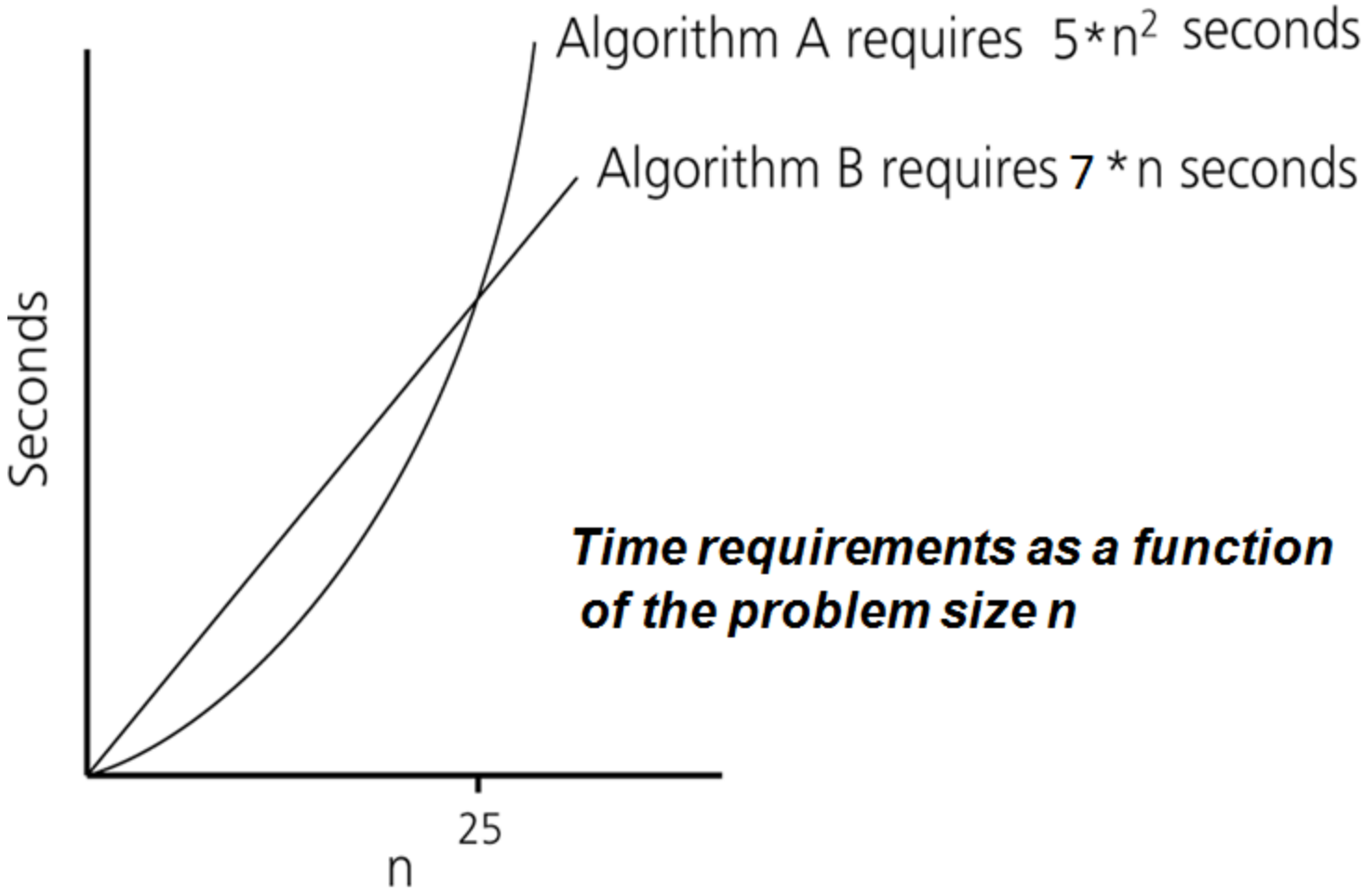
Concepts of Algorithms
CS-211
Chapter 2
Algorithm Growth Rates

Dr. Fethi Fkih
College of Computer
Qassim University.

