Université Clermont Auvergne
ISIMA FISA 1
Algorithmique

Florent Foucaud
2025-2026

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 en fonction de N ?
4. En déduire la complexité (approximative) en temps $T(N)$ de l'algorithme de recherche dichotomique.
5. On va maintenant faire une autre analyse en passant par des équations. 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 (sans preuve) $T(N)$ 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`)`

Solution.

- C'est une "ligne", de $N = 16$, $\frac{N}{2} = 8$, $\frac{N}{4} = 4$, $\frac{N}{8} = 2$, $\frac{N}{16} = 1$.
- 9 opérations dans le pire des cas

- Si $m == n$: // Cout $= 1$ (fait dans tous les cas)
  - ⋆ Si $T[m] == x$: // Cout : 2
    Retourner m // Cout : 1
  - ⋆ Sinon :
    Retourner 0 // Cout : 1
- Sinon :
  - ⋆ $k = \lfloor \frac{m+n}{2} \rfloor$ // Cout : 4
  - ⋆ Si $T[k] < x$: // Cout : 2
    Retourner rechercheDicho(T,$k+1$,n,x) // Cout : 2
  - ⋆ Sinon :
    Retourner rechercheDicho(T,m,k,x) // Cout : 1

- la hauteur de l'arbre est $log_2(N) + 1$
- On a $log_2(N) + 1$ appels exécutions de l'algorithme (hauteur de l'arbre). A chaque appel on fait au plus 9 opérations cela donne donc $9(log_2(N) + 1) \in O(\log(N))$ opérations au total. *(En pinaillant, le dernier appel ne donne que 4 opérations donc on obtient* $9log_2(N) + 4$*.)*

  Analyse par calcul et substitution :

  Plus formellement, on peut faire l'analyse suivante.

  Soit $N' = m - n + 1$ (la taille du sous-tableau $T[m...n]$). Au premier appel, on a $N' = N$. À chaque appel récursif, au plus 9 opérations de base sont réalisées. À chaque étape, k divise le tableau exactement en deux parties égales et on fait un appel récursif sur un tableau deux fois moins grand, sauf si $N' = 1$.

  Dans ce (pire des) cas, on peut done écrire que $T(N') \leq 9 + T(N'/2)$. Au deuxième appel récursif, on a $T(N'/2) \leq 9 + T(N'/4)$, ce qui donne $T(N') \leq 9 + 9 + T(N'/4) \leq 18 + T(N'/4)$. Avec la même logique, au bout de i appels récursifs, on a $T(N') \leq 9 \times i + T\left(\frac{N'}{2^i}\right)$.

  Combien y a-t-il d'appels récursifs ? On s'arrête quand $m = n$, soit $N' = 1$ (cas de base de l'algorithme). On veut donc résoudre l'équation $\frac{N'}{2^i} = 1$, cela donne $N' = 2^i$ soit $i = \log_2(N')$. Pour nous, $N = N' = 2^k$, on obtient donc $i = \log_2(N) = k$.

  On a donc $T(N) \leq 9 \times \log_2(N) + T(1)$, et $T(1) = 4$, donc $T(N) \leq 9 \times \log_2(N) + 4$ ce qui appartient à la classe de complexité $\Theta(\log(N))$.

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

Pour simplifier, on supposera que $N = 2^k$ est une puissance de deux. La complexité de la sous-fonction `interClassement(`TAB_1, TAB_2`)` est $13(N_1 + N_2) + 6$ dans le pire des cas.

1. 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 ?
2. En déduire une estimation informelle de la complexité globale de l'algorithme.
3. On va maintenant calculer plus précisément cette complexité. Exprimer la complexité $T(N)$ de `triFusion` en fonction de $T(N/2)$.
4. Appliquer la formule précédente à $T(N/2)$ pour substituer $T(N/2)$ dans la formule initiale. Ainsi on exprime $T(N)$ en fonction de $T(N/4)$.
5. Faire encore une étape en exprimant $T(N)$ en fonction de $T(N/8)$.
6. En déduire (sans preuve) $T(N)$ en fonction de $T(N/2^i)$ ($i$ème étape récursive).
7. 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(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
- 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

Solution.

1. Estimons la complexité de interClassement(TAB_1, TAB_2).

   Noter que seule la première et l'une des deux dernières boucles seront parcourues.

   Tout d'abord, il y a 3 affectations de i, j, k et l'initialisation du tableau (2 instructions pour l'initialisation et le calcul de $N_1 + N_2$).

