8 janvier 2014

Classe : $2\text{IngInfo}_{A,B,C,D,E}$ Durée : $1^h30'$ Nombre de pages : 3 pages

Il est conseillé de lire attentivement les énoncées avant de se plonger sur la feuille des réponses. ***Les copies propres et bien soignées sont très appréciées. Il sera tenu compte de la lisibilité des réponses****. Le barème donné est indicatif. Aucun document n'est autorisé.*

Bonne chance !

Problème I

Le but de ce problème est d'étudier les différentes variantes de la famille d'algorithmes de tri Rapide (`QuickSort`). Il est question aussi de déterminer le nombre optimal de sous-problèmes (partitions) à produire à chaque appel récursif.

L'algorithme de tri rapide basique (`QuickSort`) se déroule comme suit : nous partons d'un tableau de N entiers qui ne sont, initialement, pas triés. Pour trier le tableau nous procèdons de manière récursive.

À chaque niveau de la récursivité, nous sélectionnons (suivant une stratégie donnée) une valeur particulière parmi les entiers que nous désigons comme « `pivot` ». Par la suite, toute les valeurs dans le tableau sont comparées avec le pivot. Nous pouvons, ainsi placer le pivot à sa position définitive dans le tableau en copiant les valeurs inférieures au pivot à gauche du pivot (au début du tableau) et les valeurs supérieures au pivot à droite du pivot (à la fin du tableau) comme dans la figure.

$\leq$ Pivot	Pivot	Pivot $\leq$

De cette manière nous avançons un pas vers le tri du tableau. Il suffit par la suit de répéter ce traitement de manière récursive pour les deux moitiés du tableau : c'est-à dire choisir un pivot pour chaque sous-tableau, comparer tous les éléments du sous-tableau au pivot, et mettre le pivot dans sa position définitive.

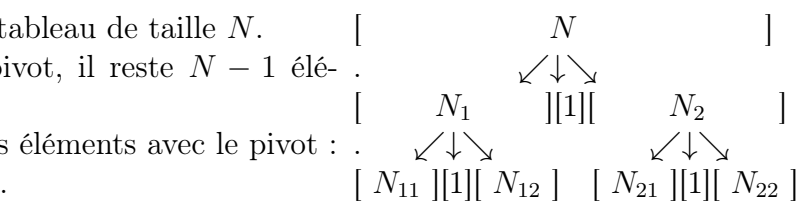
La procédure récursive se répète jusqu'à aboutir à des sous-tableaux de taille unitaire.

Un sous-tableau de taille unitaire est supposé être trié et ne demande donc aucun effort de tri et par conséquent c'est tout le tableau qui est trié.

Nous rappelons que la complexité des algorithmes de tri est calculée en terme d'opérations de comparaison.

Nous rappelons aussi que si n désigne la taille d'un problème donné, le remplacement des termes $\frac{n}{b}$ par $\lfloor \frac{n}{b} \rfloor$ ou $\lceil \frac{n}{b} \rceil$ n'affecte pas le comportement asymptotique de la récurrence. On omet donc en général les parties entières.

Nous partons d'un tableau de taille N.
Nous prenons un pivot, il reste $N-1$ éléments.
Nous comparons ces éléments avec le pivot : (N-1 comparaisons).
Nous mettons les éléments inférieurs au pivot d'un coté (soit N_1 ele^t) et les éléments supérieurs de l'autre (soit N_2 éléments).
tels que $N = N_1 + N_2$
Nous appliquons le m̂ principe aux deux tableaux
$N_1 = N_{11} + N_{12}$
$N_2 = N_{21} + N_{22}$
... etc.

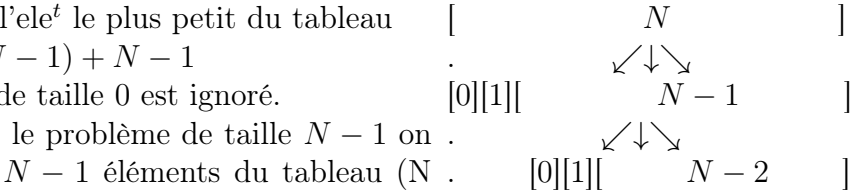


Partie A. Pire et meilleur des cas. (5 pts)

Pour calculer la complexité du tri rapide au pire et au meilleur des cas, considérons les deux cas extrêmes.

