Mathématiques pour l'informatique 1
notes de cours sur la seconde partie

L1 – Université Paris-Est, Marne-la-Vallée

Cyril Nicaud

Organisation

- Ce demi-cours est composé de 6 séances de cours et 6 séances de TD, de deux heures chacune.
- Il y a un TD toutes les deux semaines, tout le long du semestre.
- Le cours est en deux parties : 3 séances au début du semestre, et 3 autres au milieu du semestre.
- Pour ne pas rater une séance de cours ou de TD, renseignez-vous au secrétariat ou consultez ma page internet : `http://igm.univ-mlv.fr/~nicaud/L1.html`
- Le cours est sanctionné par un examen à la fin du semestre et un contrôle en milieu de semestre. La majeure partie des questions, mais pas toutes, ressemblent aux exercices fait en TD.

Le polycopié

- Le polycopié est juste une aide, la référence reste dans tout les cas le cours magistral : si une notion est abordée dans le cours mais n'est pas dans le polycopié, elle peut tomber à l'examen.
- Le polycopié est très factuel, je n'y décris ni les motivations, ni les intuitions, ni les conséquences. Il n'y a presque pas d'exemples non plus : il y a le cours magistral pour ça.
- Le polycopié vient d'être rédigé, il y aura donc inévitablement des coquilles et des erreurs. Si vous en repérez une, même si vous n'êtes pas sûr, envoyez-moi un email que je vérifie : `cyril.nicaud@univ-mlv.fr`
- **Le polycopié couvre les trois derniers cours seulement**, le premier polycopié a déjà été distribué.

CHAPITRE IV

Estimation asymptotique

Ce chapitre traite de l'estimation asymptotique de suites à valeurs positives. Il s'agit de travailler sur les objets mathématiques qui servent à mesurer les performances d'un algorithme.

IV.1 Propriétés vraies "à partir d'un certain rang"

Définition (Propriété) Une *propriété* sur $\mathbb{R}$ est une application de $\mathbb{R}$ dans $\{\texttt{Vrai}, \texttt{Faux}\}$.

Définition (Opération) Une *opération* sur $\mathbb{R}$ est une application de $\mathbb{R}\times\mathbb{R}$ dans $\mathbb{R}$. On note $x\star y$ l'image de (x, y) par l'opération $\star$, au lieu de $\star((x, y))$.

Définition (Propriété stable) Si $\star$ est une opération sur $\mathbb{R}$ et si P une propriété sur $\mathbb{R}$, on dit que P est stable pour $\star$ quand pour tout $x, y \in \mathbb{R}$, si $P(x) = \texttt{Vrai}$ et $P(y) = \texttt{Vrai}$, alors $P(x \star y) = \texttt{Vrai}$.

► La propriété "est un élément de $\mathbb{N}$" est stable pour l'addition et la multiplication.

Définition (APCR) Soit $(u_n)_{n\in\mathbb{N}}$ une suite à valeurs dans $\mathbb{R}$ et P une propriété sur $\mathbb{R}$. On dit que $(u_n)_{n\in\mathbb{N}}$ satisfait P à partir d'un certain rang, noté APCR, si il existe $n_0 \in \mathbb{N}$ tel que pour tout $n \geq n_0$, $P(n)$ est vraie.

► La suite $(u_n)_{n\in\mathbb{N}}$ définie pour tout $n \in \mathbb{N}$ par $u_n = 7n - 31$ est positive APCR.

Théorème IV.1 (APCR et stabilité) *Soit P une propriété stable par $\star$, et soient $(u_n)_{n\in\mathbb{N}}$ et $(v_n)_{n\in\mathbb{N}}$ deux suites qui satisfont P APCR. Alors la suite $(u_n \star v_n)_{n\in\mathbb{N}}$ satisfait P APCR.*

Définition (Bornée) Soit $(u_n)_{n\in\mathbb{N}}$ une suite à valeurs dans $\mathbb{R}$. On dit que $(u_n)_{n\in\mathbb{N}}$ est *bornée* s'il existe un réel $C \geq 0$ tel que pour tout $n \in \mathbb{N}$, $|u_n| \leq C$.

