# Analyse Asymptotique

# Complexité Algorithmique

# Complexité Algorithmique

- Vision pessimiste : la **complexité** d'un algortihme est souvent définie comme sa performance **asymptotique** dans le **pire cas**

# Complexité Algorithmique

- Vision pessimiste : la **complexité** d'un algortihme est souvent définie comme sa performance **asymptotique** dans le **pire cas**
- Que signifie **dans le pire des cas** ?

# Complexité Algorithmique

- Vision pessimiste : la **complexité** d'un algortihme est souvent définie comme sa performance **asymptotique** dans le **pire cas**
- Que signifie **dans le pire des cas** ?
  - Parmi toutes les données $x$ de taille $n$, on ne considère que celle qui maximise $\mathrm{Coût}_A(x)$

# Complexité Algorithmique

- Vision pessimiste : la **complexité** d'un algortihme est souvent définie comme sa performance **asymptotique** dans le **pire cas**
- Que signifie **dans le pire des cas** ?
  - Parmi toutes les données $x$ de taille $n$, on ne considère que celle qui maximise $\mathrm{Coût}_A(x)$
- Que signifie **asymptotique** ?

# Complexité Algorithmique

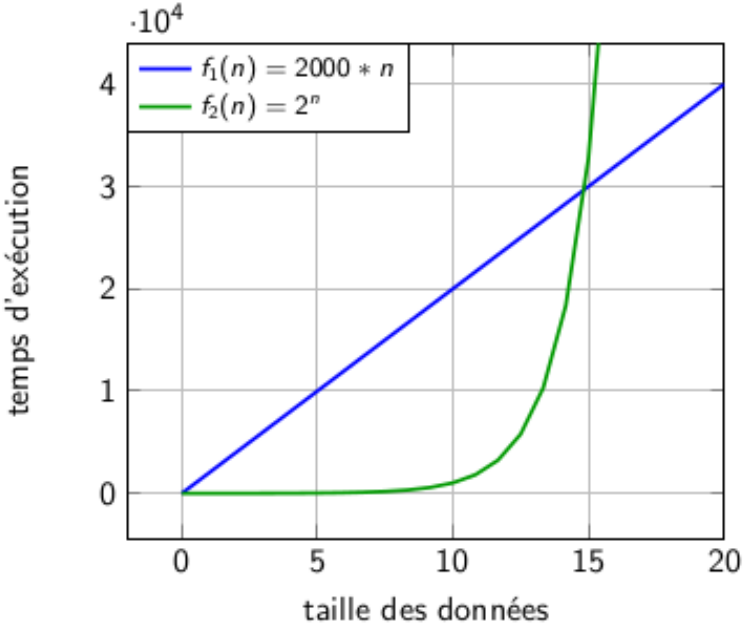
- Vision pessimiste : la **complexité** d'un algortihme est souvent définie comme sa performance **asymptotique** dans le **pire cas**
- Que signifie **dans le pire des cas** ?
  - Parmi toutes les données $x$ de taille $n$, on ne considère que celle qui maximise $\mathrm{Coût}_A(x)$
- Que signifie **asymptotique** ?
  - comportement de l'algorithme pour des données de taille $n$ *arbitrairement grande*

# Complexité Algorithmique

- Vision pessimiste : la **complexité** d'un algortihme est souvent définie comme sa performance **asymptotique** dans le **pire cas**
- Que signifie **dans le pire des cas** ?
  - Parmi toutes les données $x$ de taille $n$, on ne considère que celle qui maximise $\mathrm{Coût}_A(x)$
- Que signifie **asymptotique** ?
  - comportement de l'algorithme pour des données de taille $n$ *arbitrairement grande*
  - pourquoi ?

# Complexité Algorithmique

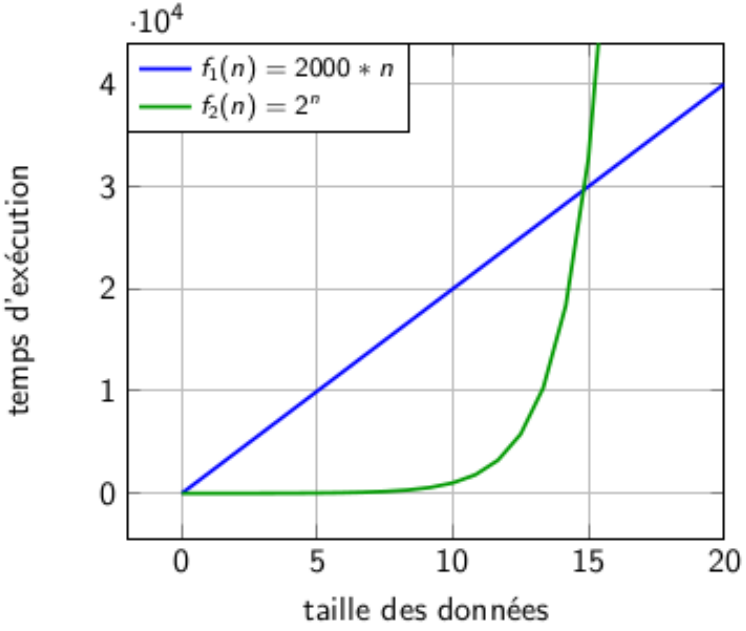
- Vision pessimiste : la **complexité** d'un algortihme est souvent définie comme sa performance **asymptotique** dans le **pire cas**
- Que signifie **dans le pire des cas** ?
  - Parmi toutes les données $x$ de taille $n$, on ne considère que celle qui maximise $\mathrm{Coût}_A(x)$
- Que signifie **asymptotique** ?
  - comportement de l'algorithme pour des données de taille $n$ *arbitrairement grande*
  - pourquoi ?