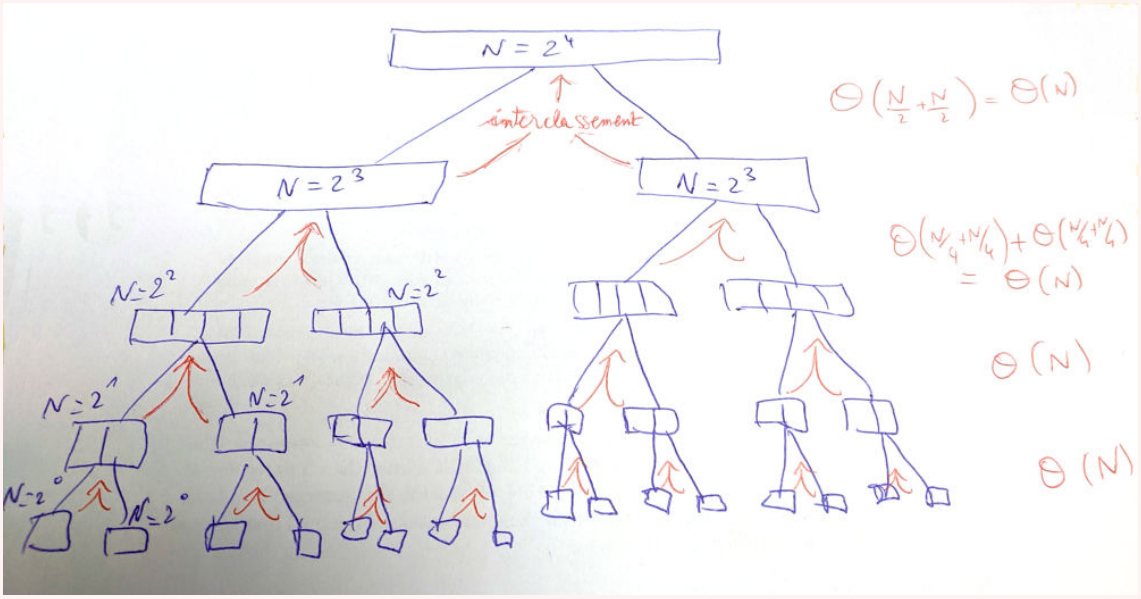
   Ensuite, il y a au plus 13 opérations de base par tour de boucle sur la première boucle (3+3+3+2+2), et 8 pour les autres boucles ($1 + 3 + 2 + 2$). Donc, dans le pire des cas, on épuise au maximum les deux tableaux, et on fait $N_1 + N_2 - 1$ tours de la première boucle, et 1 tour de l'une des deux suivantes.

   Par ailleurs il y aura pour chacune des 3 boucles, un test de fait dont le résultat est négatif, soit 5 opérations au total (3 pour la 1e boucle, 1 pour chacune des deux autres).

   Enfin, le retour de TAB (1 instruction).

   On a donc (dans le pire de cas) $3 + 2 + 13(N_1 + N_2 - 1) + 8 + 5 + 1 = 13(N_1 + N_2) + 6$ opérations.

2. À chaque niveau, le total est une somme d'interclassements, au total c'est linéaire.



3. Pour l'algorithme principal, de nouveau, de façon informelle, on note qu'on appelle récursivement l'algorithme sur deux fois la moitié du tableau, et pour chacun de ces deux appels, on fait appel à `interClassement(`TAB_1, TAB_2`)`. A chaque niveau, on fait un traitement linéaire puisqu'on fait des interclassements sur des sous-tableaux dont la somme des tailles fait N. On peut représenter la structure de ces appels par un arbre binaire dont les feuilles sont les sous-tableaux de taille 1. Cet arbre binaire a une hauteur de $\log_2(N)$, et pour chaque niveau de l'arbre, on a une complexité de $\Theta(N)$ (la somme des appels à `interClassement(`TAB_1, TAB_2`)` pour ce niveau).

   Au total, on a donc une complexité de $\Theta(N \log(N))$.

4. On a $N = m - n$. On fait une comparaison, 4 opérations mathématiques, 2 appels récursifs et 2 affectations du résultat, un appel à `interClassement(`TAB_1, TAB_2`)` avec une affectation, et un retour. Cela donne donc

$$\begin{aligned} T(N) &\leq 1+4+2+2T(N/2)+(13N+6)+N+1 \\ &= 2T(N/2)+14N+14 \end{aligned}$$

5. $T(N/2) \leq 2T(N/4)+14N/2+14$. On substitue dans la formule précédente :

$$\begin{aligned} T(N) &\leq 2T(N/2)+14N+14 \\ &= 2(2T(N/4)+14N/2+14)+14N+14 \\ &= 4T(N/4)+2\times 14N+3\times 14 \end{aligned}$$

6. $T(N/4) \leq 2T(N/8)+14N/4+14$. On substitue dans la formule précédente :

$$\begin{aligned} T(N) &\leq 4T(N/4) + 2 \times 14N + 3 \times 14 \\ &= 4(2T(N/8) + 14N/4 + 14) + 2 \times 14N + 3 \times 14 \\ &= 8T(N/8) + 3 \times 14N + 7 \times 14 \end{aligned}$$

7. Avec la même logique, au bout de i appels récursifs, on a $T(N) \leq (14N+14) \times (2^i - 1) + 2^i T\left(\frac{N}{2^i}\right)$.

$$\begin{aligned} T(N) &\leq 2^i T\left(\frac{N}{2^i}\right) + i \times 14N + \sum_{j=0}^{i-1} 2^j \times 14 \\ &= 2^i T\left(\frac{N}{2^i}\right) + i \times 14N + (2^i - 1) \times 14 \end{aligned}$$

8. On s'arrête quand $\frac{N}{2^i} = 1$ (cas de base). On veut donc résoudre l'équation $\frac{N}{2^i} = 1$, cela donne $N = 2^i$ soit $i = \log_2(N)$. Pour nous, $N = 2^k$, on obtient donc $i = \log_2(N) = k$.
   On a donc (avec $T(1) = 2$ pour le test et le return) :

$$\begin{aligned} T(N) &\leq 2^k T\left(\frac{N}{2^k}\right) + k \times 14N + (2^k - 1) \times 14 \\ &= N \times T(1) + 14N \log_2(N) + 14(N-1) \\ &= 14N \log_2(N) + 16N - 14 \end{aligned}$$

   ce qui appartient à $O(N \log(N))$.