Théorème IV.2 (Bornée APCR = bornée) *Soit $(u_n)_{n\in\mathbb{N}}$ une suite bornée APCR, alors $(u_n)_{n\in\mathbb{N}}$ est bornée (tout le temps).*

IV.2 Hierarchie de fonctions

IV.2.1 Rappels sur le logarithme

► La fonction logarithme népérien, notée ln ou log, est une fonction continue strictement croissante de $\mathbb{R}_+^*$ dans $\mathbb{R}$. On a $\ln(1) = 0$ et $\ln(e) = 1$. On a de plus :

$$\lim_{x\to 0^+} \ln(x) = -\infty$$
$$\lim_{x\to+\infty} \ln(x) = +\infty$$

► La dérivée de $x \mapsto \ln x$ est $x \mapsto \frac{1}{x}$. Son inverse est $x \mapsto e^x$.

▶ Pour tout x, y dans $\mathbb{R}_+^*$, on a $\ln(xy) = \ln(x)+\ln(y)$. Pour tout $x \in \mathbb{R}_*^+$ et pour tout $y \in \mathbb{R}$, on a $\ln(x^y) = y\ln(x)$; en particulier, $-\ln(x) = \ln\frac{1}{x}$.

Définition ($\log_b$) Soit $b \in \mathbb{R}_+^*$, le *logarithme base b* est l'application de $\mathbb{R}_+^*$ dans $\mathbb{R}$ définie par

$$\log_b(x) = \frac{\ln x}{\ln b}.$$

Définition (Base b) Soit $b \geq 2$ un entier. Tout entier $n \geq 1$ s'écrit d'une unique façon sous la forme

$$n = \sum_{i=0}^{k} a_i\, b^i,$$

où $k \in \mathbb{N}$ et les a_i sont dans $\{0, \ldots, b-1\}$ avec $a_k \neq 0$. Les a_i s'appellent l'*écriture de n en base b*.

▶ On utilise habituellement la base 10. Dans les ordinateurs classiques, les nombres entiers sont représentés en base 2.

Théorème IV.3 (Nombre de chiffres) *Soit $n \geq 1$ et $b \geq 2$ deux entiers. Le nombre de chiffres dans l'écriture en base b de n est $\lceil \log_b(n+1) \rceil$, où $\lceil x \rceil$ est la* partie entière supérieure *de x.*

IV.2.2 Vitesse de croissance

Définition (Ordre de croissance) Soit $(f_n)_{n\in\mathbb{N}}$ et $(g_n)_{n\in\mathbb{N}}$ deux suites réelles qui sont strictement positives APCR. On dit que $(f_n)_{n\in\mathbb{N}}$ croît plus lentement que $(g_n)_{n\in\mathbb{N}}$, noté $f_n \prec g_n$ quand

$$\lim_{n\to\infty} \frac{f_n}{g_n} = 0.$$

▶ Remarque : le rapport est bien défini APCR.

Théorème IV.4 (Transistivité) *La relation $\prec$ sur l'ensemble des suites réelles strictement positives APCR est transitive.*

▶ On note $\mathbf{1}$ la suite constante égale à 1.

Théorème IV.5 (Echelle) *Pour tous réels ϵ et c avec $0 < \epsilon < 1 < c$, on a*

$$\mathbf{1} \prec \log n \prec (\log n)^c \prec n^\epsilon \prec n^c \prec c^n \prec n! \prec n^n \prec c^{c^n}.$$

Théorème IV.6 (Inverse) *Soient $(f_n)_{n\in\mathbb{N}}$ et $(g_n)_{n\in\mathbb{N}}$ strictement positives APCR, les suites $(\frac{1}{f_n})_{n\in\mathbb{N}}$ et $(\frac{1}{g_n})_{n\in\mathbb{N}}$ sont définies APCR et on a*

$$f_n \prec g_n \Leftrightarrow \frac{1}{g_n} \prec \frac{1}{f_n}.$$

Théorème IV.7 (Constante multiplicative) *Soient $(f_n)_{n\in\mathbb{N}}$ et $(g_n)_{n\in\mathbb{N}}$ telles que $f_n \prec g_n$. Pour toute constante strictement positive c on a $f_n \prec c\cdot g_n$.*