1. Supposons que nous choisissons, à chaque fois, comme pivot la valeur la plus petite dans le tableau. Ainsi, le sous-tableau qui contient les éléments inférieurs au pivot sera vide et par la suite tous les éléments du tableau (sauf le pivot) se trouveront dans le sous-tableau des éléments supérieurs au pivot.
   a. Énoncer une équation de récurrence qui décrit ce sénario.

Le pivot est l'ele^t le plus petit du tableau
$T(N) = T(N-1) + N - 1$
le problème de taille 0 est ignoré.
Pour obtenir le problème de taille $N-1$ on compare les $N-1$ éléments du tableau (N elet sauf le pivot) au pivot.



   b. Calculer la complexité en développant cette équation.

$T(N) = T(N-1) + N - 1$
$= T(N-2) + N - 2 + N - 1$
$= T(N-3) + N - 3 + N - 2 + N - 1$
...
$T(1) + 1 + 2 + 3 + \ldots + N - 1$
d'après les énoncées un tableau unitaire est trié et ne demande aucun effort

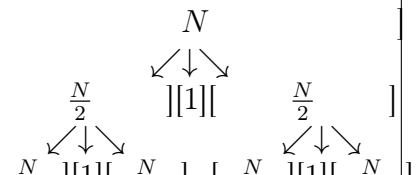
de tri $==>T(1)=0$.
$T(N)=\sum_{i=1}^{N-1}=\frac{N(N-1)}{2}=\frac{1}{2}(N^2-N)$

c. Donner une estimation asymptotique en notation $\mathcal{O}$ de cette complexité.

$T(N)=\frac{1}{2}(N^2-N)=\mathcal{O}(N^2)$

2. Supposons, maintenant, que le pivot soit à chaque fois la valeur médiane du tableau : c'est-à-dire le pivot permet de partionnner le tableau en deux sous-tableaux de même taille.
   a. Énoncer une équation de récurrence qui décrit ce sénario.

Le pivot est l'élément médian du tableau.
On obtient 2 tableaux de taille $(N-1)/2$ chacun.
Or, $(N-1)/2=\lceil N/2\rceil$ ou $\lfloor N/2\rfloor$ selon la parité de N.
Les aprties entières n'affectent pas le comportement asymtotique de la récurrence. Nous retenons alors la notation $N/2$
$T(N)=2T(N/2)+N-1$



   b. Calculer la complexité en développant cette équation.

$T(N)=2T(\frac{N}{2})+N-1$
$=2(2T(\frac{N}{4})+\frac{N}{2}-1)+N-1$
$=2(2(2T(\frac{N}{8})+\frac{N}{4}-1)+\frac{N}{2}-1)N-1$
$\vdots$
$=\underbrace{2^pT(1)}_{=0}+\sum_{i=0}^{p-1}N-2^i\ /\ p=log_2N==>2^p=N$
$=pN-2^p=N.log_2N-N+1$

   c. Donner une estimation asymptotique en notation $\mathcal{O}$ de cette complexité.

$T(N)=N.log_2N-N+1=\mathcal{O}(N.logN)$

   d. Retouver ce même résultat en appliquant le théorème de résolution des récurrences.

$f(N)=N-1=\mathcal{O}(N)=\mathcal{O}(N^1)$
$N^{log_ba}=N^{log_22}=\mathcal{O}(N^1)$
$==>f(N)=\Theta(N^1)$
On est au 2ème cas tu théorème.
$==>T(N)=\Theta(N.logN)$

3. Identifier le pire et le meilleur des cas.

Le premier cas est le pire des cas, et le deuxième cas est le meilleur.

Partie B. Complexité du k-`wayQuickSort`. (5 pts)

Comme nous considèrons, en générale, que les éléments à trier ont des valeurs très aléatoires, nous supposons, alors, que des pivots qui sont choisis de manières aléatoires permettent de partitionner les éléments en des sous-tableaux de même taille. Nous Supposons alors dans la suite de ce problème que les sous-tableaux produits après la phase de partitionnement sont de même taille.

D'autres variantes de tri rapide ont été conçues en augmentant le nombre de pivots considérés à chaque niveau.

Dans la variante 3-`wayQuickSort` nous sélectionnons deux pivots qui sont comparés à tous les éléments du tableau pour les placer dans leurs positions définitves.

Ainsi le tableau est partitionné en 3 sous-tableaux. Si P_1 et P_2 sont deux pivots tels que $P_1 < P_2$ alors :

— les éléments inférieures à P_1 sont placés à gauche de P_1 dans le tableau.
— les éléments compris entre P_1 et P_2 sont placés entre P_1 et P_2 dans le tableau.
— les éléments supérieurs à P_2 sont placés à droite de P_2 dans le tableau (voir la figure).

