Topic 5
Introduction to Complexity of An Algorithm

TMA1201 Discrete Structures & Probability
Faculty of Computing & Informatics
Multimedia University

What you will learn in this lecture:

- What is an algorithm?
- Why do we need to analyze an algorithm?
- Introduction to growth function
- Introduction to complexity of algorithm
- Big-Oh, Big-Omega, and Big-Theta Notation
- Mathematical approach
- Analysis of algorithm

Algorithms

- Algorithms are sequences of steps defined, developed, and used to solve a problem.
- Examples of its usage:
  - Generating the orderings of a finite set
  - Searching a list
  - Sorting the terms of a sequence
  - Finding shortest path in a network
- One important consideration concerning algorithm is its computational complexity.
- **Complexity of an algorithm** refers to the amount of time or space needed to execute a given algorithm by:
  - Time efficiency: how fast an algorithm runs.
  - Space efficiency: the space an algorithm requires.

Big-Oh, Big-Omega, Big-Theta Notation

- In computer science, O-, Ω-, and Θ- notations are introduced to analyze the efficiency of algorithms.
- The notations provide approximations that make it easy to evaluate large-scale differences in algorithm efficiency, while ignoring differences of a constant factor and differences that occur only for small sets of input data.

  - O - notation
  - Ω - notation
  - Θ - notation

This is read as "**Big-Oh**" notation.

This is read as "**Big-Omega**" notation.

This is read as "**Big-Theta**" notation.

O - Notation

Definition:

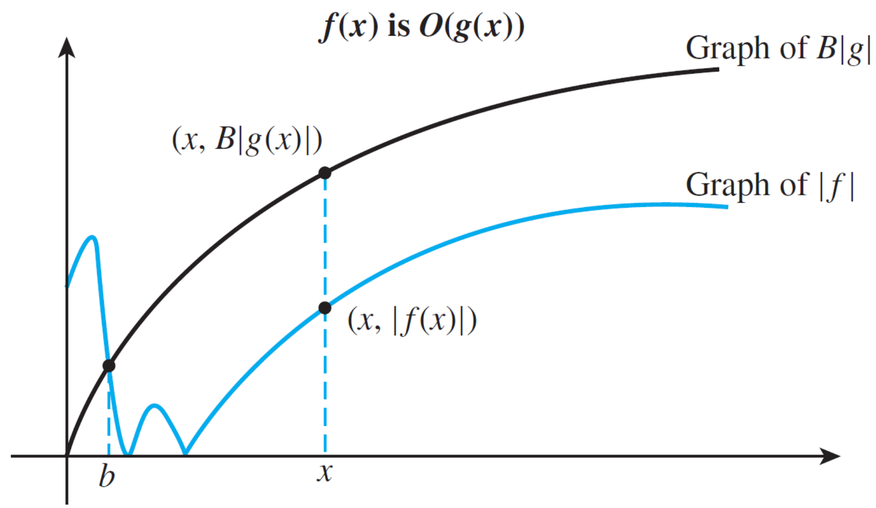
Let f and g be functions from a set of integers or the set of real numbers to the set of real numbers.

We say that **$f(n)$ is O($g(n)$)**, if there are constants B and b such that,

$$|f(n)| \leq B\,|g(n)|$$

whenever $n > b$.

The constants ***B*** and ***b*** can be viewed as witnesses to the relationship of $f(x)$ to O($g(x)$) as shown in the graph.



Example 1

Show that $f(n) = 3n^2$ is $O(n^2)$.

Solution:

$|f(n)| = |3n^2| = 3|n^2| \leq \mathbf{3}\ |n^2|$ when $n > 0$

Hence $f(n) = 3n^2$ is $O(n^2)$.

You could select a value larger than 3 as coefficient for $|n^2|$ according to the definition, but it is sufficient to define with the smallest value.

Example 2

Show that $f(n) = 2n^7 + 10n^2 + 5$ is $O(n^7)$.

Solution:

We consider $|f(n)| = |2n^7 + 10n^2 + 5|$

$$\leq |2n^7 + 15n^7| \quad \text{since } 15n^7 \geq 10n^2 + 5 \quad when \quad n > 1$$

$$\therefore |f(n)| \leq 17|n^7| \quad when \quad n > 1$$

Hence $f(n)$ is $O(n^7)$

Ω - Notation

Definition:

Let f and g be functions from a set of integers or the set of real numbers to the set of real numbers.