Théorème IV.8 (Cas fréquent) *Soient $f_n = n^a(\log n)^b$ et $g_n = n^\alpha(\log n)^\beta$, définies pour $n \geq 1$. On a*

$$f_n \prec g_n \Leftrightarrow \begin{cases} a < \alpha \\ \text{ou} \\ a = \alpha \text{ et } b < \beta \end{cases}$$

Définition (Suites équivalentes) On dit que deux suites non nulles APCR $(f_n)_{n\in\mathbb{N}}$ et $(g_n)_{n\in\mathbb{N}}$ sont équivalentes, noté $f_n \sim g_n$ quand

$$\lim_{n\to\infty} \frac{f_n}{g_n} = 1.$$

IV.3 La notation $\mathcal{O}$

IV.3.1 Définition

Définition ($\mathcal{O}$) Soient $(f_n)_{n\in\mathbb{N}}$ et $(g_n)_{n\in\mathbb{N}}$ deux suites réelles. On dit que $f_n \in \mathcal{O}(g_n)$ quand il existe une constante $C \in \mathbb{R}_+^*$ telle que APCR on ait

$$|f_n| \leq C \cdot |g_n|.$$

Autrement dit, il existe entier $n_0 \in \mathbb{N}$ tels que pour tout $n \geq n_0$,

$$|f_n| \leq C \cdot |g_n|.$$

▶ $\mathcal{O}(g_n)$ est donc un ensemble de suites.

▶ Si $(g_n)_{n\in\mathbb{N}}$ est non nulle APCR, $f_n \in O(g_n)$ signifie que la suite $\frac{f_n}{g_n}$ est bornée APCR.

▶ $\mathcal{O}(\mathbf{1})$ est l'ensemble des suites bornées APCR, qui est aussi l'ensemble des suites bornées.

Théorème IV.9 (Bornées $\subset$ $\mathcal{O}(1)$) *Toute suite bornée est dans $\mathcal{O}(1)$. La réciproque est fausse.*

IV.3.2 Autres notations similaires

Définition (Ω) On note $f_n \in \Omega(g_n)$ quand il existe un réel $C > 0$ tel que APCR on ait $|f_n| \geq C \cdot |g_n|$.

▶ On a donc $f_n \in \mathcal{O}(g_n) \Leftrightarrow g_n \in \Omega(f_n)$.

Définition (Θ) On note $f_n \in \Theta(g_n)$ quand on a à la fois $f_n \in \mathcal{O}(g_n)$ et $f_n \in \Omega(g_n)$.

IV.3.3 Définition équivalente

Théorème IV.10 (Définition équivalente) *On a $f_n \in O(g_n)$ si et seulement si il existe une suite $(h_n)_{n\in\mathbb{N}}$, bornée, telle que $f_n = h_n g_n$ APCR.*

▶ Cette définition alternative est parfois plus simple à utiliser que la définition initiale.

IV.3.4 Manipulation des O

Définition (Opérations sur des ensembles de suites) Soit $(u_n)_{n\in\mathbb{N}}$ une suite réelle et F et G deux ensembles de suites. On utilise les définitions suivantes :

$$u_n + F = \{(u_n + v_n)_{n\in\mathbb{N}} \mid (v_n)_{n\in\mathbb{N}} \in F\}$$
$$u_n \times F = \{(u_n \times v_n)_{n\in\mathbb{N}} \mid (v_n)_{n\in\mathbb{N}} \in F\}$$
$$F + G = \{(u_n + v_n)_{n\in\mathbb{N}} \mid (u_n)_{n\in\mathbb{N}} \in F,\ (v_n)_{n\in\mathbb{N}} \in G\}$$
$$F \times G = \{(u_n \times v_n)_{n\in\mathbb{N}} \mid (u_n)_{n\in\mathbb{N}} \in F,\ (v_n)_{n\in\mathbb{N}} \in G\}$$

Théorème IV.11 (Règles de calcul) *On a les règles de calculs suivantes, pour $(f_n)_{n\in\mathbb{N}}$ et $(g_n)_{n\in\mathbb{N}}$ positive APCR :*