- Soit deux algorithmes de complexités $f_1(n)$ et $f_2(n)$

# Complexité Algorithmique

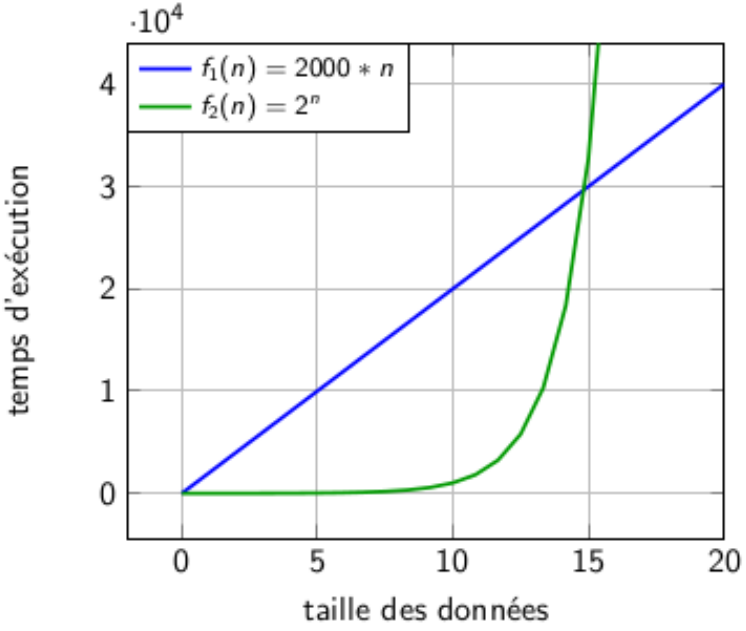
- Vision pessimiste : la **complexité** d'un algortihme est souvent définie comme sa performance **asymptotique** dans le **pire cas**
- Que signifie **dans le pire des cas** ?
  - Parmi toutes les données $x$ de taille $n$, on ne considère que celle qui maximise $\mathrm{Coût}_A(x)$
- Que signifie **asymptotique** ?
  - comportement de l'algorithme pour des données de taille $n$ *arbitrairement grande*
  - pourquoi ?




- Soit deux algorithmes de complexités $f_1(n)$ et $f_2(n)$
- Quel algorithme préférez-vous ?

# Complexité Algorithmique

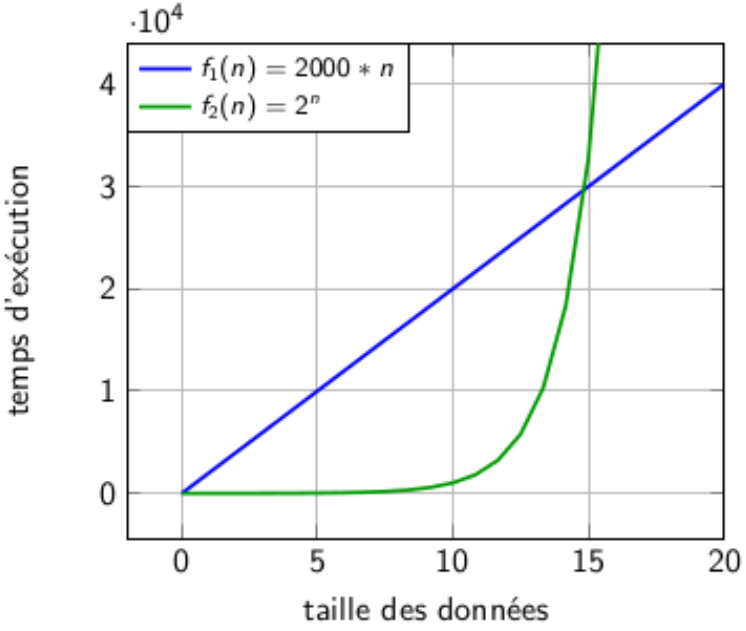
- Vision pessimiste : la **complexité** d'un algortihme est souvent définie comme sa performance **asymptotique** dans le **pire cas**
- Que signifie **dans le pire des cas** ?
  - Parmi toutes les données $x$ de taille $n$, on ne considère que celle qui maximise $\mathrm{Coût}_A(x)$
- Que signifie **asymptotique** ?
  - comportement de l'algorithme pour des données de taille $n$ *arbitrairement grande*
  - pourquoi ?




- Soit deux algorithmes de complexités $f_1(n)$ et $f_2(n)$
- Quel algorithme préférez-vous ?
- La courbe verte semble correspondre à un algorithme plus efficace...

# Complexité Algorithmique

- Vision pessimiste : la **complexité** d'un algortihme est souvent définie comme sa performance **asymptotique** dans le **pire cas**
- Que signifie **dans le pire des cas** ?
  - Parmi toutes les données $x$ de taille $n$, on ne considère que celle qui maximise $\mathrm{Coût}_A(x)$
- Que signifie **asymptotique** ?
  - comportement de l'algorithme pour des données de taille $n$ *arbitrairement grande*
  - pourquoi ?




- Soit deux algorithmes de complexités $f_1(n)$ et $f_2(n)$
- Quel algorithme préférez-vous ?
- La courbe verte semble correspondre à un algorithme plus efficace...
- ... mais seulement pour de très petites valeurs !

# Ordre de grandeur : motivation

- Les calculs à effectuer pour évaluer le temps d'exécution d'un algorithme peuvent parfois être longs et pénibles ;

# Ordre de grandeur : motivation

- Les calculs à effectuer pour évaluer le temps d'exécution d'un algorithme peuvent parfois être longs et pénibles ;
- De plus, le degré de précision qu'ils requièrent est souvent inutile ;

# Ordre de grandeur : motivation

- Les calculs à effectuer pour évaluer le temps d'exécution d'un algorithme peuvent parfois être longs et pénibles ;
- De plus, le degré de précision qu'ils requièrent est souvent inutile ;
  - $n \log n + 5n \rightarrow 5n$ va devenir "négligeable" ($n >> 1000$)

# Ordre de grandeur : motivation

- Les calculs à effectuer pour évaluer le temps d'exécution d'un algorithme peuvent parfois être longs et pénibles ;
- De plus, le degré de précision qu'ils requièrent est souvent inutile ;
  - $n \log n + 5n \rightarrow 5n$ va devenir "négligeable" ($n >> 1000$)
  - différence entre un algorithme en $10n^3$ et $9n^3$ : effacé par une accélération de $\frac{10}{9}$ de la machine

# Ordre de grandeur : motivation

- Les calculs à effectuer pour évaluer le temps d'exécution d'un algorithme peuvent parfois être longs et pénibles ;
- De plus, le degré de précision qu'ils requièrent est souvent inutile ;
  - $n \log n + 5n \rightarrow 5n$ va devenir "négligeable" ($n >> 1000$)
  - différence entre un algorithme en $10n^3$ et $9n^3$ : effacé par une accélération de $\frac{10}{9}$ de la machine
- On aura donc recours à une **approximation** de ce temps de calcul, représentée par les notations $\mathcal{O}, \Omega$ et $\Theta$

# Ordre de grandeur : motivation

- Les calculs à effectuer pour évaluer le temps d'exécution d'un algorithme peuvent parfois être longs et pénibles ;
- De plus, le degré de précision qu'ils requièrent est souvent inutile ;
  - $n \log n + 5n \rightarrow 5n$ va devenir "négligeable" ($n >> 1000$)
  - différence entre un algorithme en $10n^3$ et $9n^3$ : effacé par une accélération de $\frac{10}{9}$ de la machine
- On aura donc recours à une **approximation** de ce temps de calcul, représentée par les notations $\mathcal{O}, \Omega$ et $\Theta$

**Hypothèse simplificatrice**

On ne s'intéresse qu'aux fonctions **asymptotiquement positives** (positives pour tout $n \geq n_0$)