$\leq P_1$	P_1	$P_1 \leq \ldots \leq P_2$	P_2	$P_2 \leq$

4. Dans le cas d'un k-`wayQuickSort`, Nous utilisons $k-1$ pivots à chaque niveau pour partionner le tableau en k sous-tableaux (sous problèmes). Chaque élément du tableau est comparé (au pire des cas) à tout les pivots pour pouvoir l'insérer dans un sous-tableau particulier.
   a. Énoncer l'équation de récurrence qui permet de décrire la complexité du k-`wayQuickSort`.

On séléctionne $(k-1)$ pivots. Il nous reste $(N-k+1)$ éléments dans le tableau. On comprae les $(N-k+1)$ éléments aux $(k-1)$ pivots $==>$ $(N-k+1)(k-1)$ comparaisons.
On obtient k sous tableaux de taille $\frac{N}{k}$ chacun.
$T(N) = kT(\frac{N}{k}) + (N+1-k)(k-1)$

   b. Développer cette équation et déterminer le nombre exacte de comparaisons effectuées.

$$
\begin{aligned}
&T(N) = k\big(kT(\tfrac{N}{k^2}) + (\tfrac{N}{k}+1-k)(k-1)\big) + (N+1-k)(k-1)\\
&= k\Big(k\big(kT(\tfrac{N}{k^3})+(\tfrac{N}{k^2}+1-k)(k-1)\big)+(\tfrac{N}{k}+1-k)(k-1)\Big)+(N+1-k)(k-1)\big)\\
&\vdots\\
&= \underbrace{k^pT(1)}_{=0} + \textstyle\sum_{i=0}^{p-1} k^i(\frac{N}{k^i}-k+1)(k-1), \text{ tel que } p = log_kN \text{ et } k^p = N\\
&= (k-1)\Big(pN - (k-1)\textstyle\sum_{i=0}^{p-1} k^i\Big)
\end{aligned}
$$

$= (k-1)\Big(pN - (k-1)\frac{k^p-1}{k-1}\Big)$
$= (k-1)(pN - N + 1)$
$= (k-1)(N.log_k N - N + 1)$

c. Donner une estimation asymptotique en notation $\mathcal{O}$ de cette complexité.

$T(N) = (k-1)(N.logN - N + 1) = \mathcal{O}(N.log_k N)$

d. Retourver cette même estimation asymptotique en appliquant le théorème de résolution des récurrences.

$f(N) = (N + 1 - k)(k-1) = \mathcal{O}(N^1)$
$N^{log_b a} = N^{log_k k} = N^1$
On est toujours au 2eme cas du théorème : $T(N) = \Theta(N.log_k N)$

Partie C. Détermination de la valeur optimale de k. (5 pts)

5. Nous cherchons maintenant le nombre de pivots (k) optimal qui permet de minimiser le nombre de comparaisons effectuées par le k-`wayQuickSort`.
   a. Vérifier que nous trouvons la complexité du tri rapide de base quand nous remplaçons k par 2 dans la formule trouvée dans (`4.a.`).

La complexité du k-wayQuickSort : $T(N) = (k-1)(N.log_k N - N + 1)$
pour $k = 2$, $T(N)$ devient $N.log_2 N - N + 1$ (voir 2.b.)

   b. Pour déterminer la valeur optimale de k, donner la complexité des 2-`wayQuickSort`, 4-`wayQuickSort` et 16-`wayQuickSort` en remplaçant dans la formule de (`4.a.`) k par la valeur appropriée.

- $k = 2 : T(N) = 2T(\frac{N}{2}) + N - 1 ==> T(N) = Nlog_2 N - N + 1$
- $k = 4 : T(N) = 4T(\frac{N}{4}) + (N-3) \times 3 ==> T(N) = 3Nlog_4 N - N + 1$
- $k = 16 : T(N) = 16T(\frac{N}{16}) + (N-15) \times 15 ==> T(N) = 15(Nlog_{16} N - N + 1)$

   c. Trouver une relation entre $log_2 N$ et $log_4 N$ et $log_{16} N$.

$log_2 N = 2log_4 N = 4log_{16} N$

   d. Comparer les complexités des 2-`wayQuickSort`, 4-`wayQuickSort` et 16-`wayQuickSort` et dire lequel est meilleur.

