Algorithmique

ISIMA FISA1

Introduction, complexité algorithmique

Florent Foucaud

IUT CLERMONT AUVERGNE

Aurillac - Clermont-Ferrand - Le Puy-en-Velay
Montluçon - Moulins - Vichy

Organisation du cours

- 7 séances de cours-TD + DS la 8e séance (29 novembre)
- contact : `florent.foucaud@uca.fr`
- supports disponibles sur `https://perso.limos.fr/ffoucaud/Teaching/index.html`
- en suivant, 4 séances avec Renaud Chicoisne `https://sites.google.com/view/renaud-chicoisne` + évaluation

Programme :

- Complexité algorithmique en temps
- Terminaison et correction d'un algorithme
- Complexité avancée : diviser pour régner, programmation dynamique

Historique

- Premiers algorithmes :
  - Babylone, -2500 / Égypte ancienne, -1500 / Inde, -800 : premiers algorithmes (ex : divisions)
  - Grèce antique, -250 : nombres premiers (Euclide, Ératosthène)
  - Inde, 450 : résolution d'équations (Kuttaka)
  - monde arabo-persan, 850 : cryptographie, arithmétique

    (Muhammad ibn Musa al Khwarizmi, mathématicien le plus lu au Moyen-Âge)
  - 1230 : → Alchoarismi → Algorismo : notion d'algorithme

Muhammad ibn Musa al Khwarizmi
(780-850)

al-Kitab al-mukhtasar fi hisab
al-jabr wal-muqabala (820)

Historique

- Premiers algorithmes :
  - Babylone, -2500 / Égypte ancienne, -1500 / Inde, -800 : premiers algorithmes (ex : divisions)
  - Grèce antique, -250 : nombres premiers (Euclide, Ératosthène)
  - Inde, 450 : résolution d'équations (Kuttaka)
  - monde arabo-persan, 850 : cryptographie, arithmétique
    (Muhammad ibn Musa al Khwarizmi, mathématicien le plus lu au Moyen-Âge)
  - 1230 : → Alchoarismi → Algorismo : notion d'algorithme

- David Hilbert, 1928 : existe-t-il un algorithme pour résoudre toute question mathématique ? (*Entscheidungsproblem*, traduit en “problème de décision”)

David Hilbert (1862-1943)

Historique

- Premiers algorithmes :
  - Babylone, -2500 / Égypte ancienne, -1500 / Inde, -800 : premiers algorithmes (ex : divisions)
  - Grèce antique, -250 : nombres premiers (Euclide, Ératosthène)
  - Inde, 450 : résolution d'équations (Kuttaka)
  - monde arabo-persan, 850 : cryptographie, arithmétique
    (Muhammad ibn Musa al Khwarizmi, mathématicien le plus lu au Moyen-Âge)
  - 1230 : → Alchoarismi → Algorismo : notion d'algorithme

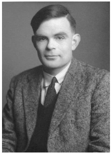
- David Hilbert, 1928 : existe-t-il un algorithme pour résoudre toute question mathématique ? (*Entscheidungsproblem*, traduit en "problème de décision")
- Kurt Gödel, 1931 : théorème d'incomplétude (logique mathématique)

Kurt Gödel (1906-1978)

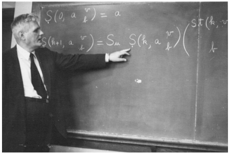
Historique

- Premiers algorithmes :
  - Babylone, -2500 / Égypte ancienne, -1500 / Inde, -800 : premiers algorithmes (ex : divisions)
  - Grèce antique, -250 : nombres premiers (Euclide, Ératosthène)
  - Inde, 450 : résolution d'équations (Kuttaka)
  - monde arabo-persan, 850 : cryptographie, arithmétique
    (Muhammad ibn Musa al Khwarizmi, mathématicien le plus lu au Moyen-Âge)
  - 1230 : → Alchoarismi → Algorismo : notion d'algorithme

- David Hilbert, 1928 : existe-t-il un algorithme pour résoudre toute question mathématique ? (*Entscheidungsproblem*, traduit en “problème de décision”)
- Kurt Gödel, 1931 : théorème d'incomplétude (logique mathématique)
- Alonzo Church et Alan Turing, 1936 : solution à l'Entscheidungsproblem : NON !
  (→ problème de l'arrêt, machines de Turing)



Alan Turing (1912-1954)



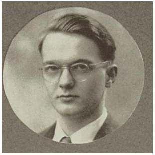
Alonzo Church (1903-1995)

Historique

- Premiers algorithmes :
  - Babylone, -2500 / Égypte ancienne, -1500 / Inde, -800 : premiers algorithmes (ex : divisions)
  - Grèce antique, -250 : nombres premiers (Euclide, Ératosthène)
  - Inde, 450 : résolution d'équations (Kuttaka)
  - monde arabo-persan, 850 : cryptographie, arithmétique
    (Muhammad ibn Musa al Khwarizmi, mathématicien le plus lu au Moyen-Âge)
  - 1230 : → Alchoarismi → Algorismo : notion d'algorithme

