Algorithmique

Analyse des algorithmes

Anicet E. T. Ebou, ediman.ebou@inphb.ci

01

Temps d’exécution d’un algorithme

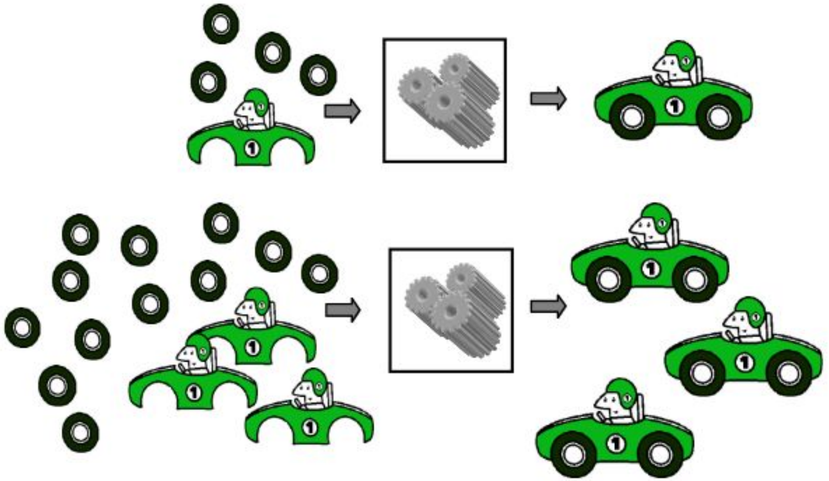
Efficacité d’un algorithme

Il peut être évalué en terme de:

- Temps d’exécution;
- Espace mémoire occupé;
- Qualité du résultat;
- Simplicité.

Temps d'exécution d'un algorithme

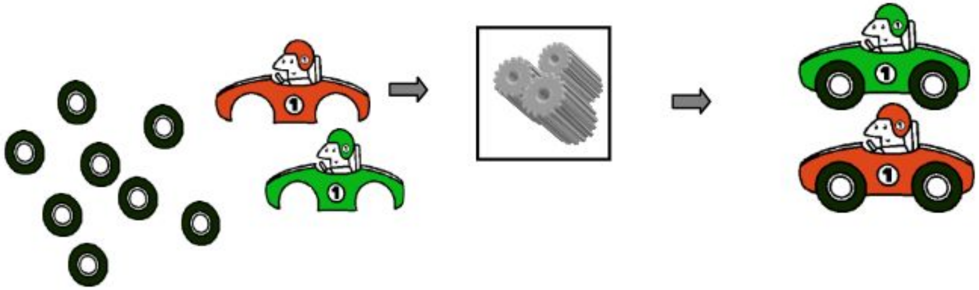
Le temps d'exécution d'un algorithme dépend de **la taille** des données d'entrées.



Tiré de CSI2510 - Prof. Paola Flocchini

Temps d’exécution d’un algorithme

Il dépend aussi de la nature des données à traiter (des entrées différentes peuvent avoir des temps d’exécution différents).

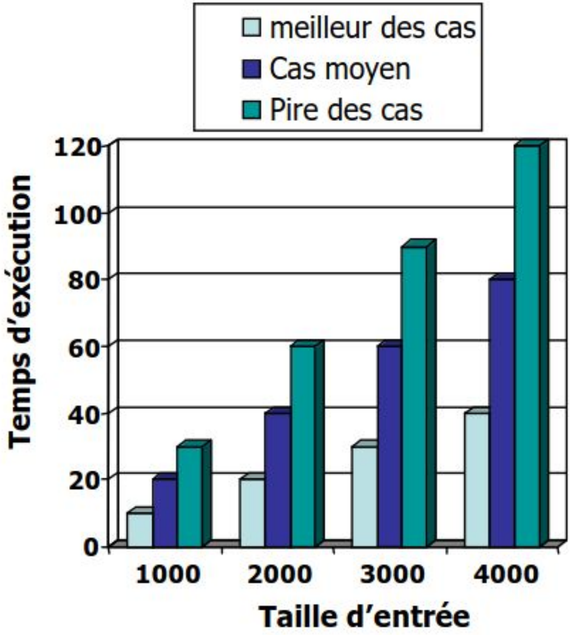


Tiré de CSI2510 - Prof. Paola Flocchini

Temps d'exécution d'un algorithme

En fonction de ces facteurs on a donc en général 3 mesures du temps d'exécution:

- Le meilleur des cas;
- Le cas moyen;
- Le pire des cas.

