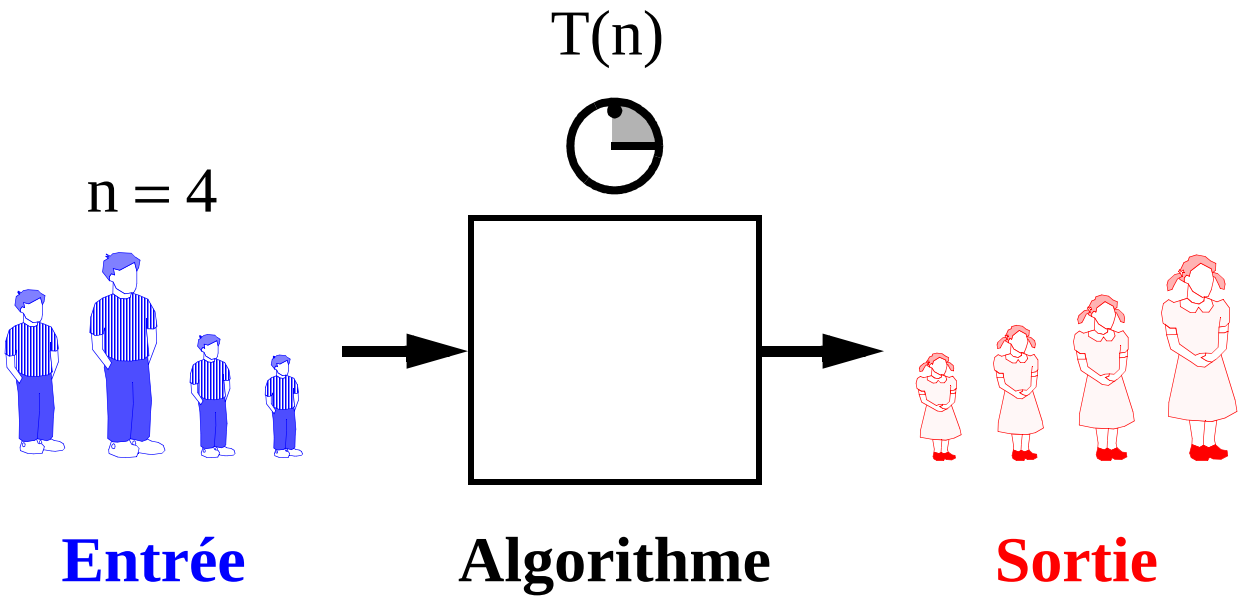
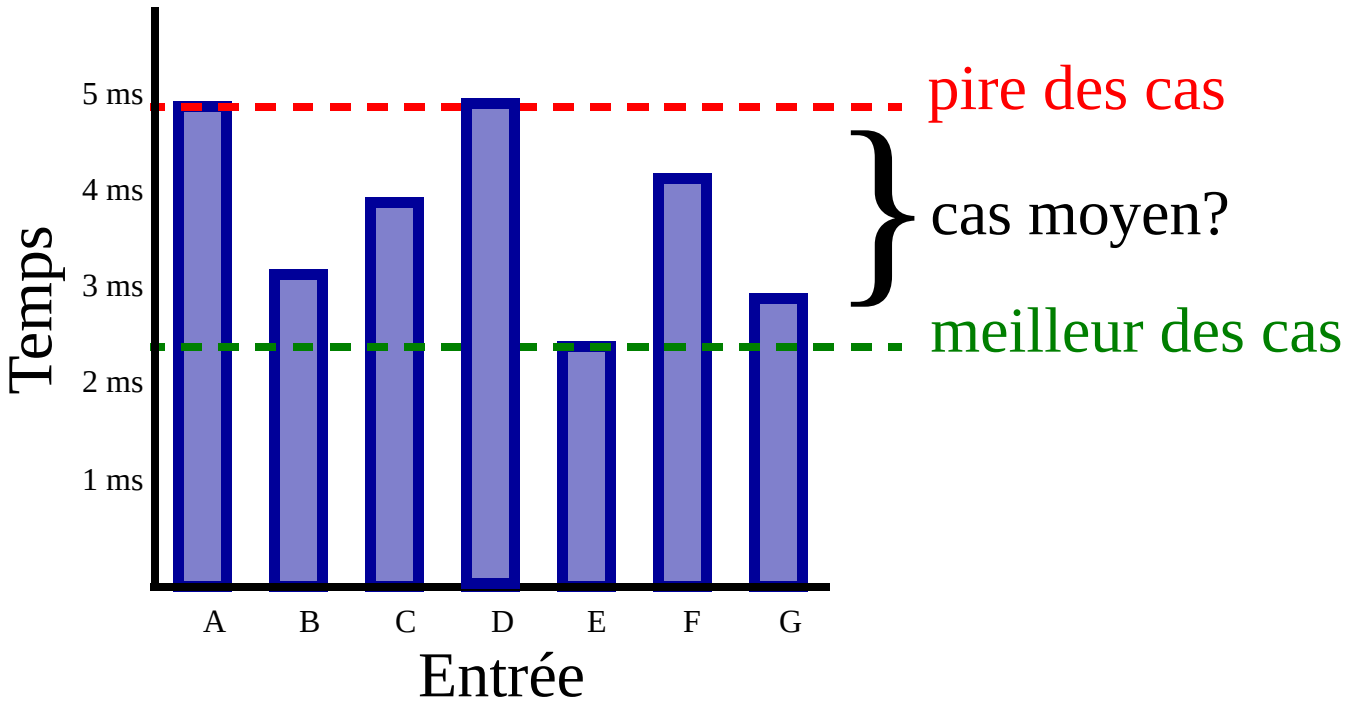
ANALYSE D'ALGORITHMES