## Notation $\mathcal{O}$ : définition

**$\mathcal{O}(g(n))$ est l'ensemble de fonctions $f(n)$ telles que :**

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

## Notation $\mathcal{O}$ : définition

**$\mathcal{O}(g(n))$ est l'ensemble de fonctions $f(n)$ telles que :**

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

**Borne supérieure :** $f(n) \in \mathcal{O}(g(n))$ **s'il existe une constante** $c$**, et un seuil à partir duquel** $f(n)$ est inférieure à $g(n)$, à un facteur $c$ près ;

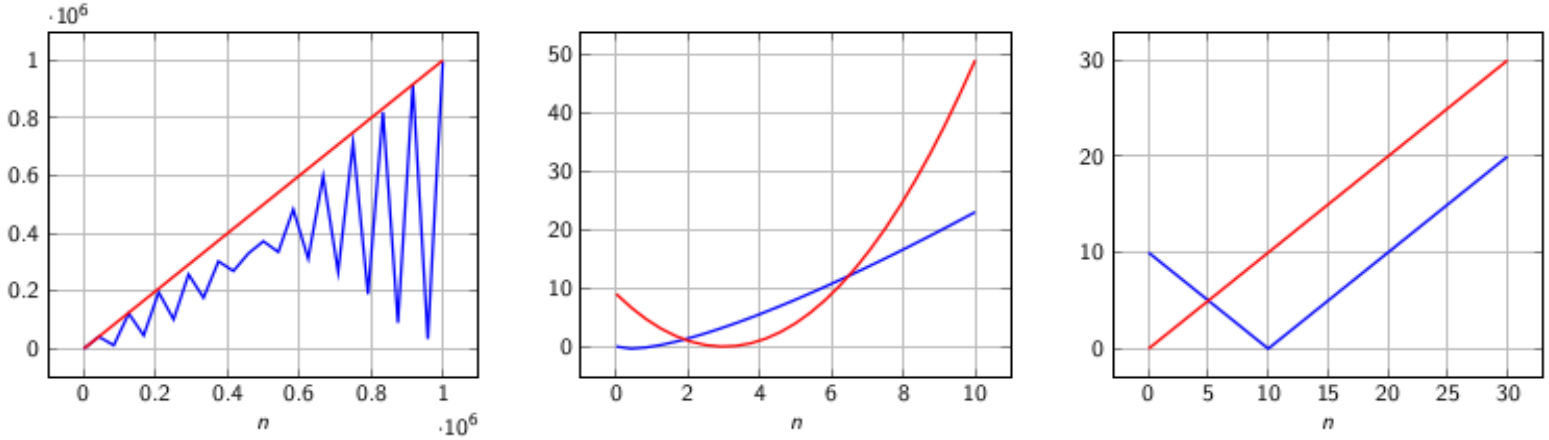
# Notation $\mathcal{O}$ : définition

**$\mathcal{O}(g(n))$ est l'ensemble de fonctions $f(n)$ telles que :**

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

**Borne supérieure :** $f(n) \in \mathcal{O}(g(n))$ **s'il existe une constante** $c$**, et un seuil à partir duquel** $f(n)$ est inférieure à $g(n)$, à un facteur $c$ près ;

**Exemple :** $f(n) \in \mathcal{O}(g(n))$



# Notation $\mathcal{O}$ : preuve

**$\mathcal{O}(g(n))$ est l'ensemble de fonctions $f(n)$ telles que :**

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

## Notation $\mathcal{O}$ : preuve

**$\mathcal{O}(g(n))$ est l'ensemble de fonctions $f(n)$ telles que :**

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

Prouver que $f(n) \in \mathcal{O}(g(n))$ : jeux contre un perfide adversaire $\forall$

## Notation $\mathcal{O}$ : preuve

**$\mathcal{O}(g(n))$ est l'ensemble de fonctions $f(n)$ telles que :**

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

Prouver que $f(n) \in \mathcal{O}(g(n))$ : jeux contre un perfide adversaire $\forall$

Tour du joueur $\exists$ objectif : $f(n) \leq cg(n)$ choisit $c$ et $n_0$

## Notation $\mathcal{O}$ : preuve

**$\mathcal{O}(g(n))$ est l'ensemble de fonctions $f(n)$ telles que :**

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

Prouver que $f(n) \in \mathcal{O}(g(n))$ : jeux contre un perfide adversaire $\forall$

| | | |
|---|---|---|
| Tour du joueur $\exists$ | objectif : $f(n) \leq cg(n)$ | choisit $c$ et $n_0$ |
| Tour du joueur $\forall$ | objectif : $f(n) > cg(n)$ | choisit $n \geq n_0$ |

## Notation $\mathcal{O}$ : preuve

**$\mathcal{O}(g(n))$ est l'ensemble de fonctions $f(n)$ telles que :**

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

Prouver que $f(n) \in \mathcal{O}(g(n))$ : jeux contre un perfide adversaire $\forall$

Tour du joueur $\exists$ objectif : $f(n) \leq cg(n)$ choisit $c$ et $n_0$

Tour du joueur $\forall$ objectif : $f(n) > cg(n)$ choisit $n \geq n_0$

Arbitre détermine le gagnant : $f(n) \leq cg(n)$

## Notation $\mathcal{O}$ : exemple

**$\mathcal{O}(g(n))$ est l'ensemble de fonctions $f(n)$ telles que :**

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

Jeux : **prouver** que la fonction $f_2(n) = 6n^2 + 2n - 8$ est en $\mathcal{O}(n^2)$ :

## Notation $\mathcal{O}$ : exemple

**$\mathcal{O}(g(n))$ est l'ensemble de fonctions $f(n)$ telles que :**

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

Jeux : **prouver** que la fonction $f_2(n) = 6n^2 + 2n - 8$ est en $\mathcal{O}(n^2)$ :

Tour du joueur $\exists$ choisit $c = 6$ et $n_0 = 0$

## Notation $\mathcal{O}$ : exemple

**$\mathcal{O}(g(n))$ est l'ensemble de fonctions $f(n)$ telles que :**

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

Jeux : **prouver** que la fonction $f_2(n) = 6n^2 + 2n - 8$ est en $\mathcal{O}(n^2)$ :

Tour du joueur $\exists$ choisit $c = 6$ et $n_0 = 0$

Tour du joueur $\forall$ choisit $n = 5$

## Notation $\mathcal{O}$ : exemple

**$\mathcal{O}(g(n))$ est l'ensemble de fonctions $f(n)$ telles que :**

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

Jeux : **prouver** que la fonction $f_2(n) = 6n^2 + 2n - 8$ est en $\mathcal{O}(n^2)$ :

| | |
|---|---|
| Tour du joueur $\exists$ | choisit $c = 6$ et $n_0 = 0$ |
| Tour du joueur $\forall$ | choisit $n = 5$ |
| Arbitre | $6 \times 5^2 + 2 \times 5 - 8 > 6 \times 5^2$ |

## Notation $\mathcal{O}$ : exemple

**$\mathcal{O}(g(n))$ est l'ensemble de fonctions $f(n)$ telles que :**

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

Jeux : **prouver** que la fonction $f_2(n) = 6n^2 + 2n - 8$ est en $\mathcal{O}(n^2)$ :

Tour du joueur $\exists$ choisit $c = 7$ et $n_0 = 0$

## Notation $\mathcal{O}$ : exemple

**$\mathcal{O}(g(n))$ est l'ensemble de fonctions $f(n)$ telles que :**

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

Jeux : **prouver** que la fonction $f_2(n) = 6n^2 + 2n - 8$ est en $\mathcal{O}(n^2)$ :

Tour du joueur $\exists$ choisit $c = 7$ et $n_0 = 0$

Tour du joueur $\forall$ choisit ?