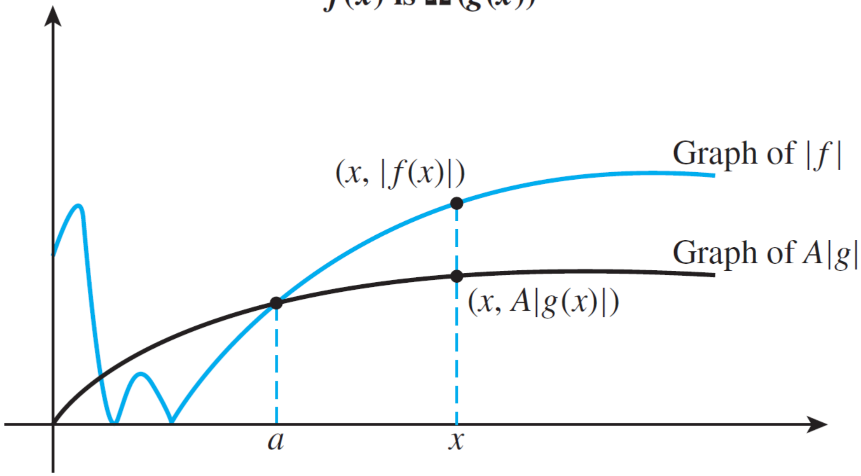
We say that $\boldsymbol{f(n)}$ **is** $\boldsymbol{\Omega(g(n))}$ if there are constants A and a such that,

$$|f(n)| \geq A\ |g(n)|$$

whenever $n > a$.

The constants A and a can be viewed as witnesses to the relationship of $f(x)$ to $\Omega(g(x))$ as shown in the graph.

$\boldsymbol{f(x)}$ **is** $\boldsymbol{\Omega(g(x))}$



Example 3

Show that $f(n) = 2n^7 + 10n^2 + 5$ is $\Omega(n^7)$.

Solution:

Consider $|f(n)| = |2n^7 + 10n^2 + 5|$

$$\geq |2n^7 + 0| \quad \text{since } 10n^2 + 5 > 0 \quad \textit{when} \quad n > 0$$

$$\therefore |f(n)| \geq 2|n^7| \quad \textit{when} \quad n > 0$$

Hence $f(n)$ is $\Omega(n^7)$

Θ - Notation

Definition:

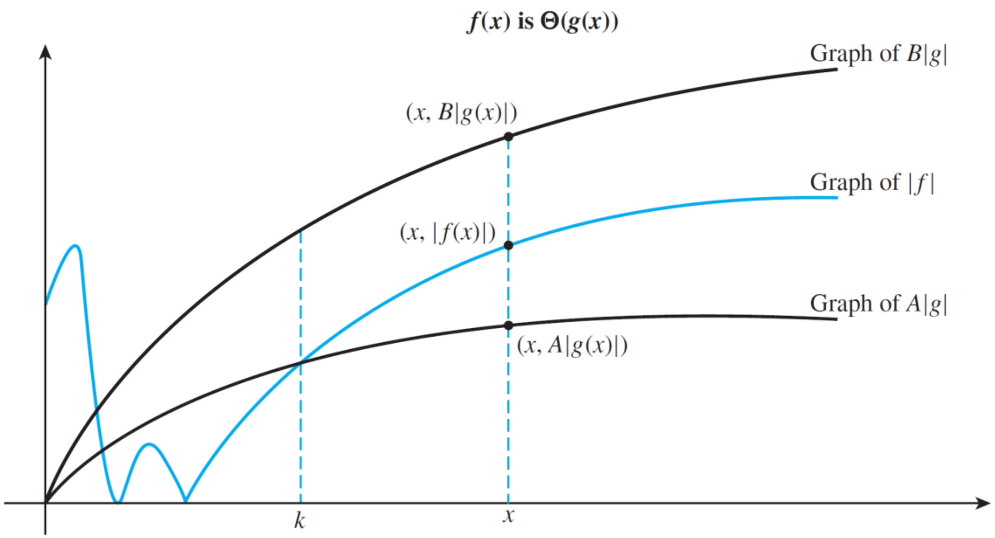
Let *f* and *g* be functions from a set of integers or the set of real numbers to the set of real numbers.

We say that ***f* (*n*) is Θ(*g*(*n*))** if $f(n)$ is $\Omega(g(n))$ and $f(n)$ is $O(g(n))$.

Note that *f (x)* is *Θ(g(x))* iff there are real numbers *A* and *B* and a positive real number *k* such that

$$A|g(x)| \leq f(x) \leq B|g(x)|$$

whenever $x > k$, as shown in the graph.



Example 4

Show that $f(n) = 3n^3 + 3n\lg n$ is $\Theta(n^3)$.

Solution:

By definition, $f(n)$ is $\Theta(g(n))$ if $f(n)$ is $\Omega(g(n))$ and $f(n)$ is $O(g(n))$.

We need to show that $f(n)$ is $\Omega(n^3)$ and $f(n)$ is $O(n^3)$.

$|f(n)| = |3n^3 + 3nlgn| \le |3n^3 + 3n^3|$ since $n^2 > \lg n$ when $n > 1$.