- Révision mathématique rapide
- Temps d'exécution
- Pseudo-code
- Analyse d'algorithmes
- Notation asymptotique
- Analyse asymptotique



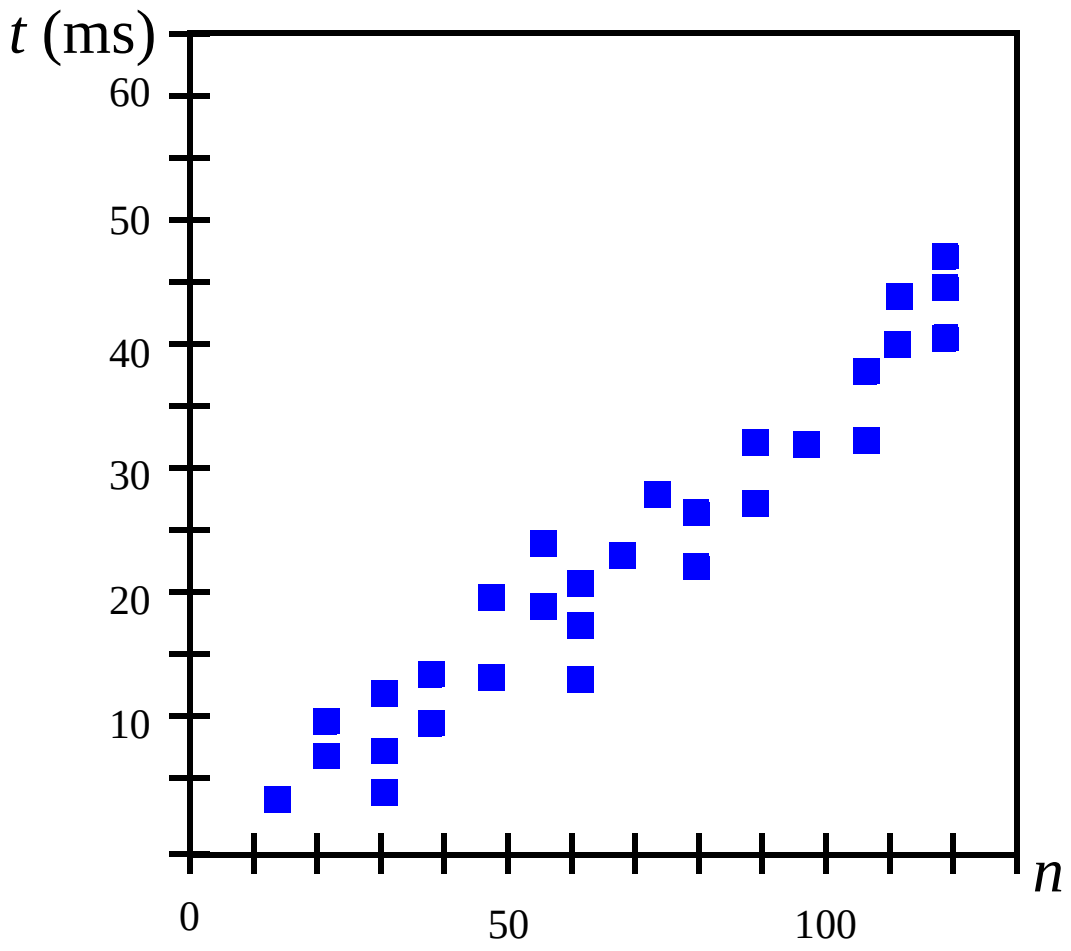
Cas moyen vs. Pire des cas: Temps d'exécution d'un algorithme