## Notation $\mathcal{O}$ : exemple

**$\mathcal{O}(g(n))$ est l'ensemble de fonctions $f(n)$ telles que :**

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

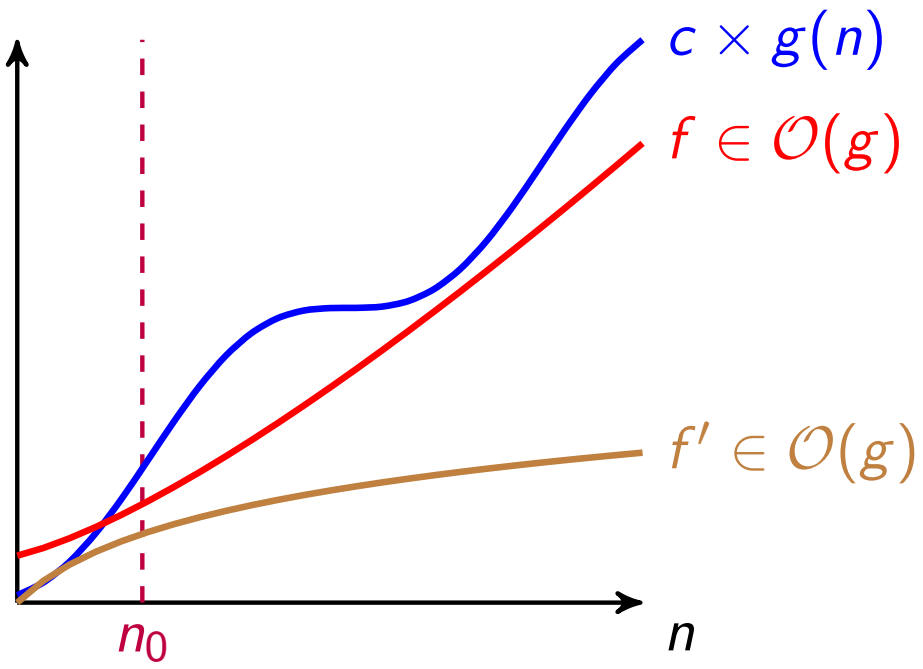
Jeux : **prouver** que la fonction $f_2(n) = 6n^2 + 2n - 8$ est en $\mathcal{O}(n^2)$ :

| | |
|---|---|
| Tour du joueur $\exists$ | choisit $c = 7$ et $n_0 = 0$ |
| Tour du joueur $\forall$ | choisit ? |
| Arbitre | $n^2 - 2n + 8 < 0$ n'a pas de solution |

## Focus sur $\mathcal{O}$

**$\mathcal{O}(g(n))$ est l'ensemble de fonctions $f(n)$ telles que :**

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




Exercice : $2n^2$ est-il en $\mathcal{O}(n^2)$ ? Pareil pour $2n$.

# Notation $\Omega$ : définition

**$\Omega(g(n))$ est l'ensemble de fonctions $f(n)$ telles que :**

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

## Notation $\Omega$ : définition

**$\Omega(g(n))$ est l'ensemble de fonctions $f(n)$ telles que :**

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

**Borne inférieure :** $f(n) \in \Omega(g(n))$ s'il existe un seuil à partir duquel $f(n)$ est supérieure à $g(n)$, à une constante multiplicative près ;

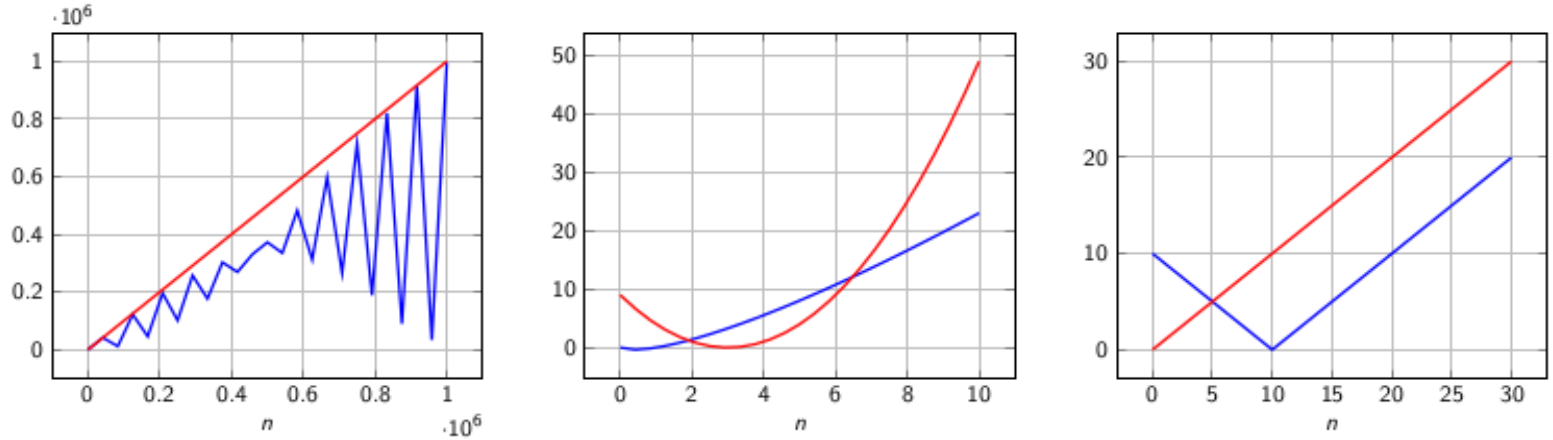
# Notation $\Omega$ : définition

**$\Omega(g(n))$ est l'ensemble de fonctions $f(n)$ telles que :**

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

**Borne inférieure :** $f(n) \in \Omega(g(n))$ s'il existe un seuil à partir duquel $f(n)$ est supérieure à $g(n)$, à une constante multiplicative près ;