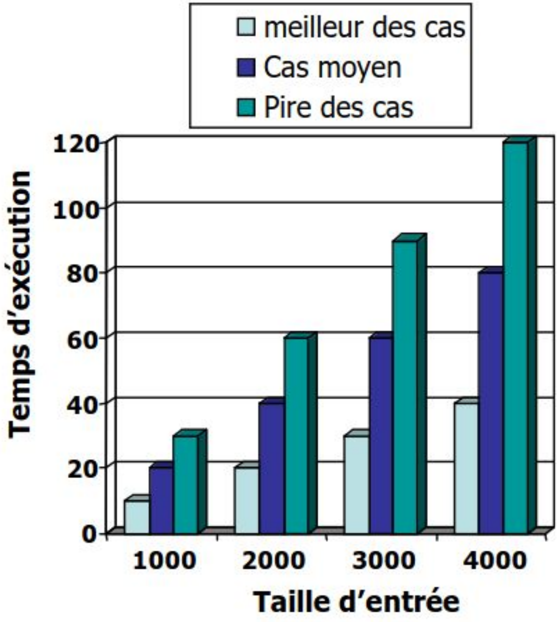


Tiré de CSI2510 - Prof. Paola Flocchini

Temps d’exécution d’un algorithme

- Trouver **le cas moyen** peut être difficile
- On se concentre souvent sur **le pire des cas**.
    - plus facile a analyser
    - d'importance cruciale

dans certaines applications (par ex. contrôle aérien, chirurgie, gestion de réseau).



Tiré de CSI2510 - Prof. Paola Flocchini

02

Mesurer le temps d’exécution

Mesurer le temps d'exécution

Deux approches sont possibles:

1. Une approche expérimentale
2. Une approche théorique.

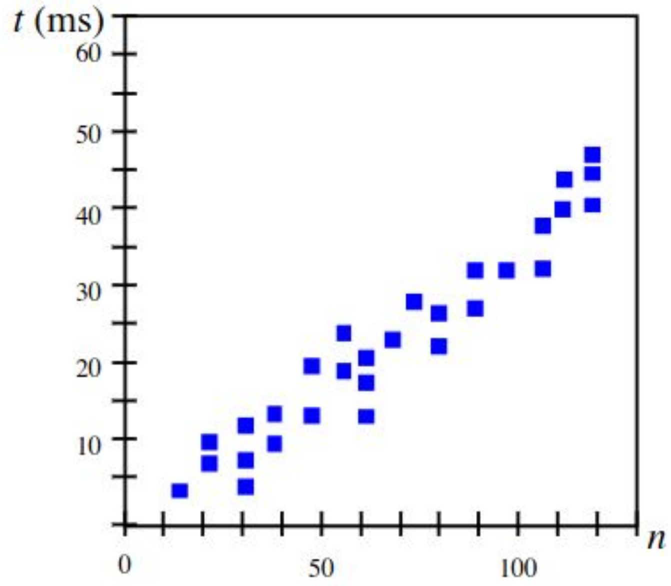
2.1

Approche expérimentale

Etude expérimentale

L'étude expérimentale peut se faire en suivant les étapes suivantes:

1. Implémenter l'algorithme;
2. Exécuter le programme avec des ensembles de données de taille et de contenu variés;
3. Mesurer précisément le temps d'exécution pour chaque cas.