- Un algorithme peut être plus performant avec certains ensembles de données qu'avec d'autres,
- Trouver le cas moyen peut s'avérer difficile, alors les algorithmes sont mesurés typiquement selon la complexité temporelle du pire des cas.
- De plus, pour certain domaines d'application (par ex. contrôle aérien, chirurgie, gestion de réseau) connaître la complexité temporelle du pire des cas est d'importance cruciale.



Mesurer le temps d'exécution

- Comment devrions-nous mesurer le temps d'exécution d'un algorithme?
- Étude expérimentale:
  - Écrivez un programme qui réalise l'algorithme.
  - Exécutez le programme avec des ensembles de données de taille et de contenu variés.
  - Utilisez une méthode (System.currentTimeMillis()) pour mesurer précisément le temps d'exécution.
  - Les mesures résultantes devraient ressembler à:



Au-delà des études expérimentales

- Les études expérimentales ont quelques restrictions:
  - Il est nécessaire de réaliser et de tester l'algorithme afin de déterminer son temps d'exécution.
  - Les essais peuvent être faits seulement sur un ensemble limité d'entrées, et ils peuvent ne pas être indicatifs du temps d'exécution d'autres entrées non considérées.
  - Afin de comparer deux algorithmes, les mêmes environnements matériel et logiciel devraient être utilisés.
- Nous développerons maintenant une méthodologie générale pour analyser le temps d'exécution d'algorithmes qui:
  - Utilise une description de haut niveau de l'algorithme au lieu de tester sa réalisation.
  - Considère toutes les entrées possibles.
  - Permet d'évaluer l'efficacité d'un algorithme indépendamment des environnements matériels et logiciels.

Pseudo-code

- Le pseudo-code est une description d'algorithme qui est plus structurée que la prose ordinaire mais moins formelle qu'un langage de programmation.
- Exemple: trouver l'élément maximal d'un vecteur (*array*).

Algorithm arrayMax(A, *n*):
    Entrée: Un vecteur A contenant *n* entiers.
    Sortie: L'élément maximal de A.
  currentMax ← A[0]
  for *i* ← 1 **to** *n* –1 **do**
    if *currentMax* < A[*i*] **then**
      currentMax ← A[*i*]
  return *currentMax*

- Le pseudo-code est notre notation de choix pour la description d'algorithmes.
- Cependant, le pseudo-code cache plusieurs problèmes liés à la conception de programmes.

