Université Clermont Auvergne
IUT 3A
Qualité algorithmique

Florent Foucaud, Pascal Lafourcade et Malika More
2024-2025

TD3 - Diviser pour régner - Complexité

Exercice 1 (Recherche dichotomique récursive)**.**

L'objectif est de calculer la complexité en temps $T(N)$ de l'algorithme de recherche dichotomique ci-dessous, appelé avec $m = 1$ et $n = N$. Pour simplifier, on supposera que $N = 2^k$ est une puissance de deux.

1. Dessiner l'arbre des appels récursifs de `rechercheDicho` pour un tableau de taille $N = 2^4$. Écrire la valeur de $n - m + 1$ (taille du sous-tableau courant) sur chaque sommet.
2. Calculer le nombre d'opérations à chaque appel récursif.
3. Dans le cas général, quelle est la hauteur de l'arbre ?
4. En déduire la complexité en temps $T(N)$ de l'algorithme de recherche dichotomique.

`rechercheDicho(`T, m, n, x`): calcul d'une position de` x `dans` T

`Entrée : Un tableau d'entiers` $T[1...N]$ `triés par ordre croissant, un entier` x`,`
`deux positions` m `et` n

`Sortie : 0 si` $x \notin T$`, et une position de` x `dans` T `(entre` m `et` n`) sinon`

- `Si` $m == n$ `:`
  - ⋆ `Si` $T[m] == x$ `:`
    `Retourner` m
  - ⋆ `Sinon :`
    `Retourner 0`
- `Sinon :`
  - ⋆ $k = \lfloor \frac{m+n}{2} \rfloor$
  - ⋆ `Si` $T[k] < x$ `:`
    `Retourner rechercheDicho(`$T, k+1, n, x$`)`
  - ⋆ `Sinon :`
    `Retourner rechercheDicho(`T, m, k, x`)`

Exercice 2 (Diviser pour régner : le tri par fusion)**.**

Pour simplifier, on supposera que $N = 2^k$ est une puissance de deux.

1. Estimer la complexité de la sous-fonction `interClassement(`TAB_1, TAB_2`)` en fonction de N_1 et N_2.
2. Dessiner l'arbre des appels récursifs de `triFusion` pour un tableau de taille $N = 2^4$. Quelle est la complexité approximative de l'algorithme sur *chaque niveau* de l'arbre ?
3. En déduire une estimation informelle de la complexité globale de l'algorithme.
4. On va maintenant calculer plus précisément cette complexité. Exprimer la complexité $T(N)$ de `triFusion` en fonction de $T(N/2)$.
5. Appliquer la formule précédente à $T(N/2)$ pour substituer $T(N/2)$ dans la formule. Ainsi on exprime $T(N)$ en fonction de $T(N/4)$.
6. Faire encore une étape en exprimant $T(N)$ en fonction de $T(N/8)$.
7. En déduire $T(N)$ en fonction de $T(N/2^i)$ ($i$ème étape récursive).
8. On rappelle que $N = 2^k$, donc si $i = k$ on a $T(N/2^i) = T(1)$. En déduire l'expression de $T(N)$ uniquement en fonction de N.

`triFusion(`TAB, m, n`): tri du sous-tableau` $TAB[m...n]$ `d'entiers par ordre croissant`

`Entrée : Un tableau d'entiers` $TAB[1...N]$ `et deux positions` m, n `avec` $m \leq n$

`Sortie : Le sous-tableau` $TAB[m...n]$ `trié`

- `Si` $m < n$:
  - $k = \lfloor \frac{m+n}{2} \rfloor$
  - TAB_1 = `triFusion(`TAB, m, k`)`
  - TAB_2 = `triFusion(`$TAB, k+1, n$`)`
  - $TAB[m...n]$ = `interClassement(`TAB_1, TAB_2`)`
- `Retourner` $TAB[m...n]$

interClassement(TAB_1, TAB_2):

Entrée : Deux tableaux d'entiers $TAB_1[1...N_1]$, $TAB_2[1...N_2]$ triés par ordre croissant

Sortie : L'union TAB de TAB_1 et TAB_2 triée par ordre croissant

- $i = 1$, $j = 1$, $k = 1$
- TAB est un tableau vide à $N_1 + N_2$ éléments
- TABant que $i \leq N_1$ et $j \leq N_2$:
  - Si $TAB_1[i] < TAB_2[j]$:
    $TAB[k] = TAB_1[i]$
    $i = i + 1$
  - Sinon :
    $TAB[k] = TAB_2[j]$
    $j = j + 1$
  - $k = k + 1$
- Tant que $i \leq N_1$:
  $TAB[k] = TAB_1[i]$
  $i = i + 1$
  $k = k + 1$
- Tant que $j \leq N_2$:
  $TAB[k] = TAB_2[j]$
  $j = j + 1$
  $k = k + 1$
- Retourner TAB

Exercice 3 (Calcul)**.**
Soit $T(n) = 3T(n/2) + n$ et $T(1) = 3$, calculer $T(8)$ et $T(n)$.

Sachant que

$$\sum_{i=0}^{i=n-1} (a)^i = \frac{a^n - 1}{a - 1}$$

Exercice 4 (Calcul)**.**
Soient 3 entiers a, b et c, $T(n) = aT(n-1) + c(b^n)$, calculer la valeur exacte de $T(n)$ en fonction de $T(0)$, a, b et c, en effectuant des substitutions successives.

Exercice 5 (Calcul)**.**
Soit $T(n) = T(n/3) + T(n/4) + 5n$, en effectuant des substitutions successives deviner la complexité de $T(n)$. Puis vérifier que cela est correct.

Exercice 6 (Master Theorem)**.**
RAPPEL

$$T(n) = a\ T\left(\frac{n}{b}\right) + f(n), \quad a \geq 1, b > 1$$

1. Si $f(n) = O(n^c)$ avec $c < \log_b a$, alors $T(n) = \Theta\left(n^{\log_b a}\right)$.
2. Si $f(n) = \Theta\left(n^c \log^k n\right)$ avec $c = \log_b a$ et une constante $k \geq 0$, alors $T(n) = \Theta\left(n^c \log^{k+1} n\right)$
3. Si $f(n) = \Omega(n^c)$ avec $c > \log_b a$ et s'il existe une constante $k < 1$ telle que, pour n assez grand, on a : $af\left(\frac{n}{b}\right) \leq kf(n)$ alors on a : $T(n) = \Theta(f(n))$

$$T(n) = a\ T\left(\frac{n}{b}\right) + O(n^d)$$

1. Si $d < \log_b a$ alors $T(n) = O(n^{\log_b a})$
2. Si $d = \log_b a$ alors $T(n) = O(n^d \log_b n)$
3. Si $d > \log_b a$ alors $T(n) = O(n^d)$

Appliquer si possible le Master Theorem pour ces formules :

1. $T(n) = 3T(n/2) + n^2$
2. $T(n) = T(n/2) + 2^n$
3. $T(n) = 16T(n/4) + n$
4. $T(n) = 2T(n/2) + nlogn$
5. $T(n) = 2T(n/2) + n/(logn)$
6. $T(n) = 0.5T(n/2) + 1/n$
7. $T(n) = \sqrt{2}T(n/2) + logn$
8. $T(n) = 3T(n/3) + n/2$
9. $T(n) = 4T(n/2) + logn$