Etude expérimentale

Les études expérimentales ont des limitations non négligeables:

- Il est nécessaire d'**implémenter** l'algorithme dans un langage de programmation;
- Lors des tests, l'ensemble des données d'entrée est réduit et ne couvre pas la totalité des cas possibles;
- Afin de comparer deux algorithmes, les **mêmes environnements matériel et logiciel** devraient être utilisés.

2.2

Approche théorique

Etude théorique

Nous avons besoin d’une **méthodologie générale** pour analyser le temps d’exécution d’algorithmes qui:

- Utilise une description de haut niveau de l’algorithme (indépendant de l'implémentation);
- Caractérise le temps d'exécution comme une fonction de la taille des données d'entrée;
- Considère toutes les entrées;
- Est indépendant des environnements matériels et logiciels.

Opérations primitives

Ce sont des opérations de bas niveau qui sont indépendantes du langage de programmation, par exemple:

- Appel et retour d'une méthode;
- Effectuer une opération arithmétique;
- Comparer deux nombres, etc…;
- Affectation d'une variable.

Opérations primitives

En observant le pseudo-code d’un algorithme on peut compter le nombre d’opérations primitives exécutées par cet algorithme et par la suite analyser son temps d’exécution et son efficacité.

Etude théorique - Exemple

```
Algorithme max_tab
Var
     tab: Tableau[0..n-1] d'Entiers
     max: Entier
Debut
     max ← tab[0]
     Pour i ← 1 à n-1 faire
          Si max < tab[i] alors
               max <- tab[i]
          FinSi
     FinPour
     Ecrire('Le max est: ' , max)
Fin
```

Etude théorique - Exemple

```
Algorithme max_tab
Var
     tab: Tableau[0..n-1] d'Entiers
     max: Entier
Debut
     max ← tab[0]
     Pour i ← 1 à n-1 faire
          Si max < tab[i] alors
               max <- tab[i]
          FinSi
     FinPour
     Ecrire('Le max est: ' , max)
Fin
```

Quelles sont les opérations primitives à compter?

- Comparaisons;
- Affectations à Max.

Etude théorique - Exemple

```
Algorithme max_tab
Var
     tab: Tableau[0..n-1] d'Entiers
     max: Entier
Debut
     max ← tab[0]
     Pour i ← 1 à n-1 faire
          Si max < tab[i] alors
               max <- tab[i]
          FinSi
     FinPour
     Ecrire('Le max est: ' , max)
Fin
```

Meilleur des cas

20	2	3	4	5	6	7	8	9	1

1 affectation

n-1 comparaisons

0 affectation

Etude théorique - Exemple

```
Algorithme max_tab
Var
      tab: Tableau[0..n-1] d'Entiers
      max: Entier
Debut
      max ← tab[0]
      Pour i ← 1 à n-1 faire
            Si max < tab[i] alors
                  max <- tab[i]
            FinSi
      FinPour
      Ecrire('Le max est: ' , max)
Fin
```

Pire des cas

1	2	3	4	5	6	7	8	9	20

1 affectation

n-1 comparaisons

n-1 affectations

Etude théorique - Exemple

```
Algorithme max_tab
Var
     tab: Tableau[0..n-1] d'Entiers
     max: Entier
Debut
     max ← tab[0]
     Pour i ← 1 à n-1 faire
          Si max < tab[i] alors
               max <- tab[i]
          FinSi
     FinPour
     Ecrire('Le max est: ' , max)
Fin
```