Qu’est-ce que le pseudo-code?

- Un mélange de langage naturel et de concepts de programmation de haut niveau qui décrit les idées générales derrière la réalisation générique d’une structure de données ou d’un algorithme.
  - Expressions: utilisez des symboles mathématiques standards pour décrire des expressions booléennes et numériques
    - utilisez ← pour des affectations (“=” en Java)
    - utilisez = pour la relation d’égalité (“==” en Java)
  - Déclaration de méthodes:
    - **Algorithm** nom(*param1*, *param2*)
  - Éléments de programmation:
    - décision: **if** ... **then** ... **[else** ... **]**
    - boucle while: **while** ... **do**
    - boucle repeat: **repeat** ... **until** ...
    - boucle for: **for** ... **do**
    - indexage de vecteur: **A[*i*]**
  - Méthodes:
    - appel: object method(args)
    - retour: **return** value

Analyse d'algorithmes

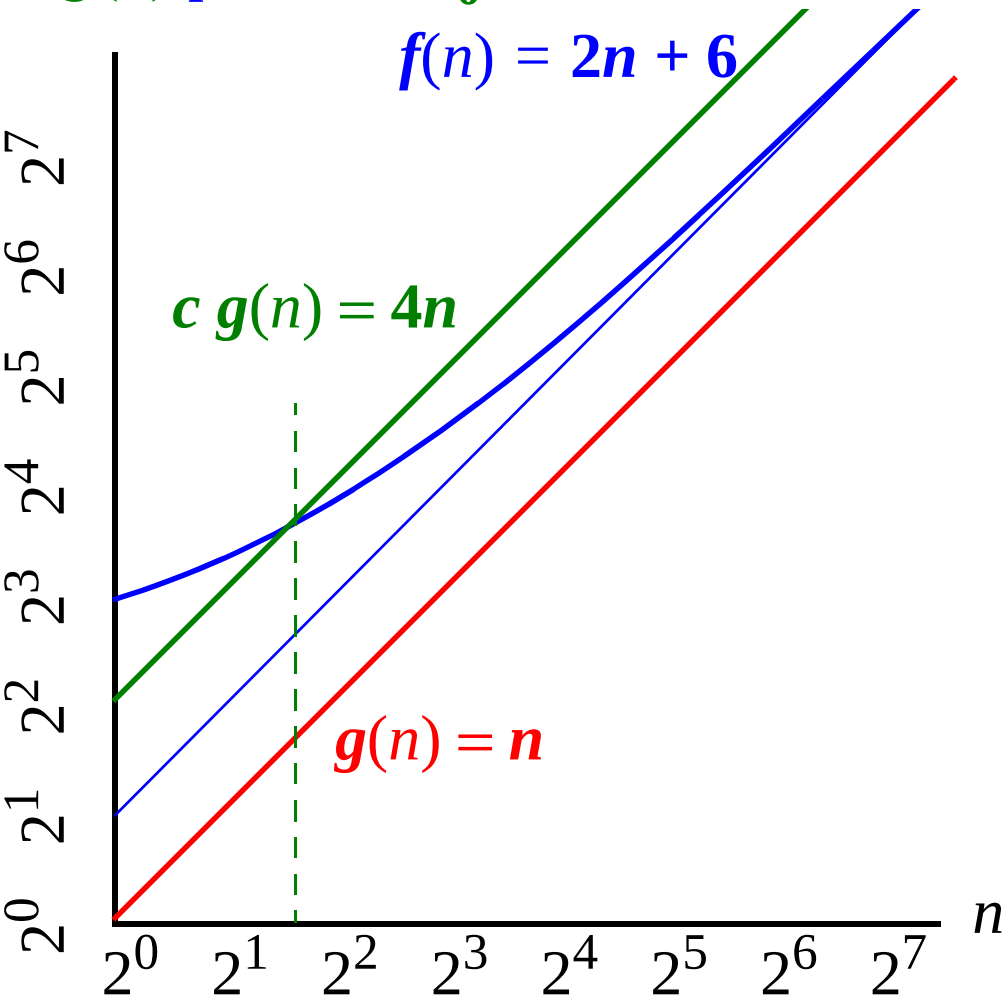
- Opérations primitives: opérations de bas niveau qui sont largement indépendantes du langage de programmation et qui peuvent être identifiées en pseudo-code, par exemple:
  - Appel et retour d'une méthode
  - effectuer une opération arithmétique (addition)
  - comparer deux nombres, etc.
- En inspectant le pseudo-code, nous pouvons compter le nombre d'opérations primitives exécutées par un algorithme.
- Exemple:

```
Algorithm arrayMax(A, n):
   Entrée: Un vecteur A contenant n entiers.
   Sortie: L'élément maximal de A.
currentMax ← A[0]
for i ← 1 to n –1 do
   if currentMax < A[i] then
      currentMax ← A[i]
return currentMax
```