- David Hilbert, 1928 : existe-t-il un algorithme pour résoudre toute question mathématique ? (*Entscheidungsproblem*, traduit en “problème de décision”)
- Kurt Gödel, 1931 : théorème d'incomplétude (logique mathématique)
- Alonzo Church et Alan Turing, 1936 : solution à l'Entscheidungsproblem : NON !
  (→ problème de l'arrêt, machines de Turing)
- Alan Cobham et Jack Edmonds, 1965 :
  un algorithme est efficace s'il est polynômial

Jack Edmonds (1934-)



Alan B. Cobham (1927-2011)

Historique

- Premiers algorithmes :
  - Babylone, -2500 / Égypte ancienne, -1500 / Inde, -800 : premiers algorithmes (ex : divisions)
  - Grèce antique, -250 : nombres premiers (Euclide, Ératosthène)
  - Inde, 450 : résolution d'équations (Kuttaka)
  - monde arabo-persan, 850 : cryptographie, arithmétique (Muhammad ibn Musa al Khwarizmi, mathématicien le plus lu au Moyen-Âge)
  - 1230 : → Alchoarismi → Algorismo : notion d'algorithme

- David Hilbert, 1928 : existe-t-il un algorithme pour résoudre toute question mathématique ? (*Entscheidungsproblem*, traduit en “problème de décision”)
- Kurt Gödel, 1931 : théorème d'incomplétude (logique mathématique)
- Alonzo Church et Alan Turing, 1936 : solution à l'Entscheidungsproblem : NON ! (→ problème de l'arrêt, machines de Turing)
- Alan Cobham et Jack Edmonds, 1965 : un algorithme est efficace s'il est polynômial
- Années 1970 : théorie de la complexité

Problématiques

Question

Mon algorithme se termine-t-il, ou au contraire, peut-il boucler à l'infini ?

Problématiques

Question

Mon algorithme se termine-t-il, ou au contraire, peut-il boucler à l'infini ?

Question

Mon algorithme est-il correct ? Calcule-t-il bien ce qu'il est sensé calculer ?

Problématiques

Question

Mon algorithme se termine-t-il, ou au contraire, peut-il boucler à l'infini ?

Question

Mon algorithme est-il correct ? Calcule-t-il bien ce qu'il est sensé calculer ?

Question

Mon algorithme est-il efficace ? Ou peut-on trouver un meilleur algorithme pour résoudre le même problème ?

Complexité d'un problème algorithmique

Problème algorithmique : une entrée, une sortie

Exemples :

- Multiplier deux nombres n_1 et n_2 encodés en binaire
- Trier une liste de n entiers
- Trouver un plus court chemin de A à B dans un graphe à n sommets
- Couvrir un réseau à n sommets avec k antennes radio

Complexité d'un problème algorithmique

Problème algorithmique : une entrée, une sortie

Exemples :

- Multiplier deux nombres n_1 et n_2 encodés en binaire
- Trier une liste de n entiers
- Trouver un plus court chemin de A à B dans un graphe à n sommets
- Couvrir un réseau à n sommets avec k antennes radio

Algorithme : série d'instructions qui résoud un problème algorithmique donné
pour un humain : manuel, instructions, recette de cuisine...
pour une machine : ≈ programme informatique

Complexité d'un problème algorithmique

Problème algorithmique : une entrée, une sortie

Exemples :

- Multiplier deux nombres n_1 et n_2 encodés en binaire
- Trier une liste de n entiers
- Trouver un plus court chemin de A à B dans un graphe à n sommets
- Couvrir un réseau à n sommets avec k antennes radio

Algorithme : série d'instructions qui résoud un problème algorithmique donné
pour un humain : manuel, instructions, recette de cuisine…
pour une machine : ≈ programme informatique

Variantes :

- problèmes online
- problèmes en streaming
- problèmes de requêtes (grandes masses de données)
- …

Complexité algorithmique

Complexité d’un algorithme : quantité de ressources nécessaires à l’algorithme, en fonction de la taille n de l’entrée

Complexité algorithmique

Complexité d'un algorithme : quantité de ressources nécessaires à l'algorithme, en fonction de la taille n de l'entrée

Complexité d'un problème algorithmique P : meilleure complexité d'un algorithme qui résoud le problème P

Complexité algorithmique

Complexité d'un algorithme : quantité de ressources nécessaires à l'algorithme, en fonction de la taille n de l'entrée

Complexité d'un problème algorithmique P : meilleure complexité d'un algorithme qui résoud le problème P

Quelles ressources mesure-t-on ?

- Complexité **en temps** $T(n)$: plus petit nombre d'étapes
- Complexité **en mémoire** $M(n)$: plus petit espace mémoire
- ...

Complexité algorithmique

Complexité d’un algorithme : quantité de ressources nécessaires à l’algorithme, en fonction de la taille n de l’entrée

Complexité d’un problème algorithmique P : meilleure complexité d’un algorithme qui résoud le problème P

Quelles ressources mesure-t-on ?

- Complexité **en temps** $T(n)$: plus petit nombre d’étapes
- Complexité **en mémoire** $M(n)$: plus petit espace mémoire
- ...

