Révision pour l'examen 1

Claude-Guy Quimper

Chapitre 1

- Qu'est-ce qu'un algorithme?
- Les grandes étapes pour passer du problème à l'algorithme:
  - Définir précisément le problème
  - Identifier l'entité qui exécutera
  - Choisir les structures de données
  - Concevoir l'algorithme
  - Décrire l'algorithme
  - Prouver que l'algorithme est correct
  - Analyse l'algorithme
  - Coder l'algorithme

Chapitre 1

- Plus grand commun diviseur
  - Algorithme de force brut
  - Algorithme d'Euclide (diminuer pour régner)

Chapitre 2

- Définition d'une opération élémentaire
- Définition d'une opération baromètre.
- Définition de la taille d'une instance
- Temps d'exécution

$$C_{worst}(n) = \max_{x:|x|=n} C(x)$$

$$C_{best}(n) = \min_{x:|x|=n} C(x)$$

$$C_{avg}(n) = \sum_{x:|x|=n} P(x)C(x)$$

Chapitre 2

- Notation asymptotique

$$O(g(n)) = \{t(n) : \exists c, n_0 : t(n) \leq cg(n) \quad \forall n \geq n_0\}$$

$$\Omega(g(n)) = \{t(n) : \exists c, n_0 : t(n) \geq cg(n) \quad \forall n \geq n_0\}$$

$$\Theta(g(n)) = \{t(n) : \exists c_1, c_2, n_0 : c_2 g(n) \leq t(n) \leq c_1 g(n) \quad \forall n \geq n_0\}$$

- Deux façons de prouver si f(n) appartient à O(g(n)).
  - Trouver les constantes c et n_0.
  - Utiliser les limites

$$\text{si } \lim_{n\to\infty} \frac{f(n)}{g(n)} \begin{cases} = & c \text{ (fini)} > 0 & \text{alors} & f(n) \in \Theta(g(n)) \\ = & 0 & \text{alors} & f(n) \in O(g(n)) \text{ et } f(n) \notin \Theta(g(n)) \\ = & +\infty & \text{alors} & f(n) \in \Omega(g(n)) \text{ et } f(n) \notin \Theta(g(n)) \end{cases}$$

Analyse d'algorithmes non récursifs

ALGORITHME ElemDiff(A[0..n-1])

//Entrée: tableau A de n éléments
//Sortie: **true** si tous les éléments
//sont distincts et **false** autrement

for i ⟵ 0 **to** n - 2 **do**

    for j ⟵ i+1 **to** n - 1 **do**

      if **A[i]=A[j]** **return false**

return true

$$C_{BEST}(n) = 1$$

$$C_{WORST}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \ldots \in \Theta(n^2)$$

Résoudre des sommations

- Utilisez l'aide-mémoire
- Utilisez les intégrales

$$\int_{l-1}^{u} f(x)dx \;\leq\; \sum_{i=l}^{u} f(i) \;\leq\; \int_{l}^{u+1} f(x)dx \text{ pour } f(x) \text{ non décroissant}$$

$$\int_{l}^{u+1} f(x)dx \;\leq\; \sum_{i=l}^{u} f(i) \;\leq\; \int_{l-1}^{u} f(x)dx \text{ pour } f(x) \text{ non croissant}$$

Analyse des algorithmes non récursifs

ALGORITHME Binary(n)

//Entrée: entier positif de valeur n
//Sortie: le nombre de bits utilisés pour
//sa représentation binaire