Notation asymptotique

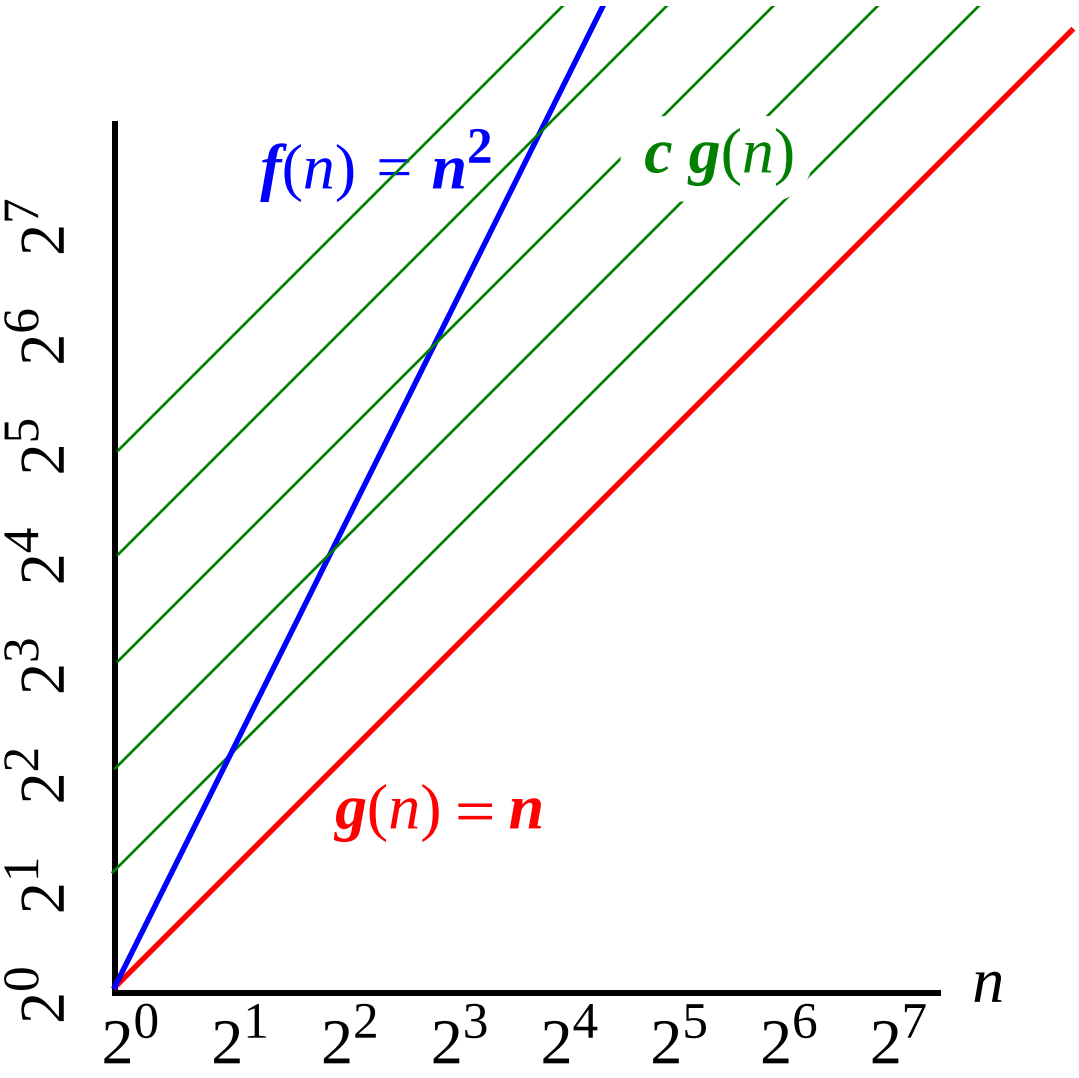
- But: simplifier l'analyse en se débarrassant de l'information superflue.
  - comme "arrondir" 1 000 001 $\approx$ 1 000 000
  - nous désirons indiquer formellement que $3n^2 \approx n^2$