**Exemple :** $g(n) \in \Omega(f(n))$



# Notation $\Theta$ : définition

**$\Theta(g(n))$ est l'ensemble de fonctions $f(n)$ telles que :**

$$\exists c_1, c_2 \in \mathbb{R}^{+*}, \exists n_0 \in \mathbb{N}, \forall n > n_0, c_1 \times g(n) \leq f(n) \leq c_2 \times g(n)$$

## Notation $\Theta$ : définition

**$\Theta(g(n))$ est l'ensemble de fonctions $f(n)$ telles que :**

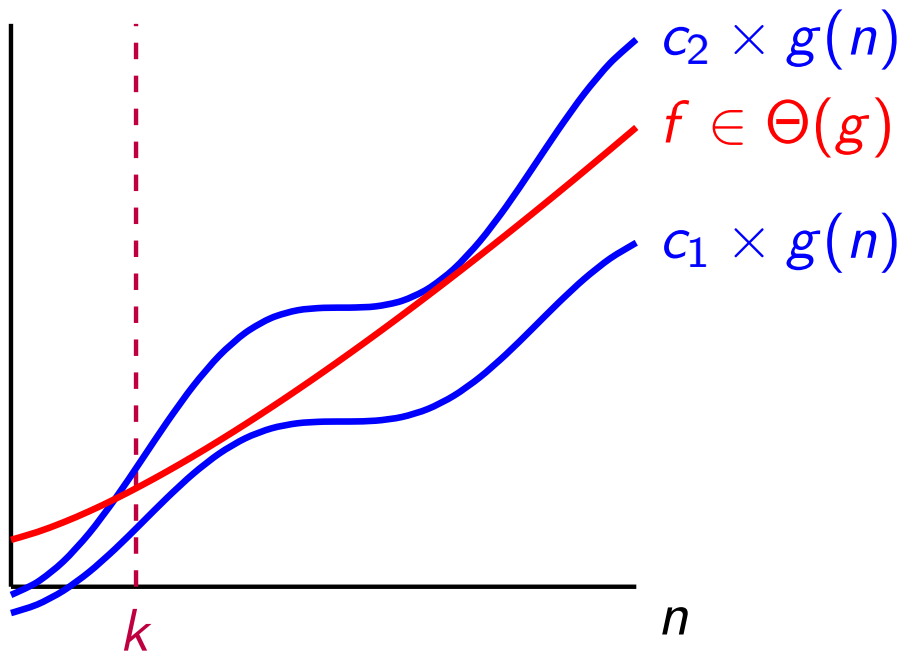
$$\exists c_1, c_2 \in \mathbb{R}^{+*}, \exists n_0 \in \mathbb{N}, \forall n > n_0, c_1 \times g(n) \leq f(n) \leq c_2 \times g(n)$$

**Borne supérieure et inférieure :** $\Theta(g(n)) = \Omega(g(n)) \cap \mathcal{O}(g(n))$ ; $f(n)$ est en $\Theta(g(n))$ si elle est **prise en sandwich** entre $c_1 g(n)$ et $c_2 g(n)$ ;

## Focus sur $\Theta$

**$f(n)$ est en $\Theta(g(n))$ si :**

$$\exists c_1, c_2 \in \mathbb{R}^{+*}, \exists n_0 \in \mathbb{N}, \forall n > n_0, \qquad c_1 \times g(n) \leq f(n) \leq c_2 \times g(n)$$




Exercice : $2n^2$ est-il en $\Theta(n^2)$ ? Pareil pour $2n$.

# Notation asymptotique d'une fonction

- Quelle est la borne asymptotique de $f(n)$ ?

# Notation asymptotique d'une fonction

- Quelle est la borne asymptotique de $f(n)$ ?

## Notation asymptotique (de l'expression fermée) d'une fonction

Les mêmes simplifications pour $\mathcal{O}, \Omega$ et $\Theta$ :

- **on ne retient que les termes dominants**
- **on supprime les constantes multiplicatives**

# Notation asymptotique d'une fonction

- Quelle est la borne asymptotique de $f(n)$ ?

## Notation asymptotique (de l'expression fermée) d'une fonction

Les mêmes simplifications pour $\mathcal{O}, \Omega$ et $\Theta$ :

- **on ne retient que les termes dominants**
- **on supprime les constantes multiplicatives**

## Exemple

Soit $g(n) = 4n^3 - 5n^2 + 2n + 3$ ;

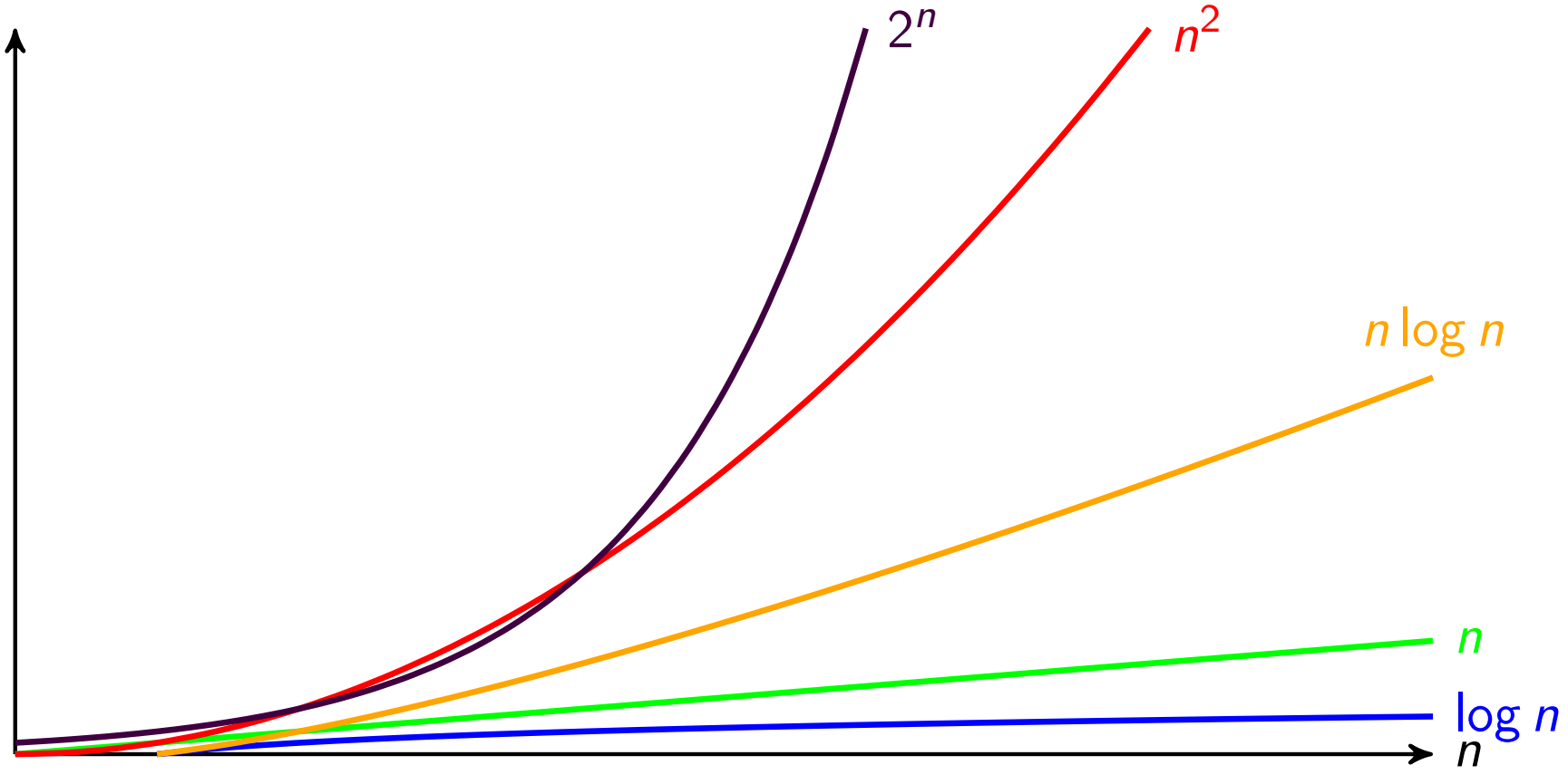
1. on ne retient que le terme de plus haut degré : $4n^3$ (pour $n$ assez grand le terme en $n^3$ "domine" les autres, en choisissant bien $c_1, c_2$, on peut avoir $c_1 n^3 \leq g(n) \leq c_2 n^3$)
2. on supprime les constantes multiplicatives : $n^3$ (on peut la choisir !)