- $f_n \in \mathcal{O}(f_n)$.
- $c \times O(f_n) \subset \mathcal{O}(f_n)$, *pour* $c\imath\mathbb{R}$.
- $\mathcal{O}(f_n) \times \mathcal{O}(g_n) \subset \mathcal{O}(f_n \times g_n)$.
- $\mathcal{O}(f_n) + \mathcal{O}(g_n) \subset \mathcal{O}(\max(f_n, g_n))$.
- *Si* $f_n \in \mathcal{O}(g_n)$ *alors* $\mathcal{O}(f_n) + \mathcal{O}(g_n) \subset \mathcal{O}(g_n)$.
- *Si* $f_n \prec g_n$ *alors* $f_n \in \mathcal{O}(g_n)$.
- *Si* $f_n \sim g_n$ *alors* $f_n \in \mathcal{O}(g_n)$.

▶ Attention, si $((f_n)_{n\in\mathbb{N}})$ est strictement positive APCR, $O(f_n)^{-1} = \Omega(1/f_n)$.

IV.4 Approximation par intégrale

Soit f une fonction de $\mathbb{R}_*^+$ dans $\mathbb{R}$, monotone (on prendra croissante pour illustrer la méthode) et intégrable. On considère la série de terme général f_n : $S_n = \sum_{i=1}^n f(i)$, dont on cherche un développement asymptotique.

En utilisant la croissance de f, on a pour tout $x \in [i, i+1]$, avec $i \in \mathbb{N}^*$: $f(i) \leq f(x) \leq f(i+1)$. On intègre sur l'intervalle : $f(i) \leq \int_1^n f(x)dx \leq f(i+1)$. On somme pour i de 1 à $n-1$:

$$S_n - f_n \leq \int_1^n f(x)dx \leq S_n - f(1)$$

Et donc

$$\int_1^n f(x)dx + f(1) \leq S_n \leq \int_1^n f(x)dx + f_n$$

Ce qui suffit souvent à trouver un équivalent asymptotique à S_n.

CHAPITRE V

Complexité d'un algorithme

▶ **Important :** Ce chapitre est beaucoup plus de l'informatique que des mathématiques et se prête mal à des notes succinctes comme le reste du cours. En conséquence, il y aura très peu d'informations dans le poly : l'essentiel des considérations du cours **ne sont pas présentes ci-dessous**.

V.1 Introduction

Définition (Algorithme) Un *algorithme* est un procédé automatique pour résoudre un problème en un nombre fini d'étapes.

▶ Le but de ce chapitre est de donner des outils pour comparer différentes solutions algorithmiques à un problème donné.

▶ Pour quantifier les performances d'un algorithme on doit se munir d'une notion de *taille* sur les entrées.

▶ La *complexité* d'un algorithme est la quantité de ressources nécessaires pour traiter des entrées. On la voit comme une fonction de n, la taille de l'entrée.

▶ Les principales ressources mesurées sont le *temps* (nombre d'instructions utilisées) et l'*espace* (quantité d'espace mémoire nécessaire).

▶ On distingue plusieurs types d'analyses de complexité : l'analyse dans le *meilleur des cas*, le *pire des cas* et *en moyenne*. Pour ce cours on étudie exclusivement le pire des cas. Donc si $T(\mathcal{A})$ est le nombre d'instructions nécessaires pour que l'algorithmes fasse ses calculs sur l'entrée $\mathcal{A}$, on s'intéresse à la suite $(t_n)_{n\in\mathbb{N}}$ définie par

$$t_n = \max_{|\mathcal{A}|=n} T(\mathcal{A}),$$

où $|\mathcal{A}|$ est la taille de l'entrée $\mathcal{A}$.

▶ Il n'est souvent pas pertinent d'essayer de quantifier trop précisément t_n, vu qu'on raisonne au niveau de l'algorithme et non d'une implatation. On se contente donc d'estimer t_n avec un ordre de grandeur en Θ ou $\mathcal{O}$. Un résultat typique : la complexité de l'algorithme de tri par insertion est en $\mathcal{O}(n^2)$.