Meilleur des cas

1 affectation + (n-1) comparaisons

Pire des cas

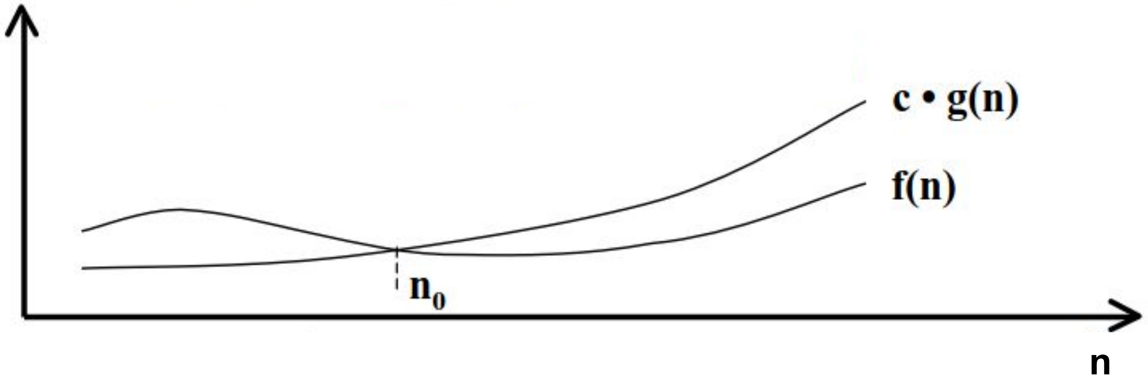
n affectations + (n-1) comparaisons

03

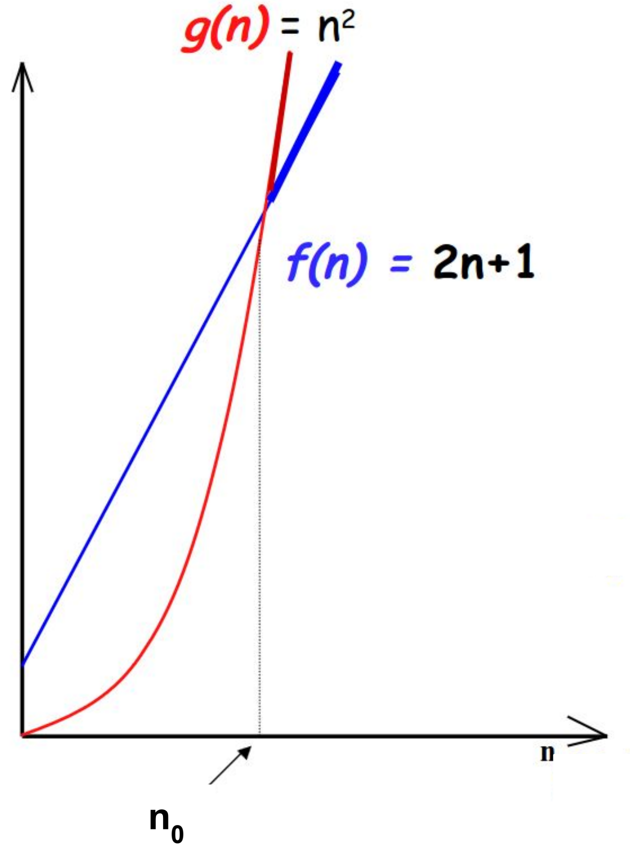
Notation asymptotique

Big-Oh (Grand Oh) - Limite supérieure

Soit les fonctions f(n) et g(n), nous disons que f(n) est O(g(n)) (ou f(n) = O(g(n)) ou f(n) ∈ O(g(n)) si et seulement si il y a des constantes positives c et n_0 tel que $f(n) \le c\ g(n)$ pour $n \ge n_0$.