- La notation "Grand-O"
  soit les fonctions $f(n)$ et $g(n)$, nous disons que $f(n)$ est $O(g(n))$ **si et seulement si**
  il y a des constantes positives c et n_0 tel que
  $f(n) \leq c\, g(n)$ pour $n \geq n_0$



Un autre exemple

- n^2 **n'est pas** O(n)
- nous ne pouvons pas trouver c et n_0 tel que $n^2 \leq c\,n$ for $n \geq n_0$



Notation asymptotique (suite)

- Note: Même si il est correct de dire "$7n - 3$ est $\boldsymbol{O}(n^3)$", une meilleure formulation est "$7n - 3$ est $\boldsymbol{O}(n)$", c'est-à-dire, nous devrions faire l'approximation la plus juste possible.
- Règle simple: laissez tomber les termes d'ordre inférieur de même que les facteurs
  - $7n - 3$ est $\boldsymbol{O}(n)$
  - $8n^2 \log n + 5n^2 + n$ est $\boldsymbol{O}(n^2 \log n)$
- Classes spéciales d'algorithmes:
  - logarithmique: $\boldsymbol{O}(\log n)$
  - linéaire $\boldsymbol{O}(n)$
  - quadratique $\boldsymbol{O}(n^2)$
  - polynomial $\boldsymbol{O}(n^k)$, $k \geq 1$
  - exponentiel $\boldsymbol{O}(a^n)$, $n > 1$
- "Parenté" de Grand-O
  - $\Omega(f(n))$: Grand-Oméga
  - $\Theta(f(n))$: Grand-Thêta

Analyse asymptotique et temps d'exécution

- Utilisez la notation Grand-O pour indiquer le nombre d'opérations primitives exécutées en fonction de la taille d'entrée.
- Par exemple, nous disons que l'algorithme arrayMax a un temps d'exécution $\boldsymbol{O}(n)$.
- En comparant les temps d'exécution asymptotiques
  - un algorithme d'ordre $\boldsymbol{O}(n)$ est meilleur qu'un autre d'ordre $\boldsymbol{O}(n^2)$
  - de la même façon, $\boldsymbol{O}(\log n)$ est meilleur que $\boldsymbol{O}(n)$
  - hiérarchie de fonctions:
  - $\log n << n^{-2} << n << n \log n << n^2 << n^3 << 2^n$
- Attention!
  - Méfiez-vous des facteurs constants très grands. Un algorithme au temps d'exécution 1 000 000 n est quand même $\boldsymbol{O}(n)$ et peut être moins efficace sur votre ensemble de données qu'un autre au temps d'exécution $2n^2$, qui est $\boldsymbol{O}(n^2)$.