count ⟵ 1

while n > 1 **do**

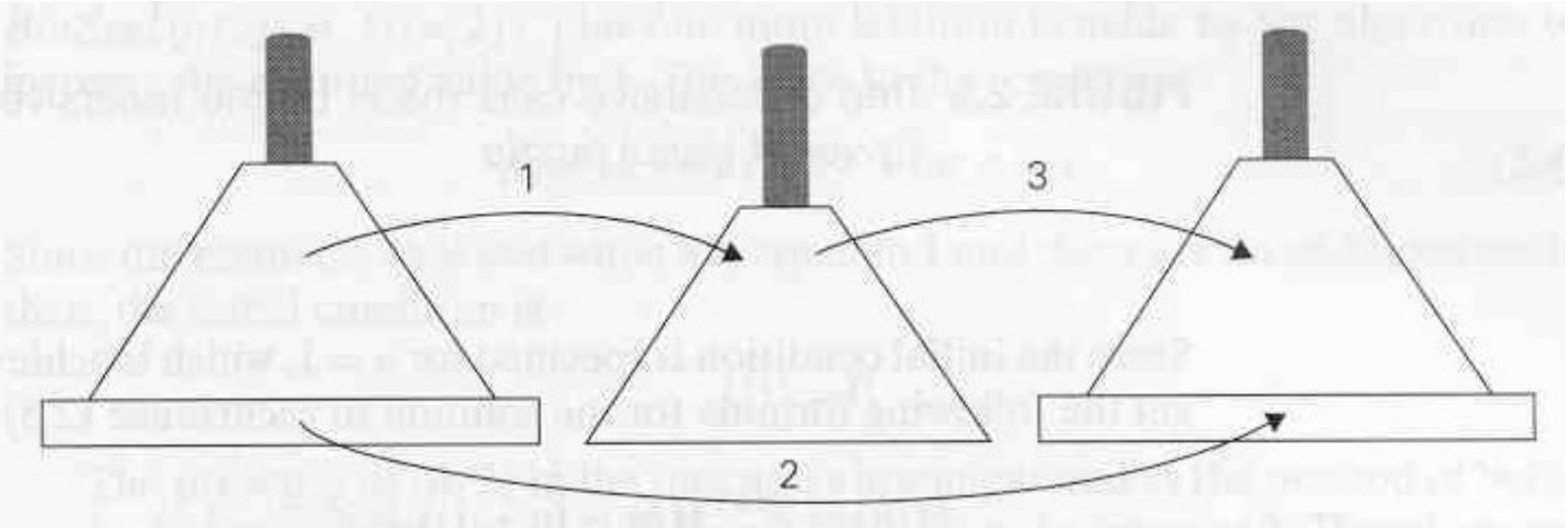
count ⟵ count + 1

n ⟵ $\lceil n/2 \rceil$

return count

- On peut définir l'efficacité de l'algorithme à l'aide d'une fonction récursive.
- $C(n) = C(\lceil n/2 \rceil) + 1$

Analyse des algorithmes récursifs



- C(n) = C(n-1) + 1 + C(n-1) = 2C(n-1) + 1 ∀ n > 1 **(la récurrence)**
- Avec: C(1) = 1 **(condition initiale)**

Analyse des algorithmes récursifs

- Si l'efficacité de votre algorithme est donnée par une récurrence de la forme

$$T(n) = rT(\frac{n}{b}) + f(n)$$

- et que l'on vous demande de donner l'efficacité asymptotique de l'algorithme, utilisez le théorème général.

$$T(n) \in \begin{cases} \Theta(n^d) & \text{si } r < b^d \\ \Theta(n^d \log n) & \text{si } r = b^d \\ \Theta(n^{\log_b r}) & \text{si } r > b^d \end{cases}$$

- Si vous ne pouvez pas utiliser le théorème général, alors vous devez résoudre la récurrence avec la substitution à rebours.

Fonctions harmonieuses

- Une fonction non négative f(n) est **éventuellement non décroissante** si et seulement s'il existe n_0 tel que f(n) est non décroissante dans $[n_0,1)$
- Une fonction éventuellement non décroissante est **harmonieuse** si et seulement si $f(2n)\ 2\ \Theta(f(n))$

- **Théorème (règle de l'harmonie):** Si C(n) est éventuellement non décroissant, si f(n) est harmonieuse et si $C(n) \in \Theta(f(n))$ pour $n = b^k$ avec $k \in N$ alors $C(n) \in \Theta(f(n))\ \forall\ n \in N$

Chapitre 3

- Algorithmes de force brute
  - pgcdFB
  - Tri à bulles

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1 = \sum_{i=0}^{n-2} (n-1-i)$$

$$= \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \in \Theta(n^2)$$

$$\text{car } \lim_{n\to\infty} \frac{n(n-1)}{2n^2} = \lim_{n\to\infty} \left(\frac{1}{2} - \frac{1}{2n}\right) = \frac{1}{2}$$

Chapitre 4: Diviser pour régner

ALGORITHME DiviserPourRégner(x)
  if |x| < seuil **return** *adhoc*(x)
  diviser x en plus petites instances x_1, x_2, …, x_r
  for i ⟵ 1 **to** r **do** y_i ⟵ DiviserPourRégner(x_i)
  recombiner les y_i pour obtenir la solution y de x
  return y

- Respectez ce gabarit. Le solutionnaire le respecte 99% du temps. C'est votre meilleur indice pour résoudre le problème.