Remarque : $M(n) \leq T(n)$

Complexité algorithmique

Complexité d'un algorithme : quantité de ressources nécessaires à l'algorithme, en fonction de la taille n de l'entrée

Complexité d'un problème algorithmique P : meilleure complexité d'un algorithme qui résoud le problème P

Quelles ressources mesure-t-on ?

- Complexité **en temps** $T(n)$: plus petit nombre d'étapes
- Complexité **en mémoire** $M(n)$: plus petit espace mémoire
- ...

Remarque : $M(n) \leq T(n)$

Comment les mesure-t-on ?

- Complexité **dans le pire des cas** (coût maximum)
- Complexité **en moyenne** (coût moyen)
  → selon une distribution probabiliste des entrées
- ...

Taille de l’entrée

Attention à l’encodage !

Taille de l'entrée

Attention à l'encodage !

- Entier n quelconque $\rightarrow \lceil \log_2(n) \rceil$ bits

Taille de l’entrée

Attention à l’encodage !

- Entier n quelconque → $\lceil \log_2(n) \rceil$ bits
- Entier en langage C → borné par 8 octets (constante)

Taille de l'entrée

Attention à l'encodage !

- Entier n quelconque → $\lceil \log_2(n) \rceil$ bits
- Entier en langage C → borné par 8 octets (constante)
- Tableau d'entiers de longueur n → $n \times$ (taille d'un entier)

Taille de l'entrée

Attention à l'encodage !

- Entier n quelconque → $\lceil \log_2(n) \rceil$ bits
- Entier en langage C → borné par 8 octets (constante)
- Tableau d'entiers de longueur n → $n\times$ (taille d'un entier)
- Graphe à n sommets et m arêtes → $(n+m)\times$ (taille d'un entier)

Explosion combinatoire

Complexité algorithmique en temps pour un problème donné :
$T(n)$ opérations pour une entrée de taille n

Meilleurs problèmes : complexité constante $T(n) \rightarrow 1, 10$ ou logarithmique $T(n) \rightarrow \log_2(n), 3\log(n)$...

Meilleurs problèmes : complexité linéaire $T(n) \rightarrow 10n, 2n, 1000n, n$ ...

Problèmes “raisonnables” : complexité polynomiale $T(n) \rightarrow 4n^2, 10n^3, n^{1000}$ ...
(en pratique : au-delà de $O(n^2)$, c’est compliqué)

Problèmes difficiles : complexité exponentielle $T(n) \rightarrow 2^n, n!, n^n, 2^{2^n}$ ...
$\rightarrow$ Intuition : *on teste toutes les solutions possibles*

Explosion combinatoire

Complexité algorithmique en temps pour un problème donné :

$T(n)$ opérations pour une entrée de taille n

Meilleurs problèmes : complexité constante $T(n) \to 1, 10$ ou logarithmique $T(n) \to \log_2(n), 3\log(n)$...

Meilleurs problèmes : complexité linéaire $T(n) \to 10n, 2n, 1000n, n$ ...

Problèmes “raisonnables” : complexité polynomiale $T(n) \to 4n^2, 10n^3, n^{1000}$ ...

(en pratique : au-delà de $O(n^2)$, c'est compliqué)

Problèmes difficiles : complexité exponentielle $T(n) \to 2^n, n!, n^n, 2^{2^n}$ ...

→ Intuition : *on teste toutes les solutions possibles*

$T(n)$	$n = 10$	$n = 50$	$n = 100$	$n = 200$	$n = 300$
n	10	50	100	200	300
$100n$	1000	5000	10000	20000	30000
n^2	100	2500	10000	40000	90000
2^n	1024	(16 chiffres)	(31 chiffres)	(60 chiffres)	(91 chiffres)
$n!$	3628800	(64 chiffres)	(157 chiffres)	(374 chiffres)	(614 chiffres)

Explosion combinatoire

Complexité algorithmique en temps pour un problème donné :

$T(n)$ opérations pour une entrée de taille n

Meilleurs problèmes : complexité constante $T(n) \to 1, 10$ ou logarithmique $T(n) \to \log_2(n), 3\log(n)$...

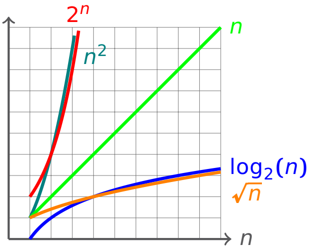
Meilleurs problèmes : complexité linéaire $T(n) \to 10n, 2n, 1000n, n$ ...

Problèmes “raisonnables” : complexité polynomiale $T(n) \to 4n^2, 10n^3, n^{1000}$ ...

(en pratique : au-delà de $O(n^2)$, c’est compliqué)

Problèmes difficiles : complexité exponentielle $T(n) \to 2^n, n!, n^n, 2^{2^n}$ ...

→ Intuition : *on teste toutes les solutions possibles*



n varie de 0 à 10

Explosion combinatoire

Complexité algorithmique en temps pour un problème donné :

$T(n)$ opérations pour une entrée de taille n

Meilleurs problèmes : complexité constante $T(n) \rightarrow 1, 10$ ou logarithmique $T(n) \rightarrow \log_2(n), 3\log(n)$...

