2022-2023, Semestre d'automne
L3, Licence Sciences et Technologies
Université Lyon 1

LIFAPC: Algorithmique, Programmation et Complexité

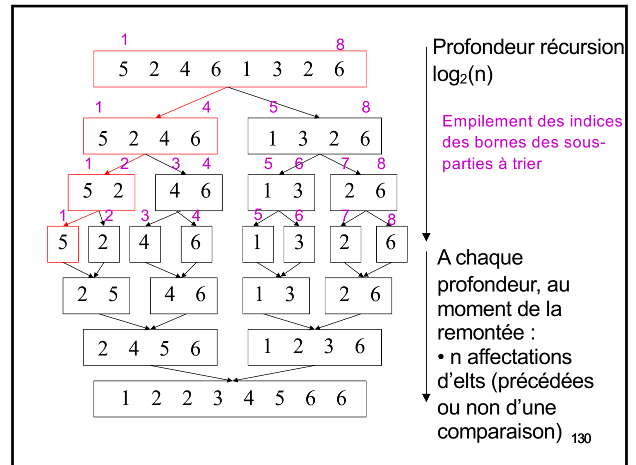
Chaine Raphaëlle (responsable semestre automne)
E-mail : raphaelle.chaine@liris.cnrs.fr
http://liris.cnrs.fr/membres?idn=rchaine

Spécificités des algorithmes itératifs et récursifs

- Calculs de complexité
- Complexité des algorithmes itératifs :
  - Utilisation des règles révisées dans les slides 103 et 104
- Complexité des algorithmes récursifs :
  - Solution d'une équation de récurrence

Complexité d'un algorithme récursif

- Le temps d'exécution d'un algorithme récursif est généralement solution d'une équation de récurrence
- Exemple : Analyse de la procédure TriFusionRec
  - Appelons $T_{TF}(n)$ le temps d'exécution de l'algorithme sur un tableau de taille n



A chaque profondeur, au moment de la remontée :
- n affectations d'elts (précédées ou non d'une comparaison)

Temps $T_{TF}(n)$ pour trier un tableau de taille n

- $T_{TF}(1)$=T(initialisation des paramètres formels) +T(comparaison d'indice)=C_1
- Pour n≥2
  $T_{TF}(n)$= T(initialisation des paramètres formels)
  + T(comparaison d'indice)
  + T(affectation d'indice)
  + 2*$T_{TF}(n/2)$
  + T(fusionner,n)
- Or T(fusionner,n)=θ(n)

$$T(n) = \begin{cases} C1 \text{ si } n=1 \\ 2T(n/2)+\theta(n) + C1 + C2 \text{ si } n>1 \end{cases}$$

$$T(n) = \begin{cases} C1 \text{ si } n=1 \\ 2T(n/2)+\theta(n) \text{ si } n>1 \end{cases}$$

Résolution

Méthode par développement itératif
Soit n>1 et soit $k=lg_2(n)$
(on supposera ici que n correspond exactement à 2^k)
T(n)=2T(n/2)+θ(n)
ie. T(n) = 2(2T(n/2²)+ θ(n/2)) + θ(n)
ie. $T(n) = 2^2T(n/2^2)+2\theta(n/2) + \theta(n)$
ie. $T(n) = 2^kT(n/2^k)+ 2^{k-1}\theta(n/2^{k-1}) + 2^{k-2}\theta(n/2^{k-2}) \ldots +\theta(n)$
ie. $T(n) = 2^kT(n/2^k)+ \theta(n) + \ldots +\theta(n)$
ie. $T(n) = n\ C1 + lg_2(n)\theta(n)$
ie. $T(n) = \theta(n\ lg_2(n))$

133

133

- Il s'agit du temps d'exécution d'un algorithme qui divise un problème de taille n en a≥1 sous-problèmes de taille n/b avec b>1 (stratégie **« diviser pour régner »**)

$$T(n) = aT(\frac{n}{b}) + f(n) \quad n>0$$

f(n) décrit le coût de la division du problème en a sous-problèmes et de la recomposition des résultats