V.2 Principes généraux

▶ **(ligne la plus effectuée)** La façon la plus simple dévaluer la complexité d'un algorithme est la suivante : un programme est constitué d'un nombre fini de lignes. Appelons-les ℓ_1 à ℓ_k. Soit n_1 ... n_k le nombre de fois qu'elles sont effectuées. La complexité de l'algo est $\sum \ell_i n_i$. Soit ℓ_j l'une des lignes qui est effectuée le plus souvent. Si toutes les lignes s'effectuent en temps $O(1)$, la complexité de l'algorithme est majorée par $kn_jO(1)$ soit $O(n_j)$.

▶ Attention aux instructions qui appellent d'autres fonctions et qui ne s'éxécutent pas en temps constant.

▶ Quand le programme contient une ou plusieurs boucles imbriquées, on se retrouve à estimer des sommes. On peut souvent utiliser le théorème suivant.

Théorème V.1 (Sommation) *Soit $(f_n)_{n\in\mathbb{N}}$ et $(g_n)_{n\in\mathbb{N}}$ deux suites positives, avec $f_n \in \mathcal{O}(g_n)$. On suppose de plus que $\sum_{i=0}^{n} g_i$ n'est pas toujours nul. Alors*

$$\sum_{i=0}^{n} f_i = O\left(\sum_{i=0}^{n} g_i\right).$$

Plus généralement, si $\alpha : \mathbb{N} \to \mathbb{N}$ est une application croissante qui tend vers l'infini et $a \in \mathbb{N}$:

$$\sum_{i=a}^{\alpha(n)} f_i = O\left(\sum_{i=a}^{\alpha(n)} g_i\right).$$

V.3 Classes de complexités classiques

On voit souvent apparaître les complexités suivantes :

• $\mathcal{O}(\log n)$ Ce sont des algorithmes très rapides. Exemples typiques : recherche dichotomique, exponentiation rapide, *etc.*

• $\mathcal{O}(n)$ (on dit *linéaire*). Typiquement quand on parcourt un tableau ou une liste un nombre borné de fois : recherche dans un tableau, minimum d'une liste, *etc.*

• $\mathcal{O}(n \log n)$. Vous l'avez principalement vu pour les algorithmes efficaces de tri : tri rapide, tri fusion, tri par tas, *etc.* Cette complexité apparaît régulièrement lorsque l'on fait du "diviser pour régner".

• $\mathcal{O}(n^2)$ (on dit *quadratique*). Quand on manipule des tableaux à deux dimensions, ou qu'on effectue un assez grand nombre de calculs sur un tableau à une dimension : somme de deux matrices, transposée d'une matrice, tri insertion, tri bulle, tri selection, *etc.*

V.4 Trois exemples importants

V.4.1 Dichotomie

▶ On veut chercher si un entier x est dans un tableau trié T de taille n.

```
int dicho(int *t, int n, int x) {
  int a,b,mid;
  a = 0;  b = n;
  while(a <= b) {
    mid = (b+a)/2;
    if (t[mid] == x)
      return 1;
    if (t[mid] < x)
      a = mid + 1;
    else
      b = mid - 1;
  }
  return 0;
}
```

Théorème V.2 *La complexité de l'algorithme* `dicho` *est en $\mathcal{O}(\log n)$.*

V.4.2 Exponentiation rapide

▶ On veut calculer x^n, où $n \in \mathbb{N}$ mesure la taille de l'entrée.

```
float puissance(float x,int n) {
  if (n==0)
    return 1;
  if (n&1)
    return x*puissance(x,n-1);
  return puissance(x*x,n/2);
}
```

Théorème V.3 *La complexité de l'algorithme* `puissance` *est en $\mathcal{O}(\log n)$.*

V.4.3 Schéma de Hörner

▶ On veut calculer $P(x)$, où x est un réel (`float`) et P est un polynôme de degré n, réprésenté par un tableau : si $P(X) = a_0 + a_1 X + \ldots + a_n X^n$, alors pour tout $i \in \{0, \ldots, n\}$ on a $P[i] = a_i$.

```
float Horner(float P[], int n, float x) {
  int i; float r = P[n];
  for(i=n-1;i>=0;i--)
    r = (r*x)+P[i];
  return r;
}
```

Théorème V.4 *La complexité de l'algorithme* `Horner` *est en $\mathcal{O}(n)$.*