$k = 1$	$k = 4$	$k = 16$
$T_2(N) = N.log_2 N - N + 1$	$T_4(N) = 3(\frac{N}{2}log_2 N - N + 1)$ $= \frac{3}{2}(N.log_2 N - 3N + 3)$	$T_{16}(N) = 15(\frac{N}{4}log_2 N - N + 1)$ $= \frac{15}{4}(N.log_2 N - 15N + 15)$

Pour N assez grand $T_2(N) < T_4(N) < T_{16}(N)$

e. Généraliser le constat observé et déterminer la valeur optimale du paramètre k.

Plus k est grand, plus la complexité est grande.
Plus le nombre de pivots est réduit, mieux c'est.
==> le nombre minimal et optimal de pivots = 1.

Partie D. Amélioration du k-`wayQuickSort` (5 pts)

Une amélioration possible sur le k-`wayQuickSort` est de structurer les pivots en un arbre binaire de recherche équilibré. Ceci permet de réduire le nombre de comparaisons effectuées lors de la phase de partitionnement.

Pour partitionner un tableau en k sous-tableaux, au lieu de comparer les éléments avec tous les pivots, nous passons par un ABR équilibré. La question de partitionnement des éléments revient à chercher dans quel partition on va insérer chacun des éléments.

6. Considérons les variantes k-`wayQuickSort` telles que k est une puissance de 2. Les $k-1$ pivots utilisés tiennent dans un ABR complet et parfaitement équilibré.
   a. Déterminer le nombre de comparaisons nécessaires pour trouver dans quel sous-tableau on doit insérer un élément du tableau.

Le nombre comparaisons à faire est la hauteur de l'arbre des pivots = $log_2 k$.

   b. Réviser la complexité du k-`wayQuickSort` en tenant compte de cette amélioration.

$$
\begin{aligned}
&T(N) = kT(\tfrac{N}{k}) + (N+1-k)log_2 k \\
&T(N) = k\big(kT(\tfrac{N}{k^2}) + (\tfrac{N}{k}+1-k)log_2 k\big) + (N+1-k)log_2 k \\
&= k\Big(k\big(kT(\tfrac{N}{k^3}) + (\tfrac{N}{k^2}+1-k)log_2 k\big) + (\tfrac{N}{k}+1-k)log_2 k\Big) + (N+1-k)log_2 k\big) \\
&\vdots \\
&= \underbrace{k^p T(1)}_{=0} + \textstyle\sum_{i=0}^{p-1} k^i(\tfrac{N}{k^i} - k + 1)log_2 k, \text{ tel que } p = log_k N \text{ et } k^p = N \\
&= log_2 k\Big(pN - (k-1)\textstyle\sum_{i=0}^{p-1} k^i\Big) \\
&= log_2 k\Big(pN - (k-1)\tfrac{k^p-1}{k-1}\Big) \\
&= log_2 k(pN - N + 1) \\
&= log_2 k(N.log_k N - N + 1)
\end{aligned}
$$

   c. Comparer les complexités des 4-`wayQuickSort`, 8-`wayQuickSort` et 16-wayQuicksort et dire lequel réalise moins de comparaisons.

Pour comparer les 3 complexités, nous devons ramener les *log* dans la même base.
$log_2 x = 2log_4 x = 3log_8 x = 4log_{16} x$

4-`wayQuickSort` :
$T(N) = log_2 4(N.log_4 N - N + 1) = 2(\frac{N}{2}.log_2 N - N + 1) = N.log_2 N - \frac{N-1}{2}$
8-`wayQuickSort` :
$T(N) = log_2 8(N.log_8 N - N + 1) = 3(\frac{N}{3}.log_2 N - N + 1) = N.log_2 N - \frac{N-1}{3}$
16-`wayQuickSort` :
$T(N) = log_2 16(N.log_{16} N - N + 1) = 4(\frac{N}{4}.log_2 N - N + 1) = N.log_2 N - \frac{N-1}{4}$

Nous remarquons maintenant que 16-`wayQuickSort` est légèrement meilleur que 8-`wayQuickSort`, qui est encore légèrement meilleur 4-`wayQuickSort`.
Nous pouvons conclure alors que si nous structurons les pivots dans un ABR, nous pouvons augmenter le nombre de pivots sans affecter la complexité.
En effet, si nous réduisons le coût de la composante itérative, nous pouvons augmenter l'arité de la réccursivité, tout en réduisons (ou du monins conserver) la complexité global de l'algorithme.
Notons aussi, que dans notre étude nous n'avons pas tenu compte du coût de la construction des ABR des pivots (qui est en $\mathcal{O}(k.log_2 k)$), ce qui peut affecter la complexité.