Tri fusion

ALGORITHME MergeSort(A[0..n-1])
//Entrée: le tableau A
//Sortie: le tableau A trié
if n > 1
    copy A[0.. $\lceil n/2 \rceil$ - 1] **to** B[0.. $\lceil n/2 \rceil$ - 1]
    copy A[$\lceil n/2 \rceil$.. n - 1] **to** C[0.. $\lceil n/2 \rceil$ - 1]
    MergeSort(B[0.. $\lceil n/2 \rceil$ - 1])
    MergeSort(C[0.. $\lceil n/2 \rceil$ - 1])
    Merge(B, C, A)
    (**Delete** B; **Delete** C)

- Comprenez-le au point de pouvoir le recoder, mais n'apprenez pas ce pseudo-code par cœur.
- Remarquez comment les tableaux sont divisés lorsque n est pair et lorsque n est impair.

$$C(n) \in \Theta(n \log n)$$

Tri rapide

ALGORITHME QuickSort(A[l..r])
//Entrée: le sous tableau A[l..r] de A[0..n-1]
//Sortie: le sous tableau A[l..r] trié
if l < r **then**
    s ⟵ Partition(A[l..r])
    QuickSort(A[l..s-1])
    QuickSort(A[s+1..r])

$$C_{BEST}(n) \in \Theta(n \log n)$$
$$C_{avg}(n) \in \Theta(n \log n)$$
$$C_{WORST}(n) \in \Theta(n^2)$$

Strassen

$$\begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix} = \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

$$\begin{aligned} m_1 &= (a_{00} + a_{11}) \times (b_{00} + b_{11}) \\ m_2 &= (a_{10} + a_{11}) \times b_{00} \\ m_3 &= a_{00} \times (b_{01} - b_{11}) \\ m_4 &= a_{11} \times (b_{10} - b_{00}) \\ m_5 &= (a_{00} + a_{01}) \times b_{11} \\ m_6 &= (a_{10} - a_{00}) \times (b_{00} + b_{01}) \\ m_7 &= (a_{01} - a_{11}) \times (b_{10} + b_{11}) \end{aligned}$$

- Comprenez-le au point de pouvoir le recoder, mais n'apprenez pas ce pseudo-code par cœur.

Recherche binaire

ALGORITHM BinarySearch(A[0..n-1], K)
  l ← 0; r ← n-1
  while l ≤ r **do**
    m ← ⌊(l + r) / 2⌋
    if K = A[m] **then return** m
    else if K < A[m] **then** r ← m – 1
    else l ← m + 1
  return -1

- Comprenez-le au point de pouvoir le recoder, mais n'apprenez pas ce pseudo-code par cœur.
- Remarquez comment le tableau est divisé lorsque n est pair et lorsque n est impair.
- La difficulté de cet algorithme, c'est d'éviter une boucle infinie causée par une mauvaise division du tableau.

$$C_{BEST}(n) \in \Theta(1)$$
$$C_{WORST}(n) \in \Theta(\log n)$$

Diviser pour régner

- Trouver le min et le max d'un tableau (Série 4)
- Chess-Boxing (Série 4)
- Le problème des boulons (Série 4)

Chapitre 5: Diminuer pour régner

- Problème d'exponentiation
  - Trois façons différentes de calculer a^n.

Chapitre 5: Diminuer pour régner

- Algorithme d'Euclide
- Binary / BinRec (Chapitre 2)
- Trouver la fausse pièce de monnaie (Série 5)
- Le problème du colis (Série 5)

Tri par insertion

```
ALGORITHM InsertionSort(A[0..n-1])
    for i ← 1 to n-1 do
        v ← A[i]    // élément à insérer
        j ← i – 1   // positions possibles d'insertion
        while j ≥ 0 and A[j] > v do
            A[j+1] ← A[j]
            j ← j – 1
        A[j+1] ← v   // insertion
```

- Comprenez-le au point de pouvoir le recoder, mais n'apprenez pas ce pseudo-code par cœur.
- Révisez l'analyse de l'algorithme.