134

134

Master Theorem

1. Si il existe $\varepsilon > 0$ t. que $f(n) = O(n^{\log_b(a)-\varepsilon})$ alors $T(n) = \Theta(n^{\log_b(a)})$
2. Si $f(n) = \Theta(n^{\log_b(a)})$ alors $T(n) = \Theta(n^{\log_b(a)} \log_b(n))$
3. Si il existe $\varepsilon > 0$ t.que $f(n) = \Omega(n^{\log_b(a)+\varepsilon})$ et il existe $c < 1$ t. que $af(\frac{n}{b}) \leq cf(n)$ pour n grand alors $T(n) = \Theta(f(n))$

Intuition de preuve au tableau

135

135

Master Theorem

1. Si il existe $\varepsilon > 0$ t. que $f(n) = O(n^{\log_b(a)-\varepsilon})$ alors $T(n) = \Theta(n^{\log_b(a)})$ Coût de la décomposition et recomposition négligeable devant le coût lié à la récursion
2. Si $f(n) = \Theta(n^{\log_b(a)})$ alors $T(n) = \Theta(n^{\log_b(a)} \log_b(n))$ Coût de la décomposition et recomposition similaire à celui lié à la récursion
3. Si il existe $\varepsilon > 0$ t.que $f(n) = \Omega(n^{\log_b(a)+\varepsilon})$ et il existe $c < 1$ t. que $af(\frac{n}{b}) \leq cf(n)$ pour n grand alors $T(n) = \Theta(f(n))$

Coût de la décomposition et recomposition non négligeable devant le coût lié à la récursion, … mais maîtrisé!

136

136

- Construisons ensemble une procédure récursive dont la complexité augmente avec la valeur d'un entier n passé en paramètre.
- On souhaite connaître son comportement asymptotique quand n augmente.
- Cette fonction contient des instructions d'affichage (« Coucou ») sur la sortie standard et répond à une stratégie récursive.
  - Cette fonction s'appelle elle-même 6 fois (a=6) pour une valeur de n divisée par b=3.
  - Les instructions d'affichage sont situées en dehors du cas d'arrêt

137

137

```
void proc(int n)
  {
       for(int i=0;i<6;i++)
         proc(n/3);
       for(int j=0;j<n;j++)
          affiche(« coucou »);
   }
```

- Application du Master Theorem
  - f(n)=n affichages de « Coucou » hors des appels récursifs emboîtés.
  - $\log_3(6)$ = 1.6309… (ie de la forme 1+epsilon)
  - Cas 1 du Master Theorem : $T(n)=\theta(n^{\log3(6)})$

138

138

Retour sur le tri fusion

- Remarque : La version récursive du tri fusion étudiée aujourd'hui diffère de celle que vous aviez découverte en LIFAP3.
- Il s'agissait d'une version où on ne coupait pas la séquence S des éléments à trier de la même manière!
  - La taille de la séquence n'était pas connue à priori
  - Version adaptée au tri des séquences rangées dans des fichiers ou des listes chaînées

139

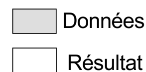
139

- Il s'agissait d'un algorithme itératif qui traitait la séquence initiale comme une séquence de monotonies de longueur 1
- A chaque passage dans la boucle
  - Répartition des monotonies dans 2 autres séquences S1 et S2, à raison d'1 monotonie sur 2 (éclatement)
  - Fusion de la kième monotonie de S1 avec la kième monotonie de S2 et réécriture dans S qui contient ainsi des monotonies de longueur double (sauf peut-être la dernière)

140

140

```
procédure TriFusionItératif(
    donnée-résultat S : séquence)
variables S1,S2 : séquence
début
    lgmono←1
    répéter
        Éclatement( S , lgmono , S1 , S2 )
        Fusion( S1 , S2 , lgmono , S , nbmono )
        lgmono=lgmono*2
    tantque nbmono≠1
fin
```



Données

Résultat

141

141

- Eléments supplémentaires de comparaison
  - TriFusionItératif n'est pas un tri sur place (besoin de séquences supplémentaires, le tri ne se fait pas sur place)
  - A votre avis TriFusionRec est-il un tri sur place?

142

142

- De même que TriFusionItératif, TriFusionRec n'est pas un tri sur place :
  - Fusionner(tab,p,q,r) nécessite un espace supplémentaire proportionnel à la taille du sous-tableau à réorganiser
  - il ne faut pas oublier l'espace requis pour gérer la récursion!
- En revanche sur une liste chainée il est possible d'utiliser le triFusion « sur place », sans nécessiter d'espace supplémentaire;

143

143