Exemple d'analyse asymptotique

- Un algorithme pour calculer les moyennes préfixes:

Algorithm prefixAverages1(X):
Entrée: Un vecteur de nombres X à n éléments.
Sortie: Un vecteur de nombres A à n éléments tel que A[i] est la moyenne des éléments X[0], ... , X[i].
Soit A un vecteur de n nombres.
for $i \leftarrow 0$ **to** n - 1 **do**
  $a \leftarrow 0$
  for $j \leftarrow 0$ **to** i **do**
    $a \leftarrow a + X[j]$
  A[i] $\leftarrow a/(i + 1)$
return array A

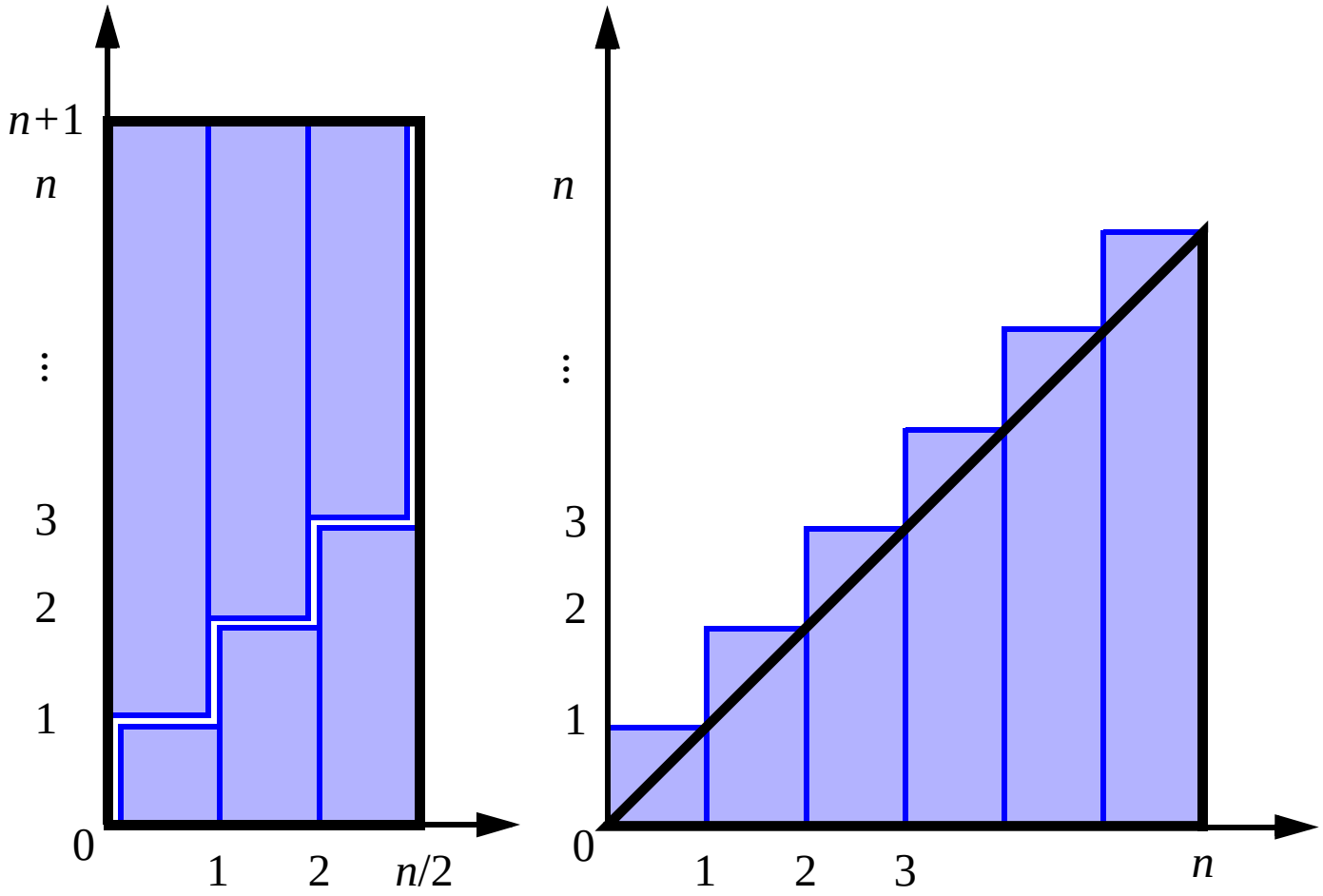
- Analyse ...

