Algorithmique 1
Introduction

Hajer Akid & Sandrine Julia

Semestre 4

Organisation

Thèmes

- Diviser pour régner – *Divide and Conquer*
- Algorithme glouton – *Greedy algorithm*
- Programmation dynamique – *Dynamic programming*

Objectifs

- Savoir analyser l'efficacité d'un algorithme
- Savoir développer des algorithmes efficaces
- Comprendre la notion de complexité d'un problème

Organisation

Modalités de connaissances

- Partiel
- Contrôle terminal

Trouver les informations

- Sur Moodle
- Sur le web

Remerciements

- à Emmanuel Kounalis
- à Marie Pelleau

Références

- T. Cormen, C. Leiserson, R. Rivest, **Introduction à l'algorithmique**
- S. Dasgupta, C. H. Papadimitriou, U. V. Vazirani, **Algorithms**
- T. Roughgarden, **Algorithms Illuminated**
- C. Berge, **Graphes et hypergraphes**
- M. Gondran et M. Minoux, **Graphes et Algorithmes**

Autres livres selon votre goût : ne pas hésiter à en consulter plusieurs

Les algorithmes dans les medias

Algorithmes

- Les programmes deviennent gros
- On ne peut plus programmer et penser n'importe comment

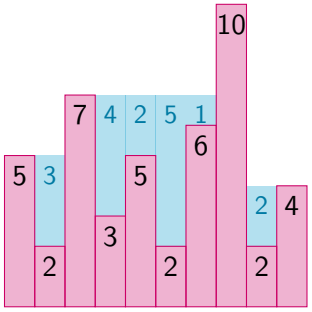
"For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing."

— Francis Sullivan

Pourquoi étudier les algorithmes ?

- Anciennes racines, nouveaux problèmes
- Leur impact est vaste et de grande envergure
- Pour devenir un programmeur compétent
- Résoudre des problèmes qui ne pourraient pas être résolus autrement
- Pour la stimulation intellectuelle

Question d'un entretien d'embauche chez Google



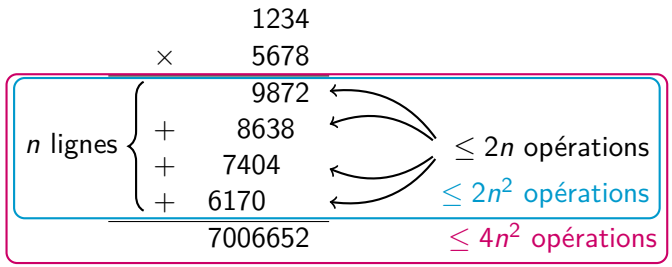
Soit un histogramme avec n barres sur lequel on verse un volume d'eau infini

- Donner un algorithme pour calculer le volume d'eau résiduel (17)
- Donner un algorithme pour calculer le volume d'eau résiduel **en temps linéaire**

Multiplication d'entiers

Soient x et y deux entiers non négatifs de n chiffres, donner un algorithme pour calculer $x \cdot y$

Comme à l'école



Quelle est la complexité de cet algorithme ?

Multiplication d'entiers

Soient x et y deux entiers non négatifs de n chiffres, donner un algorithme pour calculer $x \cdot y$

Comme les égyptiens/russe/éthiopiens

~~1234~~	~~5678~~
617	11356
~~308~~	~~22712~~
~~154~~	~~45424~~
77	90848
~~38~~	~~181696~~
19	363392
9	726784
~~4~~	~~1453568~~
~~2~~	~~2907136~~
1	5814272
	7006652

- Diviser par 2 le nombre de droite
- Multiplier par 2 le nombre de gauche
- Additionner les nombres de gauche qui sont face à un nombre impair

Quelle est la complexité de cet algorithme ?