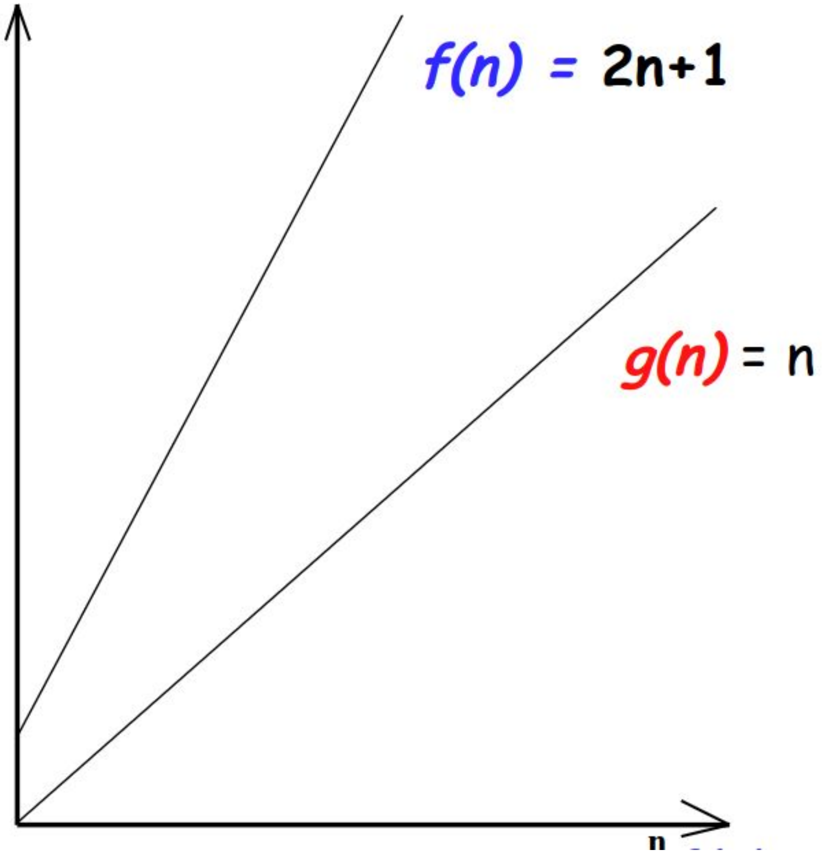
Big-Oh (Grand Oh) - exemple graphique



f(n) est $O(n^2)$, car il existe un c et un n_0 tel que f(n) ≤ c g(n) pour $n \geq n_0$.