$\therefore$ $|f(n)| \le 6|n^3|$ when n > 1 or $f(n)$ is $O(n^3)$

$|3n^3 + 3nlgn| \ge 3|n^3+0|$ since $3n\lg n > 0$ when $n > 1$

Hence $|f(n)| \ge 3|n^3|$ when $n > 1$ or $f(n)$ is $\Omega(n^3)$

Since $f(n)$ is $O(n^3)$ and $f(n)$ is $\Omega(n^3)$, $f(n)$ is $\Theta(n^3)$.

Big-Oh, Big-Omega, Big-Theta Notation

- When $f(x)$ is $O(g(x))$, we have an **upper bound**, in terms of $g(x)$, for the size of $f(x)$ for large values of x.
- When $f(x)$ is $\Omega(g(x))$, we have a **lower bound**, in terms of $g(x)$, for the size of $f(x)$ for large values of x.
- When $f(x)$ is $\Theta(g(x))$, we have **both upper bound and lower bound**, in terms of $g(x)$, for the size of $f(x)$ for large values of x.

Mathematical Approach

- Given some function $f(n)$ that describes an algorithm, you have to find another function $g(n)$ by selecting some parts of the function $f(n)$ that decides the growth of function $f(n)$.

- Mathematically, this is done by either:
  - removing the residual terms from a polynomial function (normally these are the constant and/or the low order terms in the polynomial),
  - maximizing all the terms in a polynomial function.

Analysis of An Algorithm

- In computer science, a correct algorithm might not be efficient if the time or space taken for its execution is too large.
- Analysis of an algorithm refers to the process of deriving the estimates of the complexity (or growth function) for the time and/or space needed to execute an algorithm.
- An analysis table is usually used to determine the growth function (in time) for predicting the execution time of an algorithm and deriving the complexity of the algorithm.

Analysis of An Algorithm

Commonly used terminology for the complexity of algorithms.

Complexity	*Terminology*
$\Theta(1)$	Constant complexity
$\Theta(\log n)$	Logarithmic complexity
$\Theta(\log(\log n))$	Log log complexity
$\Theta(n)$	Linear complexity
$\Theta(n\log n)$	Log linear complexity
$\Theta(n^2)$	Quadratic complexity
$\Theta(n^m)$ where integer m>1	Polynomial complexity
$\Theta(c^n)$ where integer n>1	Exponential complexity
$\Theta(n!)$	Factorial complexity

Simple Algorithm Analysis

Example 5:

1	System.in.readln(x);	c_1
2	System.out.writeln(x);	c_2

Line number

Execution time (the constant C is 1). Input size is n.

Solution: Let f(n) denotes the time complexity to the code fragment

$$f(n) = c_1 + c_2 \Rightarrow f(n) \text{ is } \Theta(n^0)$$

Example 6: Find the theta notation for the time complexity of the statement "x:=5" being executed in the following code fragment

1	for (int i = 1; i <= 10; i++)	
	{	
2	x := 5;	$10c_1$
	}	

Execution time with constant 10 when the Input size is n.

Solution: $f(n) = 10c_1 \Rightarrow f(n) \text{ is } \Theta(n^0)$

What are the values of *O* and Ω of each *f*(n) above?

Example 7

Find the theta notation for time complexity of the statement "System.in.readln(x)" being executed in the following code fragment

	int sum;	
1	sum := 0;	
2	for (int i = 1; i <= n; i++)	
	{	
3	System.in.readln(x);	n*c
4	sum := sum + x;	
	}	
5	System.out.println(sum)	

Solution: $f(n) = nc \Rightarrow f(n)$ is $\Theta(n)$

Example 8

Find the theta notation in terms of n for the time complexity of the statement "sum := sum + F[i,j]" being executed in the following code fragment.

```
1    for (int i = 1; i <= n; i++)
2      for (int j = 1; j <= i; j++)
3          sum := sum + F[i,j];
```

Solution:

-First i set to 1, j runs from 1 to 1, the statement of line 3 is executed one time.
-Then i set to 2, j runs from 1 to 2, the statement of line 3 is executed 2 times, and so on. The time complexity of line 3 being executed is

f(n) = (1 + 2 +… + n)c = $\frac{cn(n+1)}{2}$ where c is the time required to executed line 3.

Hence f(n) is $\Theta(n^2)$

Summary

We have learnt the following concepts related to the complexity of an algorithm:

- Big-Oh
- Big-Omega
- Big-Theta
- Mathematical approach to estimate the complexity of an algorithm.
- Simple algorithm analysis using an analysis table.

Exercise 1

Show that $f(n) = 2n^2$ +nlg(n) is $\Theta(n^2)$.

Solution:

Exercise 2

Find the theta notation in terms of n for the time complexity of the statement “sumsq= sumsq+ i*i” being executed in the following code fragment.

```
sumsq = 0;
i = 0;
(while i < n) {
        sumsq= sumsq+ i*i;
        i = i + 1;
}
```

Solution: