UNIVERSITÉ CENTRALE
HONORIS UNITED UNIVERSITIES

Algorithmique et Structures de Données II

Année universitaire
2020-2021

Dr. Marwa CHAIEB

Complexité

Complexité d'un algorithme

- La complexité d'un algorithme est le nombre d'opérations élémentaires qu'il doit effectuer pour mener à bien un calcul en fonction de la taille des données d'entrée.
- Nous avons donc deux éléments à prendre en compte :
  - La taille des données ;
  - Le temps de calcul.

Efficacité

L'efficacité d'un algorithme peut etre évaluée par:

- rapidité (en termes de temps d'exécution)
- consommation de ressources (espace de stockage, mémoire utilisée)

- La théorie de complexité étudie l'efficacité des algorithmes

Théorie de la complexité

- La théorie de la complexité vise à répondre aux besoins d'efficacité des algorithmes(programme)
- Elle permet :
  - de classer les problèmes selon leur difficulté
  - de classer les algorithmes selon leur efficacité
  - de comparer les algorithmes résolvant un problème donné afin de faire un choix éclairé sans devoir les implémenter

Evaluation de la rapidité d'un algorithme

- On ne mesure pas la durée en heures, minutes, secondes,...
- cela impliquerait d'implémenter les algorithmes qu'on veut comparer
- de plus, ces mesures ne seraient pas pertinentes car le même algorithme sera plus rapide sur une machine plus puissante

- Au lieu de ca, on utilise des unités de temps abstraites proportionnelles au nombre d'opérations effectuées
- Au besoin, on pourra ensuite adapter ces quantités en fonction de la machine sur laquelle l'algorithme s'exécute

Calcul du temps d'exécution

Règles:

- Chaque instruction basique consomme une unité de temps (affectation d'une variable, comparaison,...)
- Chaque itération d'une boucle rajoute le nombre d'unités de temps consommées dans le corps de cette boucle
- Chaque appel de fonction rajoute le nombre d'unités de temps consommées dans cette fonction
- Pour avoir le nombre d'opérations effectuées par l'algorithme, on additionne le tout

Exemple:

Temps de calcul d'une fonction factorielle:

- Ecrire un algorithme qui permet de calculer la factorielle d'un entier n.
- Quelle est sa complexité?

Complexité algorithmique

La complexité d'un algorithme est une mesure de sa performance asymptoptique dans le pire cas

- que signifie asymtotique ?
- on s'intéresse à des données très grandes
- pourquoi ?
- les petites valeurs ne sont pas assez informatives
- Que signifie "dans le pire cas" ?
- on s'intéresse à la performance de l'algorithme dans les situations où le problème prend le plus de temps à résoudre
- pourquoi ?
- on veut être sûr que l'algorithme ne prendra jamais plus de temps que ce qu'on a estimé

Règles de calcul

- On calcule le temps d'exécution, mais on effectue les simplifications suivantes :
  - on oublie les constantes multiplicatives (elles valent 1)
  - on annule les constantes additives
  - on ne retient que les termes dominants
- Exemple (simplifications) Soit $g(n) = 4n^3 - 5n^2 + 2n + 3$ opérations
  - on remplace les constantes multiplicatives par 1 : $1n^3 - 1n^2 + 1n + 3$
  - on annule les constantes additives : $n^3 - n^2 + n + 0$
  - on garde le terme de plus haut degré : $n^3 + 0$
  - on a donc $g(n) = O(n^3)$.

Justification des simplifications

Les processeurs actuels effectuent plusieurs milliards d'opérations à la seconde

- une affectation requière 2 ou 4 unités de temps ne change donc pas grand-chose
- d'où le remplacement des constantes par des 1 pour les multiplications

- un nombre constant d'instructions est donc aussi négligeable par rapport à la croissance de la taille des données
- d'où l'annulation des constantes additives

- pour de grandes valeurs de n, le terme de plus haut degré l'emportera
- d'où l'annulation des termes inférieurs

Complexité de la fonction factorielle


```
int factorielle(n)
fact = 1;                O(1)
i = 2;                   O(1)
while (i <= n)                    O(n)
     fact = fact * i;     O(1)
     i = i + 1;           O(1)
return fact              O(1)
```

Complexité de la fonction = O(1) + O(n) * O(1) + O(1) = O(n)

Classes de complexité

- On peut ranger les fonctions équivalentes dans la même classe
- Deux algorithmes appartenant à la meme classe sont considérés de meme complexité
- Les classes de complexité les plus fréquentes (par ordre croissant en termes de O(.))

Complexité	Classe
$O(1)$	constant
$O(logn)$	logarithmique
$O(n)$	linéaire
$O(nlogn)$	sous-quadratique
$O(n^2)$	quadratique
$O(n^3)$	cubique
$O(2^n)$	exponentiel
$O(n!)$	factorielle

Exercices

Récursivité