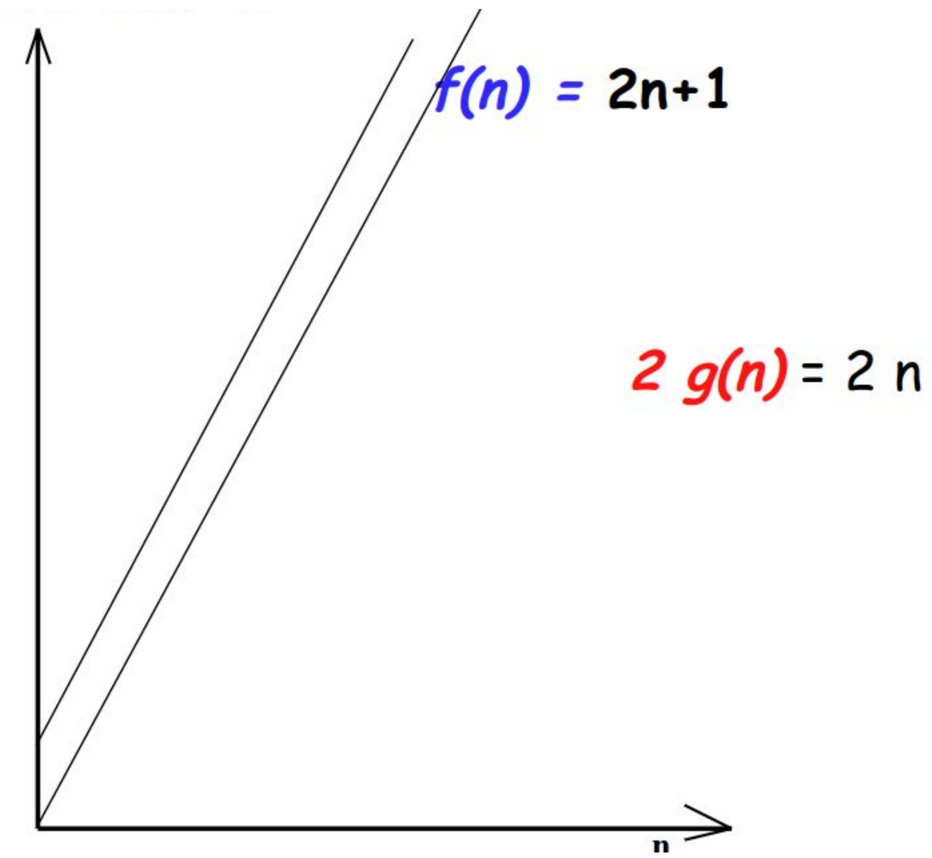
Big-Oh (Grand Oh) - exemple graphique

Mais on a aussi



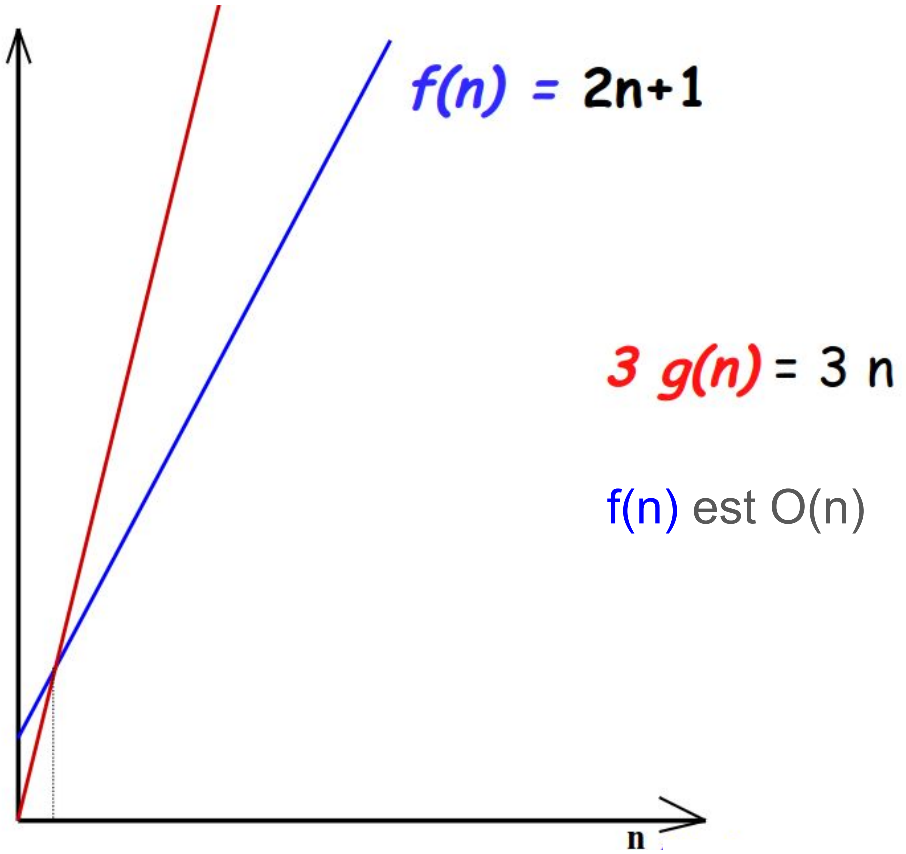
Big-Oh (Grand Oh) - exemple graphique

Mais on a aussi



Big-Oh (Grand Oh) - exemple graphique

Mais on a aussi

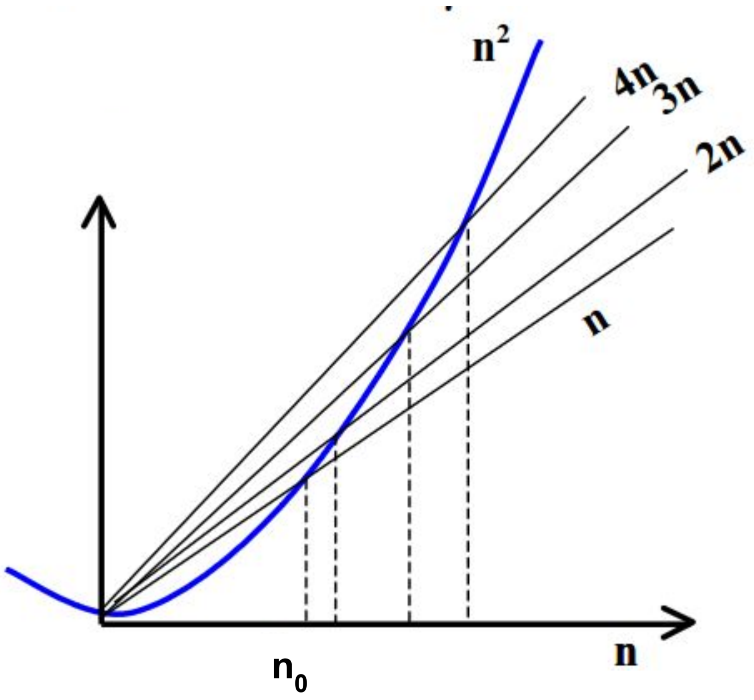


f(n) est O(n)

Big-Oh (Grand Oh) - Limite supérieure

Mais n^2 n'est pas O(n) parce que nous ne pouvons pas trouver c et n_0 tel que $n^2 \leq c\, n$ pour $n \geq n_0$.

En d'autre termes, n'importe comment grand un c est choisi il y a un n assez grand tel que $n^2 > cn$.



Big-Oh (Grand Oh) - Limite supérieure: exemple

Prouver que $f(n) = 60n^2 + 5n + 1$ est $O(n^2)$.

Il faut trouver un nombre c et un nombre n_0 tel que:

$60n^2 + 5n + 1 \leq c\, n^2$ pour tout $n \geq n_0$

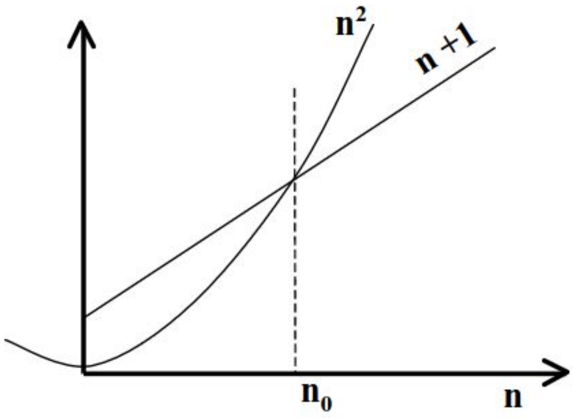
