Université Clermont Auvergne
ISIMA FISA 1
Algorithmique

Florent Foucaud
2024-2025

TD2 - Diviser pour régner

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 $N = 2^k$.

1. Dessiner l'arbre des appels récursifs de `rechercheDicho` pour *une exécution* de l'algorithme, avec 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 (approximative) en temps $T(N)$ de l'algorithme de recherche dichotomique.
5. On va être maintenant plus précis. Exprimer tout d'abord $T(N)$ en fonction de $T(N/2)$.
6. Remplacer N par $N/2$ dans la formule obtenue pour exprimer $T(N/2)$ en fonction de $T(N/4)$. Substituer la valeur obtenue pour $T(N/2)$ dans la formule précédente, pour obtenir $T(N)$ seulement en fonction de $T(N/4)$. Simplifier.
7. Généraliser pour exprimer $T(N)$ en fonction de $T(N/2^i)$ (ième 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.

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

```
interClassement(TAB1, TAB2):

Entrée : Deux tableaux d'entiers TAB1[1...N1], TAB2[1...N2] triés par ordre
     croissant

Sortie : L'union TAB de TAB1 et TAB2 triée par ordre croissant
```

- $i = 1$, $j = 1$, $k = 1$
- TAB est un tableau vide à $N_1 + N_2$ éléments
- Tant 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