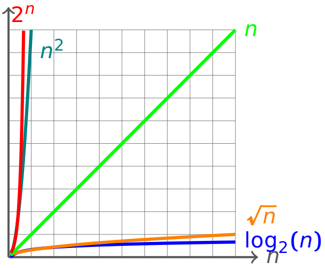
Meilleurs problèmes : complexité linéaire $T(n) \rightarrow 10n, 2n, 1000n, n$ ...

Problèmes "raisonnables" : complexité polynomiale $T(n) \rightarrow 4n^2, 10n^3, n^{1000}$ ...

(en pratique : au-delà de $O(n^2)$, c'est compliqué)

Problèmes difficiles : complexité exponentielle $T(n) \rightarrow 2^n, n!, n^n, 2^{2^n}$ ...

→ Intuition : *on teste toutes les solutions possibles*



n varie de 0 à 100

Explosion combinatoire

Complexité algorithmique en temps pour un problème donné :

$T(n)$ opérations pour une entrée de taille n

Meilleurs problèmes : complexité constante $T(n) \rightarrow 1, 10$ ou logarithmique $T(n) \rightarrow \log_2(n), 3\log(n)$...

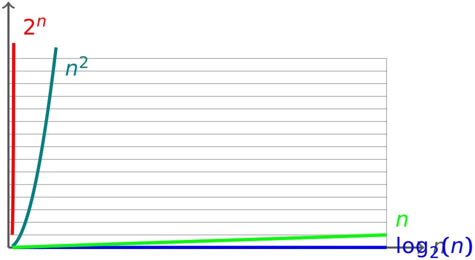
Meilleurs problèmes : complexité linéaire $T(n) \rightarrow 10n, 2n, 1000n, n$ ...

Problèmes "raisonnables" : complexité polynomiale $T(n) \rightarrow 4n^2, 10n^3, n^{1000}$ ...

(en pratique : au-delà de $O(n^2)$, c'est compliqué)

Problèmes difficiles : complexité exponentielle $T(n) \rightarrow 2^n, n!, n^n, 2^{2^n}$ ...

→ Intuition : *on teste toutes les solutions possibles*



n varie de 0 à 1000 (échelle aplatie)

Un exemple simple : calcul de la factorielle

factorielle(*n*) :

- Entrée : un entier n
- Sortie : $n!$

- $i = n$
- $res = 1$
- Tant que $i > 1$ faire :
  - $res = res * i$
  - $i = i - 1$
- Retourner res

Opérations élémentaires :

- affectations
- opérations arithmétiques
- opérations booléennes
- accès à un tableau

Un exemple simple : calcul de la factorielle

factorielle(n) :

- Entrée : un entier n
- Sortie : $n!$

- $i = n$ 1 opération : affectation
- $res = 1$
- Tant que $i > 1$ faire :
  - $res = res * i$
  - $i = i - 1$
- Retourner res

Opérations élémentaires :

- affectations
- opérations arithmétiques
- opérations booléennes
- accès à un tableau

Un exemple simple : calcul de la factorielle

factorielle(n) :

- Entrée : un entier n
- Sortie : $n!$

- $i = n$ 1 opération : affectation
- $res = 1$ 1 opération : affectation
- Tant que $i > 1$ faire :
  - $res = res * i$
  - $i = i - 1$
- Retourner res

Opérations élémentaires :

- affectations
- opérations arithmétiques
- opérations booléennes
- accès à un tableau

Un exemple simple : calcul de la factorielle

factorielle(n) :

- Entrée : un entier n
- Sortie : $n!$

- $i = n$ 1 opération : affectation
- $res = 1$ 1 opération : affectation
- Tant que $i > 1$ faire : 1 opération par tour de boucle + 1 opération supplémentaire
  - $res = res * i$
  - $i = i - 1$
- Retourner res

Opérations élémentaires :

- affectations
- opérations arithmétiques
- opérations booléennes
- accès à un tableau

Un exemple simple : calcul de la factorielle

factorielle(n) :

- Entrée : un entier n
- Sortie : $n!$

- $i = n$ — 1 opération : affectation
- $res = 1$ — 1 opération : affectation
- Tant que $i > 1$ faire : — 1 opération par tour de boucle + 1 opération supplémentaire
  - $res = res * i$ — 2 opérations : affectation et multiplication
  - $i = i - 1$
- Retourner res

Opérations élémentaires :

- affectations
- opérations arithmétiques
- opérations booléennes
- accès à un tableau

Un exemple simple : calcul de la factorielle

factorielle(n) **:**

- Entrée : un entier n
- Sortie : $n!$

- $i = n$ — 1 opération : affectation
- $res = 1$ — 1 opération : affectation
- Tant que $i > 1$ faire : — 1 opération par tour de boucle + 1 opération supplémentaire
  - $res = res * i$ — 2 opérations : affectation et multiplication
  - $i = i - 1$ — 2 opérations (idem)
- Retourner res

Opérations élémentaires :

- affectations
- opérations arithmétiques
- opérations booléennes
- accès à un tableau

Un exemple simple : calcul de la factorielle

factorielle(n) **:**

- Entrée : un entier n
- Sortie : $n!$

- $i = n$ — 1 opération : affectation
- $res = 1$ — 1 opération : affectation
- Tant que $i > 1$ faire : — 1 opération par tour de boucle + 1 opération supplémentaire
  – $res = res * i$ — 2 opérations : affectation et multiplication
  – $i = i - 1$ — 2 opérations (idem)
- Retourner res — 1 opération