$$\left.\begin{array}{l} 5n \leq 5n^2 \text{ pour tout } n \geq 1 \\ 1 \leq n^2 \text{ pour tout } n \geq 1 \end{array}\right\} \quad f(n) \leq 60n^2 + 5n^2 + n^2 \text{ pour tout } n \geq 1$$

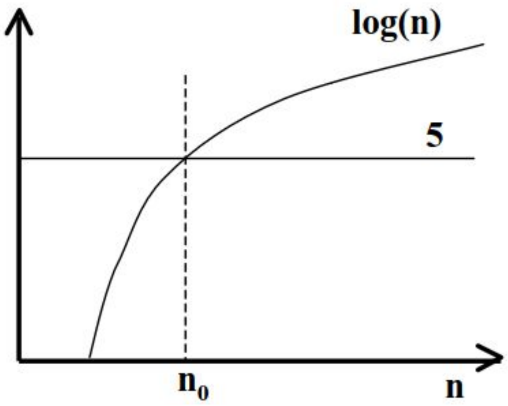
$f(n) \leq 66n^2$ pour tout $n \geq 1$ $\quad c = 66$ et $n_0 = 1 \Rightarrow f(n) = O(n^2)$

Big-Oh (Grand Oh) - Limite supérieure

A mémoriser:

$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$





Big-Oh (Grand Oh) - Limite supérieure

n =	2	16	256	1024
log log n	0	2	3	3.32
log n	1	4	8	10
n	2	16	256	1024
n log n	2	64	448	10 200
n^2	4	256	65 500	$1.05 * 10^6$
n^3	8	4 100	16 800 800	$1.07 * 10^9$
2^n	4	35 500	$11.7 * 10^6$	$1.80 * 10^{308}$

Big-Oh (Grand Oh) - Limite supérieure

Théorème: Si $g(n)$ est $O(f(n))$, alors pour n'importe quelle constante $c > 0$, $g(n)$ est aussi $O(c\, f(n))$.