et on a donc $g(n) \in \Theta(n^3)$

# Ordre de grandeur : exemples

- $5n^3 + 3n + 7 \in \mathcal{O}(n^3)$ ?
- $5n^3 + 3n + 7 \in \Theta(n^3)$ ?
- $5n^3 + 3n + 7 \in \mathcal{O}(n^4)$ ?
- $5n^3 + 3n + 7 \in \Theta(n^4)$ ?
- $\log n \in \mathcal{O}(n)$ ?
- $\log n \in \Theta(n)$ ?

# Relation des principaux ordres de grandeur




Indépendant de la taille de la donnée : $\mathcal{O}(1)/\Theta(1)$

# Vocabulaire

Un algorithme dont la donnée est de taille $|x| = n$ est dit :

- **Constant** si sa complexité est en $\mathcal{O}(1)$
- **Logarithmique** si sa complexité est en $\Theta(\log n)$
- **Linéaire** si sa complexité est en $\Theta(n)$
- **Quadratique** si sa complexité est en $\Theta(n^2)$
- **Polynomial** si sa complexité est en $\mathcal{O}(n^{\mathcal{O}(1)})$
- **Exponentiel** si sa complexité est en $\Theta(c^{\Theta(n)})$ pour une constante $c > 1$

# Quelques remarques (à prouver comme exercice)

# Quelques remarques (à prouver comme exercice)

- $f \in \mathcal{O}(g)$ ssi $g \in \Omega(f)$
- $f \in \Theta(g)$ ssi $g \in \Theta(f)$

# Quelques remarques (à prouver comme exercice)

- $f \in \mathcal{O}(g)$ ssi $g \in \Omega(f)$
- $f \in \Theta(g)$ ssi $g \in \Theta(f)$

- Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c > 0$ (constante) alors $f \in \Theta(g)$ (et donc $g \in \Theta(f)$)

# Quelques remarques (à prouver comme exercice)

- $f \in \mathcal{O}(g)$ ssi $g \in \Omega(f)$
- $f \in \Theta(g)$ ssi $g \in \Theta(f)$

- Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c > 0$ (constante) alors $f \in \Theta(g)$ (et donc $g \in \Theta(f)$)
- Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ alors $f \in \mathcal{O}(g)$ et $f \notin \Omega(g)$

# Quelques remarques (à prouver comme exercice)

- $f \in \mathcal{O}(g)$ ssi $g \in \Omega(f)$
- $f \in \Theta(g)$ ssi $g \in \Theta(f)$

- Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c > 0$ (constante) alors $f \in \Theta(g)$ (et donc $g \in \Theta(f)$)
- Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ alors $f \in \mathcal{O}(g)$ et $f \notin \Omega(g)$
- Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ alors $f \in \Omega(g)$ et $f \notin \mathcal{O}(g)$

# Quelques remarques (à prouver comme exercice)

- $f \in \mathcal{O}(g)$ ssi $g \in \Omega(f)$
- $f \in \Theta(g)$ ssi $g \in \Theta(f)$

- Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c > 0$ (constante) alors $f \in \Theta(g)$ (et donc $g \in \Theta(f)$)
- Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ alors $f \in \mathcal{O}(g)$ et $f \notin \Omega(g)$
- Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ alors $f \in \Omega(g)$ et $f \notin \mathcal{O}(g)$

## Règle de l'Hôpital

$f$ et $g$ deux fonctions dérivables t.q. $\lim\limits_{n\to\infty} f(n) = \lim\limits_{n\to\infty} g(n) = \infty$, alors :

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)} \text{ si cette limite existe.}$$

$f'$ (respectivement $g'$) représente la dévirée de $f$ (respectivement $g$)

# Règles de calculs : combinaisons des complexités

- Les instructions de base prennent un temps constant, noté $\mathcal{O}(1)$ ;

# Règles de calculs : combinaisons des complexités

- Les instructions de base prennent un temps constant, noté $\mathcal{O}(1)$ ;
- On additionne les complexités d'opérations en séquence :

$$\Theta(f(n)) + \Theta(g(n)) = \Theta(f(n) + g(n))$$

# Règles de calculs : combinaisons des complexités

- Les instructions de base prennent un temps constant, noté $\mathcal{O}(1)$ ;
- On additionne les complexités d'opérations en séquence :

$$\Theta(f(n)) + \Theta(g(n)) = \Theta(f(n) + g(n))$$

- Branchements conditionnels : max (analyse dans le *pire des cas*)

# Règles de calculs : combinaisons des complexités

- Les instructions de base prennent un temps constant, noté $\mathcal{O}(1)$ ;
- On additionne les complexités d'opérations en séquence :

$$\Theta(f(n)) + \Theta(g(n)) = \Theta(f(n) + g(n))$$

- Branchements conditionnels : max (analyse dans le *pire des cas*)
- L'ordre de grandeur maximum est égal à la somme des ordres de grandeur :