Multiplication d'entiers

Entiers à 2 chiffres

$$\begin{aligned}(10 \cdot a + b) \cdot (10 \cdot c + d) &= 100 \cdot a \cdot c + 10 \cdot a \cdot d + 10 \cdot b \cdot c + b \cdot d \\ &= 100 \cdot a \cdot c + 10 \cdot (a \cdot d + b \cdot c) + b \cdot d\end{aligned}$$

$\rightarrow$ 4 multiplications

$$\begin{aligned}(a + b) \cdot (c + d) &= a \cdot c + a \cdot d + b \cdot c + b \cdot d \\ a \cdot d + b \cdot c &= (a + b) \cdot (c + d) - a \cdot c - b \cdot d\end{aligned}$$

$$\begin{aligned}(10 \cdot a + b) \cdot (10 \cdot c + d) &= 100 \cdot a \cdot c \\ &+ 10 \cdot ((a + b) \cdot (c + d) - a \cdot c - b \cdot d) \\ &+ b \cdot d\end{aligned}$$

$\rightarrow$ 3 multiplications

Multiplication d'entiers

Algorithme de Karatsuba (1962)

Appels récursifs pour la multiplication

$$\begin{aligned}x \cdot y &= (10^{n/2} \cdot a + b) \cdot (10^{n/2} \cdot c + d) \\ &= 10^n \cdot a \cdot c + 10^{n/2} \cdot (a \cdot d + b \cdot c) + b \cdot d \\ &= 10^n \cdot a \cdot c + 10^{n/2} \cdot ((a + b) \cdot (c + d) - a \cdot c - b \cdot d) + b \cdot d\end{aligned}$$

Complexité ? $O(n^{\log_2(3)})$

Algorithme de Harvey et van der Hoeven (2021)

Complexité en $O(n \log(n))$

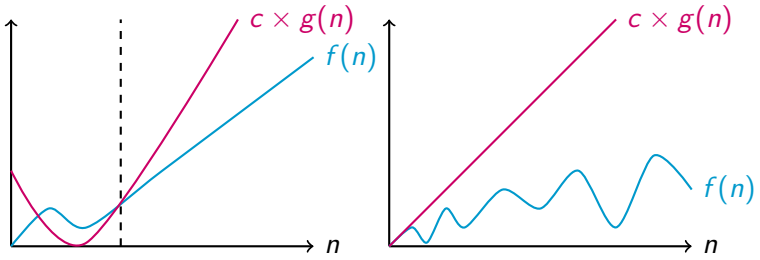
Notation de Landau

Definition

$O(g(n))$ est l'ensemble des fonctions $f(n)$ telles que

$$\exists n_0 \in \mathbb{N}, \exists c \in \mathbb{R}^+, \forall n \geq n_0, f(n) \leq c \times g(n)$$

$f(n) \in O(g(n))$ s'il existe une constante c est un seuil à partir duquel $f(n)$ est inférieure à $g(n)$ à un facteur c près



Complexité

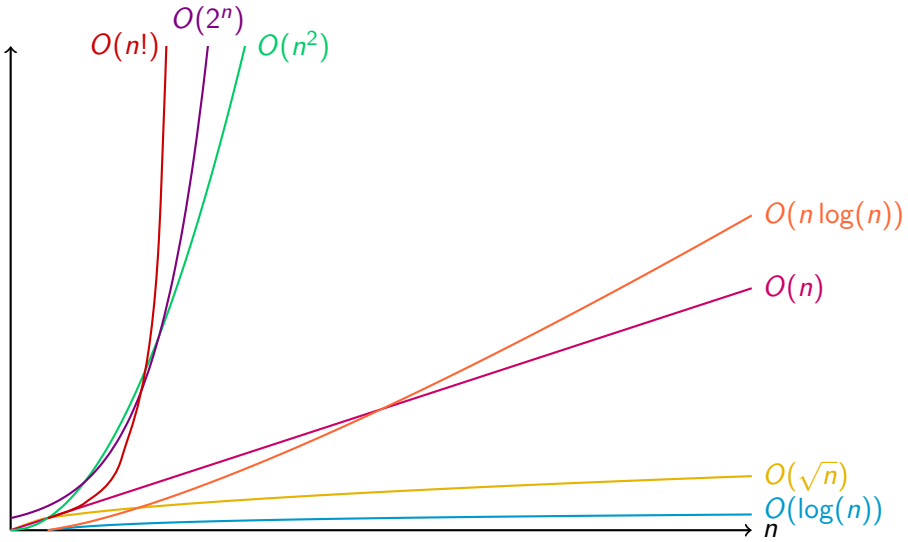
Règles de simplifications

- Les constantes peuvent être omises : $14n^2$ devient n^2
- n^a domine n^b si $a > b$: n^2 domine n
- Toute exponentielle domine tout polynôme : 2^n domine n^8
- Tout polynôme domine tout logarithme : n domine $\log(n)^5$