Théorème: Si $f_1(n) = O(g_1(n))$ et $f_2(n) = O(g_2(n))$ alors $f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n)))$.

Exemple 1: $2n^3 + 3n^2 = O(\max(2n^3, 3n^2)) = O(2n^3) = O(n^3)$

Exemple 2: $n^2 + 3 \log n - 7 = O(\max(n^2, 3 \log n - 7)) = O(n^2)$

Big-Oh (Grand Oh) - Limite supérieure

Pour donner le Grand Oh, il faut faire l'approximation la plus proche possible. C'est à dire:

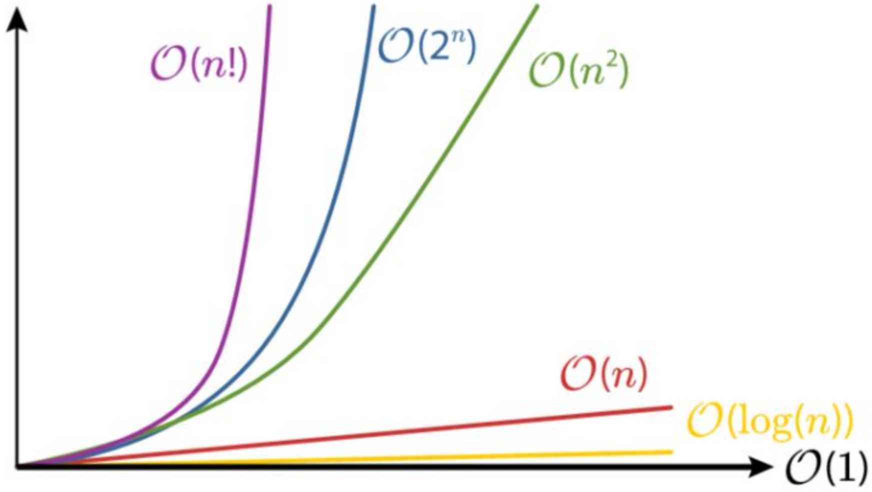
- Utiliser la plus petite classe possible: par exemple, Il est correct de dire que $5n - 3$ est $O(n^3)$ mais la meilleure formulation est de dire $5n - 3$ est $O(n)$.
- Utiliser l'expression la plus simple de la classe: dire $10n + 15$ est $O(n)$ au lieu de $10n + 15$ est $O(10n)$.

Big-Oh (Grand Oh) - Limite supérieure

- Laisser tomber les termes d'ordre inférieur ainsi que les coefficients:
  - $7n - 3$ est $O(n)$;
  - $6n^2\log(n) + 3n^2 + 5n$ est $O(n^2\log n)$;
  - $n^5 + 1000n^4 + 20n^3 - 8$ est $O(n^5)$.

Classes de complexité

- Constant: $O(1)$
- Logarithmique: $O(\log n)$
- Linéaire: $O(n)$
- Sous-quadratique: $O(n \log n)$
- Quadratique: $O(n^2)$
- Cubique: $O(n^3)$
- Polynomiale: $O(n^k)$, $k \geq 1$
- Exponentielle: $O(a^n)$, $n > 1$
- Factorielle: $O(n!)$



Mathématiques à réviser

Proprieté des logarithmes:

- $\log_b(xy) = \log_b x + \log_b y$
- $\log_b (x/y) = \log_b x - \log_b y$
- $\log_b xa = a\log_b x$
- $\log_b a = \log_x a/\log_x b$

Proprieté 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

- Plancher (floor): $\lfloor x \rfloor$ = le plus grand entier $\leq x$ $\lfloor 2.3 \rfloor = 2$
- Plafond (ceiling): $\lceil x \rceil$ = le plus petit entier $\geq x$ $\lceil 2.3 \rceil = 3$
- Progression arithmétique
- Progression géométrique

Références

CSI2510 - Prof. Paola Flocchini