Opérations élémentaires :

- affectations
- opérations arithmétiques
- opérations booléennes
- accès à un tableau

Un exemple simple : calcul de la factorielle

factorielle(n) :

- Entrée : un entier n
- Sortie : $n!$

- $i = n$ — 1 opération : affectation
- $res = 1$ — 1 opération : affectation
- Tant que $i > 1$ faire : — 1 opération par tour de boucle + 1 opération supplémentaire
  - $res = res * i$ — 2 opérations : affectation et multiplication
  - $i = i - 1$ — 2 opérations (idem)
- Retourner res — 1 opération

Au total on a :

$$
\begin{aligned}
T(n) &= 2 + 5(n-1) + 1 + 1 \\
&= 2 + 5n - 5 + 1 + 1 \\
&= 5n - 1
\end{aligned}
$$

Un exemple simple : calcul de la factorielle

factorielle(n) :

- Entrée : un entier n
- Sortie : $n!$

- $i = n$ — 1 opération : affectation
- $res = 1$ — 1 opération : affectation
- Tant que $i > 1$ faire : — 1 opération par tour de boucle + 1 opération supplémentaire
  - $res = res * i$ — 2 opérations : affectation et multiplication
  - $i = i - 1$ — 2 opérations (idem)
- Retourner res — 1 opération

Au total on a :

$$\begin{aligned} T(n) &= 2 + 5(n-1) + 1 + 1 \\ &= 2 + 5n - 5 + 1 + 1 \\ &= 5n - 1 \end{aligned}$$

Quelle complexité ?

Un exemple simple : calcul de la factorielle

factorielle(n) :

- Entrée : un entier n
- Sortie : $n!$

- $i = n$ — 1 opération : affectation
- $res = 1$ — 1 opération : affectation
- Tant que $i > 1$ faire : — 1 opération par tour de boucle + 1 opération supplémentaire
  - $res = res * i$ — 2 opérations : affectation et multiplication
  - $i = i - 1$ — 2 opérations (idem)
- Retourner res — 1 opération

Au total on a :

$$\begin{aligned} T(n) &= 2 + 5(n-1) + 1 + 1 \\ &= 2 + 5n - 5 + 1 + 1 \\ &= 5n - 1 \end{aligned}$$

Quelle complexité ?

Cela dépend de l'encodage !

si n est arbitrairement grand, encodage avec $\lceil \log_2(n) \rceil$ bits → Exponentielle car

$$5n - 1 = 5 \times 2^{\log_2(n)} - 1$$

Un exemple simple : calcul de la factorielle

factorielle(n) **:**

- Entrée : un entier n
- Sortie : $n!$

- $i = n$ — 1 opération : affectation
- $res = 1$ — 1 opération : affectation
- Tant que $i > 1$ faire : — 1 opération par tour de boucle + 1 opération supplémentaire
  - $res = res * i$ — 2 opérations : affectation et multiplication
  - $i = i - 1$ — 2 opérations (idem)
- Retourner res — 1 opération

Au total on a :

$$\begin{aligned} T(n) &= 2 + 5(n-1) + 1 + 1 \\ &= 2 + 5n - 5 + 1 + 1 \\ &= 5n - 1 \end{aligned}$$

Quelle complexité ?

Cela dépend de l'encodage !

si n est arbitrairement grand, encodage avec $\lceil \log_2(n) \rceil$ bits → Exponentielle car

$$5n - 1 = 5 \times 2^{\log_2(n)} - 1$$

si n est codé sur 4 ou 8 octets : complexité “constante”, puisque $n < 2^{63}$

→ le problème n'a alors pas beaucoup de sens...

Complexités typiques : des exemples concrets

- Recherche dichotomique dans un ensemble trié de taille n : $\log_2(n)$ (logarithmique)

Complexités typiques : des exemples concrets

- Recherche dichotomique dans un ensemble trié de taille n : $\log_2(n)$ (logarithmique)
- Parcourir un ensemble de taille n non trié : n (linéaire)

Complexités typiques : des exemples concrets

- Recherche dichotomique dans un ensemble trié de taille n : $\log_2(n)$ (logarithmique)
- Parcourir un ensemble de taille n non trié : n (linéaire)
- Trier un tableau d'entiers, chemin le plus court, etc. : n^c (polynômial)

Complexités typiques : des exemples concrets

- Recherche dichotomique dans un ensemble trié de taille n : $\log_2(n)$ (logarithmique)
- Parcourir un ensemble de taille n non trié : n (linéaire)
- Trier un tableau d'entiers, chemin le plus court, etc. : n^c (polynômial)
- k boucles imbriquées de longueur n chacune : n^k

Complexités typiques : des exemples concrets

- Recherche dichotomique dans un ensemble trié de taille n : $\log_2(n)$ (logarithmique)
- Parcourir un ensemble de taille n non trié : n (linéaire)
- Trier un tableau d'entiers, chemin le plus court, etc. : n^c (polynômial)
- k boucles imbriquées de longueur n chacune : n^k
- Parcourir les sous-ensembles d'un ensemble de taille n : 2^n (exponentiel simple)