Révision mathématique rapide

- Progression arithmétique:
  - Un exemple

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + n = \frac{n^2 + n}{2}$$

  - deux représentations visuelles



Un autre exemple

- Un meilleur algorithme pour calculer les moyennes préfixes:

Algorithm prefixAverages2(X):
Entrée: Un vecteur de nombres X à n éléments.
Sortie: Un vecteur de nombres A à n éléments tel que A[i] est la moyenne des éléments X[0], ... , X[i].
Soit A un vecteur de n nombres.
$s \leftarrow 0$
for $i \leftarrow 0$ **to** n - 1 **do**
    $s \leftarrow s + X[i]$
    A[i] $\leftarrow s/(i + 1)$
return array A

- Analyse ...

Mathématiques à réviser

- Logarithmes et exposants
  - propriétés des logarithmes:

$$\log_b(xy) = \log_b x + \log_b y$$

$$\log_b(x/y) = \log_b x - \log_b y$$

$$\log_b x^{\alpha} = \alpha \log_b x$$

$$\log_b a = \frac{\log_x a}{\log_x b}$$

  - propriétés des exposants:

$$a^{(b+c)} = a^b a^c$$

$$a^{bc} = (a^b)^c$$

$$a^b / a^c = a^{(b-c)}$$

$$b = a^{\log_a b}$$

$$b^c = a^{c*\log_a b}$$

Mathématiques à réviser (suite)

- Plancher (*Floor*)

  $\lfloor x \rfloor$ = le plus grand entier $\leq x$

- Plafond (*Ceiling*)

  $\lceil x \rceil$ = le plus petit entier $\geq x$

- Sommations
  - définition générale:

$$\sum_{i=s}^{t} f(i) = f(s) + f(s+1) + f(s+2) + \ldots + f(t)$$

  - où f est une fonction, s est l'index de départ, et t est l'index d'arrivée

- Progression géométrique: $f(i) = a^i$
  - soit un entier $n \geq 0$ et un nombre réel $0 < a \neq 1$

$$\sum_{i=0}^{n} a^i = 1 + a + a^2 + \ldots + a^n = \frac{1 - a^{n+1}}{1 - a}$$

  - les progressions géométriques ont une croissance exponentielle.

Sujets avancés: techniques de justification simples

- Par exemple
  - Trouvez un exemple
  - Trouvez un contre-exemple

- Par contradiction (“*Contra” Attack*)
  - Trouvez une contradiction dans l’inverse de l’énoncé
  - Contrapositive

- Induction et invariants de boucle
  - Induction
    - 1) Prouvez le cas de base
    - 2) Prouvez que n’importe quel cas n implique que le prochain cas (n + 1) est aussi vrai
  - Invariants de boucle
    - 1) Prouvez l’énoncé initial S_0
    - 2) Démontrez que $S_{i\text{-}1}$ implique que S_i sera vrai après l’itération i

Sujets avancés: autres techniques de justification

- Preuve par excès d'agitation des mains
- Preuve par diagramme incompréhensible
- Preuve par corruption
  - voir le professeur ou l'AE après la classe
- La méthode des nouveaux habits de l'Empereur
  - "Cette preuve est tellement évidente que seul un idiot serait incapable de la comprendre"