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

# Règles de calculs : combinaisons des complexités

- Les instructions de base prennent un temps constant, noté $\mathcal{O}(1)$ ;
- On additionne les complexités d'opérations en séquence :

$$\Theta(f(n)) + \Theta(g(n)) = \Theta(f(n) + g(n))$$

- Branchements conditionnels : max (analyse dans le *pire des cas*)
- L'ordre de grandeur maximum est égal à la somme des ordres de grandeur :

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

**Exemple**

**si** <*condition*> **alors**
  #instructions (1);
**sinon**
  #instructions (2);

# Règles de calculs : combinaisons des complexités

- Les instructions de base prennent un temps constant, noté $\mathcal{O}(1)$ ;
- On additionne les complexités d'opérations en séquence :

$$\Theta(f(n)) + \Theta(g(n)) = \Theta(f(n) + g(n))$$

- Branchements conditionnels : max (analyse dans le *pire des cas*)
- L'ordre de grandeur maximum est égal à la somme des ordres de grandeur :

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

**Exemple**

**si** <*condition*> **alors**
  #instructions (1);
**sinon**
  #instructions (2);

$\Theta(g(n))$

# Règles de calculs : combinaisons des complexités

- Les instructions de base prennent un temps constant, noté $\mathcal{O}(1)$ ;
- On additionne les complexités d'opérations en séquence :

$$\Theta(f(n)) + \Theta(g(n)) = \Theta(f(n) + g(n))$$

- Branchements conditionnels : max (analyse dans le *pire des cas*)
- L'ordre de grandeur maximum est égal à la somme des ordres de grandeur :

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

**Exemple**

| | |
|---|---|
| **si** <*condition*> **alors** | $\Theta(g(n))$ |
| #instructions (1); | $\Theta(f_1(n))$ |
| **sinon** | |
| #instructions (2); | |

# Règles de calculs : combinaisons des complexités

- Les instructions de base prennent un temps constant, noté $\mathcal{O}(1)$ ;
- On additionne les complexités d'opérations en séquence :

$$\Theta(f(n)) + \Theta(g(n)) = \Theta(f(n) + g(n))$$

- Branchements conditionnels : max (analyse dans le *pire des cas*)
- L'ordre de grandeur maximum est égal à la somme des ordres de grandeur :

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

**Exemple**

| | |
|---|---|
| **si** *<condition>* **alors** | $\Theta(g(n))$ |
| #instructions (1); | $\Theta(f_1(n))$ |
| **sinon** | |
| #instructions (2); | $\Theta(f_2(n))$ |

# Règles de calculs : combinaisons des complexités

- Les instructions de base prennent un temps constant, noté $\mathcal{O}(1)$ ;
- On additionne les complexités d'opérations en séquence :

$$\Theta(f(n)) + \Theta(g(n)) = \Theta(f(n) + g(n))$$

- Branchements conditionnels : max (analyse dans le *pire des cas*)
- L'ordre de grandeur maximum est égal à la somme des ordres de grandeur :

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

**Exemple**

**si** *<condition>* **alors**
  #instructions (1);
**sinon**
  #instructions (2);

$$\left.\begin{array}{l} \Theta(g(n)) \\ \Theta(f_1(n)) \\ \\ \Theta(f_2(n)) \end{array}\right\} = \Theta(g(n) + f_1(n) + f_2(n))$$

# Règles de calculs : combinaison des complexité

- Dans les boucles, on multiplie la complexité du corps de la boucle par le nombre d'itérations ;

# Règles de calculs : combinaison des complexité

- Dans les boucles, on multiplie la complexité du corps de la boucle par le nombre d'itérations ;

- Calcul de la complexité d'une boucle `while` :

## Exemple

*en supposant qu'on a $\Theta(h(n))$ itérations*

**tant que** *<condition>* **faire**
  #instructions ;

$$\left.\begin{array}{l}\Theta(g(n))\\ \Theta(f(n))\end{array}\right\} = \Theta(h(n) \times (g(n) + f(n)))$$

# Règles de calculs : combinaison des complexité

- Dans les boucles, on multiplie la complexité du corps de la boucle par le nombre d'itérations ;

- Calcul de la complexité d'une boucle `for` :

## Exemple

**pour** *i allant de a à b* **faire**
  #instructions ;

$$\left.\Theta(f(n))\right\} = \Theta((b-a+1) \times f(n))$$

# Calcul de la complexité asymptotique d'un algorithme

- Pour calculer la complexité d'un algorithme :

# Calcul de la complexité asymptotique d'un algorithme

- Pour calculer la complexité d'un algorithme :

1. on calcule la complexité de chaque "partie" de l'algorithme ;

# Calcul de la complexité asymptotique d'un algorithme

- Pour calculer la complexité d'un algorithme :

1. on calcule la complexité de chaque "partie" de l'algorithme ;
2. on combine ces complexités conformément aux règles qu'on vient de voir ;

# Calcul de la complexité asymptotique d'un algorithme

- Pour calculer la complexité d'un algorithme :

1. on calcule la complexité de chaque "partie" de l'algorithme ;
2. on combine ces complexités conformément aux règles qu'on vient de voir ;
3. on simplifie le résultat grâce aux règles de simplifications qu'on a vues ;
   - ★ élimination des constantes, et
   - ★ conservation du (des) termes dominants

# Exemple : calcul de la factorielle de $n \in \mathbb{N}$

- Reprenons le calcul de la factorielle, qui nécessitait $3n$ opérations :

**Algorithme :** Factorielle(n)

| | nombre | coût |
|---|---|---|
| **Données :** un entier $n$ | | |
| **Résultat :** un entier valant $n!$ | | |
| $fact, i$ : entier; | | |
| **début** | | |
| $fact \leftarrow 2$; | | |
| **pour** $i$ *allant de* 3 *à* $n$ **faire** | | |
| $fact \leftarrow fact * i$; | | |
| **retourner** $fact$; | | |

# Exemple : calcul de la factorielle de $n \in \mathbb{N}$

- Reprenons le calcul de la factorielle, qui nécessitait $3n$ opérations :

**Algorithme :** Factorielle(n)

**Données :** un entier $n$

**Résultat :** un entier valant $n!$

$fact, i$ : entier;

**début**

  $fact \leftarrow 2$;

  **pour** $i$ *allant de* 3 *à* $n$ **faire**

    $fact \leftarrow fact * i$;

  **retourner** $fact$;

| | nombre | coût |
|---|---|---|
| initialisation : | $\Theta(1)\times$ | $\Theta(1)$ |

# Exemple : calcul de la factorielle de $n \in \mathbb{N}$

- Reprenons le calcul de la factorielle, qui nécessitait $3n$ opérations :

**Algorithme :** Factorielle(n)

**Données :** un entier $n$

**Résultat :** un entier valant $n!$

$fact, i$ : entier;