$$C_{BEST}(n) \in \Theta(n)$$

$$C_{avg}(n) \in \Theta(n^2)$$

$$C_{WORST}(n) \in \Theta(n^2)$$

Le problème de sélection

ALGORITHME SelectionRec(A[l..r], k)
//Entrée: un sous tableau A[l..r] et un entier k : $1 \leq k \leq r - l + 1$
//Sortie: le kième plus petit élément dans A[l..r]
s ⟵ Partition(A[l..r])
if s-l+1 = k **return** A[s]
if s-l+1 > k **return** SelectionRec(A[l..s-1], k)
if s-l+1 < k **return** SelectionRec(A[s+1..r], k-s+l-1)

- Comprenez-le au point de pouvoir le recoder, mais n'apprenez pas ce pseudo-code par cœur.

Algorithmes probabilistes

$$C_{best}(n) = \min_{x:|x|=n} C(x)$$

$$C_{worst}(n) = \max_{x:|x|=n} C(x)$$

$$C_{avg}(n) = \mathbb{E}_{x:|x|=n} C(x) = \sum_{x:|x|=n} P_n(x) C(x)$$

Pas à l'examen intra.

$$\mathcal{C}_{best}(n) = \min_{x:|x|=n} \mathbb{E}_\rho C(x,\rho) = \min_{x:|x|=n} \sum_\rho p(\rho) C(x,\rho)$$

$$\mathcal{C}_{worst}(n) = \max_{x:|x|=n} \mathbb{E}_\rho C(x,\rho) = \max_{x:|x|=n} \sum_\rho p(\rho) C(x,\rho)$$

$$\mathcal{C}_{avg}(n) = \mathbb{E}_{x:|x|=n} \mathbb{E}_\rho C(x,\rho) = \sum_{x:|x|=n} P_n(x) \sum_\rho p(\rho) C(x,\rho)$$

- Algorithme de partition randomisé
  - Sélection randomisé: $C(n) \in \Theta(n)$
  - Tri rapide randomisé: $C(n) \in \Theta(n \log n)$

Chapitre 6: Transformer pour régner

- Pré triage
  - Trouver si deux éléments d'un tableau sont distincts.
  - Rechercher plusieurs éléments dans un tableau
- Monceaux
  - Propriétés d'un monceau: hauteur, nombre de feuilles, nombre de parents.

$$h = \lfloor \log_2(n) \rfloor$$
$$f = \left\lceil \frac{n}{2} \right\rceil$$
$$p = \left\lfloor \frac{n}{2} \right\rfloor$$

HeapBottomUp (heapify)

Procédure `HeapBottomUp` ($H[1..n]$)

// Construit un monceau à partir des éléments du tableau
// Input : Un tableau $H[1..n]$ d'éléments pouvant être comparés
// Output : Un monceau $H[1..n]$
pour $i = \lfloor n/2 \rfloor$ *en descendant jusqu'à 1* **faire**
  `Tamiser`($H[1..n]$, i)

Procédure `Tamiser` ($H[1..n]$, i)

// Tamise l'élément $H[i]$ dans le monceau jusqu'à ce qu'il soit plus grand
// que ses deux enfants ou qu'il devienne un enfant.
// Input : Un tableau $H[1..n]$ d'éléments pouvant être comparés et
// un index $1 \leq i \leq n$
// Output : Le tableau modifié
$k \leftarrow i$
$v \leftarrow H[k]$
monceau $\leftarrow$ faux
tant que $\neg$monceau $\wedge\ 2 \times k \leq n$ **faire**
  $j \leftarrow 2 \times k$
  // S'il y a deux enfants
  si $j < n$ **alors**
    si $H[j] < H[j+1]$ **alors** $j \leftarrow j+1$
  si $v \geq H[j]$ **alors**
    monceau $\leftarrow$ vrai
  sinon
    $H[k] \leftarrow H[j]$
    $k \leftarrow j$
$H[k] \leftarrow v$

Dans tous les cas

$$C(n) \in \Theta(n)$$

Autres algorithmes

- Réviser l'algorithme qui retire le plus grand élément d'un monceau.
- Réviser l'algorithme qui ajoute un élément au monceau.
- Réviser le tri par monceau (heapsort)

$$C(n) \in \Theta(n \log n)$$