Grand O

- $O(f(n)) + O(g(n)) = O(f(n) + g(n)) = O(max(f(n) + g(n)))$
- $\sum_{i=1}^{k} O(f(n)) = O\left(\sum_{i=1}^{k} f(n)\right)$
- $O(f(n)) \times O(g(n)) = O(f(n) \times g(n))$
- $c \times O(f(n)) = O(c \times f(n)) = O(f(n))$

Comparaison



Rappels Mathématiques

Sommes

- $\sum_{k=1}^{m}(c \times a_k + b_k) = c \sum_{k=1}^{m} a_k + \sum_{k=1}^{m} b_k$
- $\sum_{k=m}^{p} a_k = \sum_{k=1}^{p} a_k - \sum_{k=1}^{m-1} a_k$
- $\sum_{k=1}^{m} k = 1 + 2 + \cdots + m = \frac{m(m+1)}{2}$
- $\sum_{k=1}^{m} k^2 = 1^2 + 2^2 + \cdots + m^2 = \frac{m(m+1)(2m+1)}{6}$
- $\sum_{k=0}^{m} r^k = r^0 + r^1 + \cdots + r^m = \frac{r^{m+1}-1}{r-1}, r \neq 1$

Règles de calculs

- Les instructions de base prennent un temps constant : $O(1)$
- Les affections dépendent de la partie droite
- On additionne les compléxités d'opérations de séquence : $O(f(n)) + O(g(n))$
- Branchement conditionnel on fait un max :

$$\max(O(f(n)), O(g(n))) = O(f(n)) + O(g(n))$$

Exemple

```
si (condition) alors      O(g(n))
  instruction 1           O(f1(n))
sinon
  instruction 2           O(f2(n))
```

$\Big\} = O(g(n) + f_1(n) + f_2(n))$

Règles de calculs

- Dans les boucles, il faut connaître le nombre d'itérations et la complexité du corps de la boucle

Exemple

Si on a m itérations

```
tant que (condition) faire   O(g(n))
  instructions               O(f(n))
```

$\Big\} = O\left(\sum_{i=1}^{m} g(n) + f(n)\right)$

Exemple

```
pour i de a à b faire
  instructions              O(f(n))
```

$\Big\} = O\left(\sum_{i=a}^{b} f(n)\right)$

Calcul de la complexité d'un algorithme

1. Calcul de la complexité de chaque "partie" de l'algorithme
2. Combinaison des complexités grâces aux règles de calculs
3. Simplification du résultat grâce aux règles de simplifications
   - Élimination des constantes
   - Conservation du (des) terme dominant

Notation dans ce cours

- $T(n)$ la complexité en temps
- $S(n)$ la complexité en espace

Calcul de la complexité d'un algorithme

Exemple (Tri à bulles)

A tableau de n entiers à trier

```
pour i de n à 2 faire ..........................
  pour j de 1 à i−1 faire ......................
    si (A[j+1] < A[j]) alors   O(1)
      tmp ← A[j+1]             O(1)
      A[j+1] ← A[j]            O(1)
      A[j] ← tmp               O(1)
```

$\Big\} = O(1)$ $\Big\} = O(i)$ $\Big\} = O(n^2)$

$T(n) \in O(n^2)$

Exemple (Multiplication comme à l'école)

Les entiers sont représentés par des tableaux X et Y de taille n

```
pour i de n à 1 faire                      } = O(n^2)
   r ← 0                     O(1)
   pour j de n à 1 faire                   } = O(n)
      tmp ← X[i]*Y[j] + r    O(1) }
      M[i][i+j] ← tmp%10     O(1) } = O(1)
      r ← tmp/10             O(1) }

res ← 0;                     O(1)
pour j de 1 à 2n faire                     } = O(n^2)
   res ← res * 10            O(1)
   pour i de 1 à n faire                   } = O(n)
      res ← res + M[i][j]    O(1)
                                           (ensemble) = O(n^2)
```

$T(n) \in O(n^2)$