Complexités typiques : des exemples concrets

- Recherche dichotomique dans un ensemble trié de taille n : $\log_2(n)$ (logarithmique)
- Parcourir un ensemble de taille n non trié : n (linéaire)
- Trier un tableau d'entiers, chemin le plus court, etc. : n^c (polynômial)
- k boucles imbriquées de longueur n chacune : n^k
- Parcourir les sous-ensembles d'un ensemble de taille n : 2^n (exponentiel simple)
- Énumérer toutes les partitions d'un ensemble de taille n : n^n ($= 2^{n \log_2(n)}$, super-exponentiel)

Complexités typiques : des exemples concrets

- Recherche dichotomique dans un ensemble trié de taille n : $\log_2(n)$ (logarithmique)
- Parcourir un ensemble de taille n non trié : n (linéaire)
- Trier un tableau d'entiers, chemin le plus court, etc. : n^c (polynômial)
- k boucles imbriquées de longueur n chacune : n^k
- Parcourir les sous-ensembles d'un ensemble de taille n : 2^n (exponentiel simple)
- Énumérer toutes les partitions d'un ensemble de taille n : n^n ($= 2^{n\log_2(n)}$, super-exponentiel)
- Énumérer toutes les permutations d'un ensemble de taille n : $n!$ $\approx n^n$ par l'*approximation de Stirling* $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

Complexités typiques : des exemples concrets

- Recherche dichotomique dans un ensemble trié de taille n : $\log_2(n)$ (logarithmique)
- Parcourir un ensemble de taille n non trié : n (linéaire)
- Trier un tableau d'entiers, chemin le plus court, etc. : n^c (polynômial)
- k boucles imbriquées de longueur n chacune : n^k
- Parcourir les sous-ensembles d'un ensemble de taille n : 2^n (exponentiel simple)
- Énumérer toutes les partitions d'un ensemble de taille n : n^n $(= 2^{n \log_2(n)}$, super-exponentiel)
- Énumérer toutes les permutations d'un ensemble de taille n : $n!$ $\approx n^n$ par l'*approximation de Stirling* $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$
- Énumérer les sous-ensembles de sous-ensembles : 2^{2^n} (doublement exponentiel)

Complexités typiques : des exemples concrets

- Recherche dichotomique dans un ensemble trié de taille n : $\log_2(n)$
  (logarithmique)
- Parcourir un ensemble de taille n non trié : n (linéaire)
- Trier un tableau d'entiers, chemin le plus court, etc. : n^c (polynômial)
- k boucles imbriquées de longueur n chacune : n^k
- Parcourir les sous-ensembles d'un ensemble de taille n : 2^n
  (exponentiel simple)
- Énumérer toutes les partitions d'un ensemble de taille n : n^n
  ($= 2^{n \log_2(n)}$, super-exponentiel)
- Énumérer toutes les permutations d'un ensemble de taille n : $n!$
  $\approx n^n$ par l'*approximation de Stirling* $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$
- Énumérer les sous-ensembles de sous-ensembles : 2^{2^n}
  (doublement exponentiel)
- ...

Faut-il vraiment être si précis ?

Le plus souvent, on souhaite simplement distinguer les types de complexité
→ logarithmique, linéaire, quadratique, exponentiel...

Faut-il vraiment être si précis ?

Le plus souvent, on souhaite simplement distinguer les types de complexité
→ logarithmique, linéaire, quadratique, exponentiel...

La complexité *exacte* dépend de toute façon de la machine, du langage de programmation, du compilateur...

Faut-il vraiment être si précis ?

Le plus souvent, on souhaite simplement distinguer les types de complexité
→ logarithmique, linéaire, quadratique, exponentiel...

La complexité *exacte* dépend de toute façon de la machine, du langage de programmation, du compilateur...

On utilise pour cela des notations asymptotiques qui omettent les facteurs constants et les “cas de base pathologiques”.

Notations asymptotiques : grand Oh

Inventées de 1894 à 1960 par Bachmann, Hardy, Knuth, Landau, Littlewood…

Définition (Grand Oh)

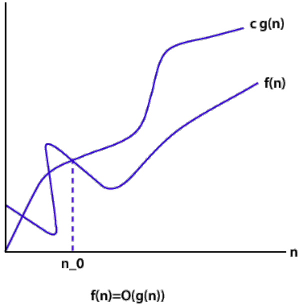
Soient deux fonctions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in O(g(n))$ si il existe une constante $c > 0 \in \mathbb{R}$ et un rang $n_0 \in \mathbb{N}$ tels que, pour tout entier $i \geq n_0$, on a $f(i) \leq c \cdot g(i)$.

Notations asymptotiques : grand Oh

Inventées de 1894 à 1960 par Bachmann, Hardy, Knuth, Landau, Littlewood...

Définition (Grand Oh)

Soient deux fonctions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in O(g(n))$ si il existe une constante $c > 0 \in \mathbb{R}$ et un rang $n_0 \in \mathbb{N}$ tels que, pour tout entier $i \geq n_0$, on a $f(i) \leq c \cdot g(i)$.