**début**

  $fact \leftarrow 2$;

  **pour** $i$ *allant de* 3 *à* $n$ **faire**

    $fact \leftarrow fact * i$;

  **retourner** $fact$;

| | nombre | coût |
|---|---|---|
| initialisation : | $\Theta(1)\times$ | $\Theta(1)$ |
| itérations : | $\Theta(n)\times$ | $\Theta(1)$ |

# Exemple : calcul de la factorielle de $n \in \mathbb{N}$

- Reprenons le calcul de la factorielle, qui nécessitait $3n$ opérations :

**Algorithme :** Factorielle(n)

**Données :** un entier $n$

**Résultat :** un entier valant $n!$

$fact, i$ : entier;

**début**

  $fact \leftarrow 2$;

  **pour** $i$ *allant de* 3 *à* $n$ **faire**

    $fact \leftarrow fact * i$;

  **retourner** $fact$;

| | nombre | coût |
|---|---|---|
| initialisation : | $\Theta(1)\times$ | $\Theta(1)$ |
| itérations : | $\Theta(n)\times$ | $\Theta(1)$ |
| mult. + affect. : | $\Theta(n)\times$ | $\Theta(1)$ |

# Exemple : calcul de la factorielle de $n \in \mathbb{N}$

- Reprenons le calcul de la factorielle, qui nécessitait $3n$ opérations :

**Algorithme :** Factorielle(n)
**Données :** un entier $n$
**Résultat :** un entier valant $n!$
$fact, i$ : entier;
**début**
  $fact \leftarrow 2$;
  **pour** $i$ *allant de* 3 *à* $n$ **faire**
    $fact \leftarrow fact * i$;
  **retourner** $fact$;

| | nombre | coût |
|---|---|---|
| initialisation : | $\Theta(1)\times$ | $\Theta(1)$ |
| itérations : | $\Theta(n)\times$ | $\Theta(1)$ |
| mult. + affect. : | $\Theta(n)\times$ | $\Theta(1)$ |
| retour fonction : | $\Theta(1)\times$ | $\Theta(1)$ |

# Exemple : calcul de la factorielle de $n \in \mathbb{N}$

- Reprenons le calcul de la factorielle, qui nécessitait $3n$ opérations :

**Algorithme :** Factorielle(n)
**Données :** un entier $n$
**Résultat :** un entier valant $n!$
$fact, i$ : entier;
**début**
  $fact \leftarrow 2$;
  **pour** $i$ *allant de* 3 *à* $n$ **faire**
    $fact \leftarrow fact * i$;
  **retourner** $fact$;

| | nombre | coût |
|---|---|---|
| initialisation : | $\Theta(1)\times$ | $\Theta(1)$ |
| itérations : | $\Theta(n)\times$ | $\Theta(1)$ |
| mult. + affect. : | $\Theta(n)\times$ | $\Theta(1)$ |
| retour fonction : | $\Theta(1)\times$ | $\Theta(1)$ |

Nombre total d'opérations :

$$\Theta(1) + \Theta(n) * \Theta(1) + \Theta(n) * \Theta(1) + \Theta(1) = \Theta(n)$$

# Exemple : TriSélection

**Algorithme :** `TriSélection`

**Données :** tableau $L$ de $n$ éléments comparables

**Résultat :** le tableau trié

```
1 pour i allant de 1 à n faire
2     m ← i;
3     pour j allant de i + 1 à n faire
4         si L[j] < L[m] alors
5             m ← j;
6     échanger L[i] et L[m];
7 retourner L;
```

| | nombre | coût |
|---|---|---|
| itérations : | $n\times$ | 1 op. |
| affectation : | $n\times$ | 1 op. |
| itérations : | $\sum_{i=1}^{n}(n-i-1)\times$ | 1 op. |
| comparaison : | $\sum_{i=1}^{n}(n-i-1)\times$ | 1 op. |
| affectation : | $n\times ?\times$ | 1 op. |
| échange : | $n\times$ | 3 op. |

# Exemple : TriSélection

**Algorithme :** `TriSélection`

**Données :** tableau $L$ de $n$ éléments comparables

**Résultat :** le tableau trié

| | | nombre | coût |
|---|---|---|---|
| 1 **pour** *$i$ allant de* 1 *à* $n$ **faire** | itérations : | $n\times$ | 1 op. |
| 2 $\quad m \leftarrow i$; | affectation : | $n\times$ | 1 op. |
| 3 $\quad$ **pour** *$j$ allant de* $i+1$ *à* $n$ **faire** | itérations : | $\sum_{i=1}^{n}(n-i-1)\times$ | 1 op. |
| 4 $\quad\quad$ **si** $L[j] < L[m]$ **alors** | comparaison : | $\sum_{i=1}^{n}(n-i-1)\times$ | 1 op. |
| 5 $\quad\quad\quad m \leftarrow j$; | affectation : | $n\times ?\times$ | 1 op. |
| 6 $\quad$ échanger $L[i]$ et $L[m]$; | échange : | $n\times$ | 3 op. |
| 7 **retourner** $L$; | | | |

## Séries arithmétiques

$$\sum_{i=1}^{n}(n-i-1) = n^2 - n - \sum_{i=1}^{n} i = n^2 - n - (1+2+3+\cdots+n) = n^2 - n - \frac{1}{2}n(n+1)$$

# Exemple : TriSélection

**Algorithme :** `TriSélection`

**Données :** tableau $L$ de $n$ éléments comparables

**Résultat :** le tableau trié

| | | nombre | coût |
|---|---|---|---|
| 1 **pour** *$i$ allant de* 1 *à* $n$ **faire** | itérations : | $n\times$ | 1 op. |
| 2 $\quad m \leftarrow i$; | affectation : | $n\times$ | 1 op. |
| 3 $\quad$ **pour** *$j$ allant de* $i+1$ *à* $n$ **faire** | itérations : | $\sum_{i=1}^{n}(n-i-1)\times$ | 1 op. |
| 4 $\quad\quad$ **si** $L[j] < L[m]$ **alors** | comparaison : | $\sum_{i=1}^{n}(n-i-1)\times$ | 1 op. |
| 5 $\quad\quad\quad m \leftarrow j$; | affectation : | $n\times ?\times$ | 1 op. |
| 6 $\quad$ échanger $L[i]$ et $L[m]$; | échange : | $n\times$ | 3 op. |
| 7 **retourner** $L$; | | | |

## Séries arithmétiques

$$\sum_{i=1}^{n}(n-i-1) = n^2 - n - \sum_{i=1}^{n} i = n^2 - n - (1+2+3+\cdots+n) = n^2 - n - \frac{1}{2}n(n+1)$$

Nombre total d'opérations : $\Theta(n^2) = \Theta(|T|^2)$