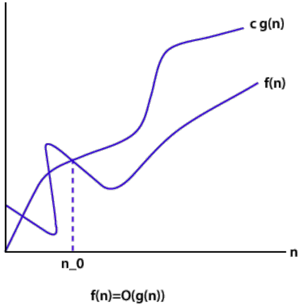


Notations asymptotiques : grand Oh

Inventées de 1894 à 1960 par Bachmann, Hardy, Knuth, Landau, Littlewood...

Définition (Grand Oh)

Soient deux fonctions $f, g : \mathbb{N} \rightarrow \mathbb{N}$. $f(n) \in O(g(n))$ si il existe une constante $c > 0 \in \mathbb{R}$ et un rang $n_0 \in \mathbb{N}$ tels que, pour tout entier $i \geq n_0$, on a $f(i) \leq c \cdot g(i)$.



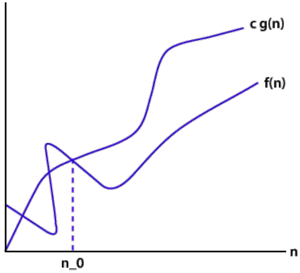
Intuitivement : *f ne grandit pas plus vite que g (à facteur constant près) lorsque n est suffisamment grand*

Notations asymptotiques : grand Oh

Inventées de 1894 à 1960 par Bachmann, Hardy, Knuth, Landau, Littlewood...

Définition (Grand Oh)

Soient deux fonctions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in O(g(n))$ si il existe une constante $c > 0 \in \mathbb{R}$ et un rang $n_0 \in \mathbb{N}$ tels que, pour tout entier $i \geq n_0$, on a $f(i) \leq c \cdot g(i)$.



f(n)=O(g(n))

Exemples :

- $2^{1000} \in O(1)$
- $100n \in O(n/1000)$
- $1000n \in O(n^2/10)$
- $n + n^2 + \log(n) \in O(n^3)$

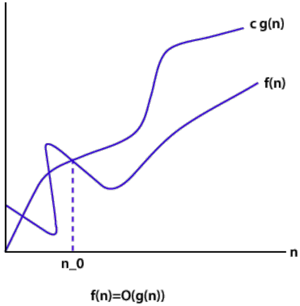
Intuitivement : *f ne grandit pas plus vite que g (à facteur constant près) lorsque n est suffisamment grand*

Notations asymptotiques : grand Oh

Inventées de 1894 à 1960 par Bachmann, Hardy, Knuth, Landau, Littlewood...

Définition (Grand Oh)

Soient deux fonctions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in O(g(n))$ si il existe une constante $c > 0 \in \mathbb{R}$ et un rang $n_0 \in \mathbb{N}$ tels que, pour tout entier $i \geq n_0$, on a $f(i) \leq c \cdot g(i)$.



Exemples :

- $2^{1000} \in O(1)$
- $100n \in O(n/1000)$
- $1000n \in O(n^2/10)$
- $n + n^2 + \log(n) \in O(n^3)$

Intuitivement : *f ne grandit pas plus vite que g (à facteur constant près) lorsque n est suffisamment grand*

Abus de notation : $10n = O(n^2)$

Notations asymptotiques : grand Omega

“La réciproque du Grand Oh”

Définition (Grand Omega)

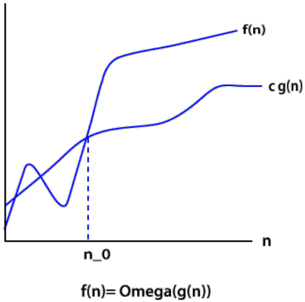
Soient deux fonctions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in \Omega(g(n))$ si il existe une constante $c > 0 \in \mathbb{R}$ et un rang $n_0 \in \mathbb{N}$ tels que, pour tout entier $i \geq n_0$, on a $f(i) \geq c \cdot g(i)$.

Notations asymptotiques : grand Omega

"La réciproque du Grand Oh"

Définition (Grand Omega)

Soient deux fonctions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in \Omega(g(n))$ si il existe une constante $c > 0 \in \mathbb{R}$ et un rang $n_0 \in \mathbb{N}$ tels que, pour tout entier $i \geq n_0$, on a $f(i) \geq c \cdot g(i)$.

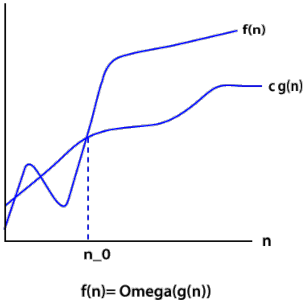


Notations asymptotiques : grand Omega

“La réciproque du Grand Oh”

Définition (Grand Omega)

Soient deux fonctions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in \Omega(g(n))$ si il existe une constante $c > 0 \in \mathbb{R}$ et un rang $n_0 \in \mathbb{N}$ tels que, pour tout entier $i \geq n_0$, on a $f(i) \geq c \cdot g(i)$.



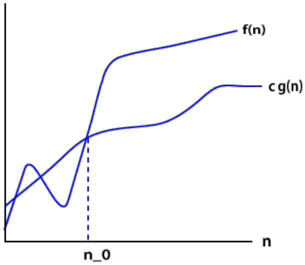
Intuitivement : *f ne grandit pas moins vite que g (à facteur constant près) lorsque n est suffisamment grand*

Notations asymptotiques : grand Omega

"La réciproque du Grand Oh"

Définition (Grand Omega)

Soient deux fonctions $f, g : \mathbb{N} \rightarrow \mathbb{N}$. $f(n) \in \Omega(g(n))$ si il existe une constante $c > 0 \in \mathbb{R}$ et un rang $n_0 \in \mathbb{N}$ tels que, pour tout entier $i \geq n_0$, on a $f(i) \geq c \cdot g(i)$.



f(n)= Omega(g(n))

Exemples :

- $2^{1000} \in \Omega(1)$
- $n^2/1000 \in \Omega(100000n^2)$
- $n \log(n)/10 \in \Omega(100n)$
- $n^3 - n^2 \in \Omega(n^2)$

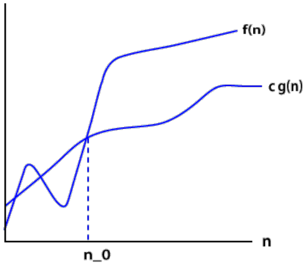
Intuitivement : *f ne grandit pas moins vite que g (à facteur constant près) lorsque n est suffisamment grand*

Notations asymptotiques : grand Omega

"La réciproque du Grand Oh"

Définition (Grand Omega)

Soient deux fonctions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in \Omega(g(n))$ si il existe une constante $c > 0 \in \mathbb{R}$ et un rang $n_0 \in \mathbb{N}$ tels que, pour tout entier $i \geq n_0$, on a $f(i) \geq c \cdot g(i)$.



f(n)= Omega(g(n))

Exemples :

- $2^{1000} \in \Omega(1)$
- $n^2/1000 \in \Omega(100000n^2)$
- $n \log(n)/10 \in \Omega(100n)$
- $n^3 - n^2 \in \Omega(n^2)$

Intuitivement : *f ne grandit pas moins vite que g (à facteur constant près) lorsque n est suffisamment grand*

Remarque : Si $f(n) \in O(g(n))$, alors $g(n) \in \Omega(f(n))$ et inversement

Notations asymptotiques : Theta

"La combinaison du grand Oh et du grand Omega"

Définition (Grand Theta)

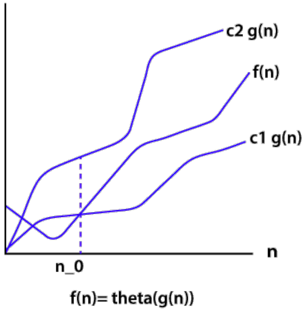
Soient deux fonctions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in \Theta(g(n))$ si $f(n) \in O(g(n))$ et $f(n) \in \Omega(g(n))$.

Notations asymptotiques : Theta

"La combinaison du grand Oh et du grand Omega"

Définition (Grand Theta)

Soient deux fonctions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in \Theta(g(n))$ si $f(n) \in O(g(n))$ et $f(n) \in \Omega(g(n))$.

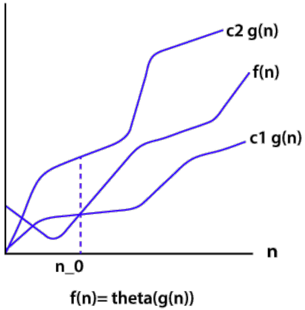


Notations asymptotiques : Theta

"La combinaison du grand Oh et du grand Omega"

Définition (Grand Theta)

Soient deux fonctions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in \Theta(g(n))$ si $f(n) \in O(g(n))$ et $f(n) \in \Omega(g(n))$.



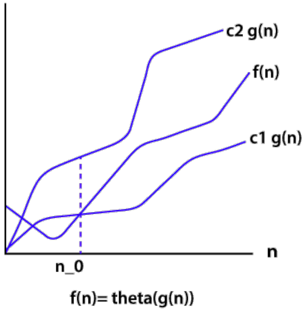
Intuitivement : *f et g grandissent aussi vite l'une que l'autre (à facteur constant près) lorsque n est suffisamment grand*

Notations asymptotiques : Theta

"La combinaison du grand Oh et du grand Omega"

Définition (Grand Theta)

Soient deux fonctions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in \Theta(g(n))$ si $f(n) \in O(g(n))$ et $f(n) \in \Omega(g(n))$.



Exemples :

- $2^{1000} \in \Theta(1)$
- $1000n \in \Theta(n/1000)$
- $10n^2 + 36n \in \Theta(n^2)$
- $n^3 - n^2 \in \Theta(n^3)$

Intuitivement : *f et g grandissent aussi vite l'une que l'autre (à facteur constant près) lorsque n est suffisamment grand*

Classes de complexité

- Complexité logarithmique : $\Theta(\log(n))$
- Complexité linéaire : $\Theta(n)$
- Complexité quadratique : $\Theta(n^2)$
- Complexité cubique : $\Theta(n^3)$
- Complexité polynômiale : $\Theta(n^c)$ pour $c > 1$
- Complexité exponentielle : $\Theta(c^n)$ pour $c > 1$
- ...