Algorithm Growth Rates

- We measure an algorithm's time requirement as a function of the *problem size*.
  - Problem size depends on the application: e.g. number of elements in a list for a sorting algorithm
  - So, for instance, we say that (if the problem size is n)
  - Algorithm A requires **5*n^2** time units to solve a problem of size n.
  - Algorithm B requires **7*n** time units to solve a problem of size n.
- The most important thing to learn is how quickly the algorithm's time requirement grows as a function of the problem size.
  - Algorithm A requires time proportional to **n^2**.
  - Algorithm B requires time proportional to **n**.
- An algorithm's proportional time requirement is known as ***growth rate***.
- We can compare the efficiency of two algorithms by comparing their growth rates.

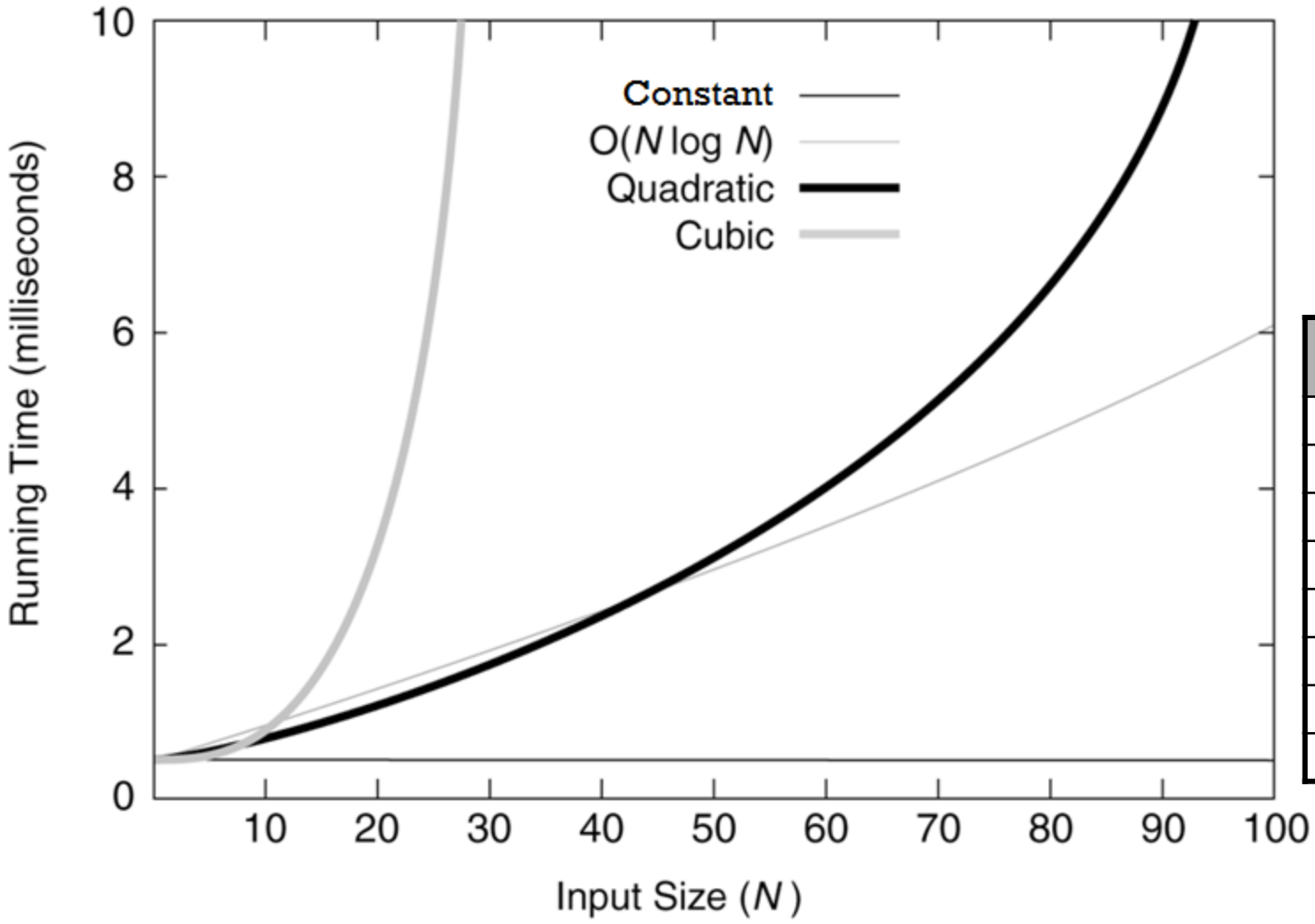
Algorithm Growth Rates (cont.)



Common Growth Rates

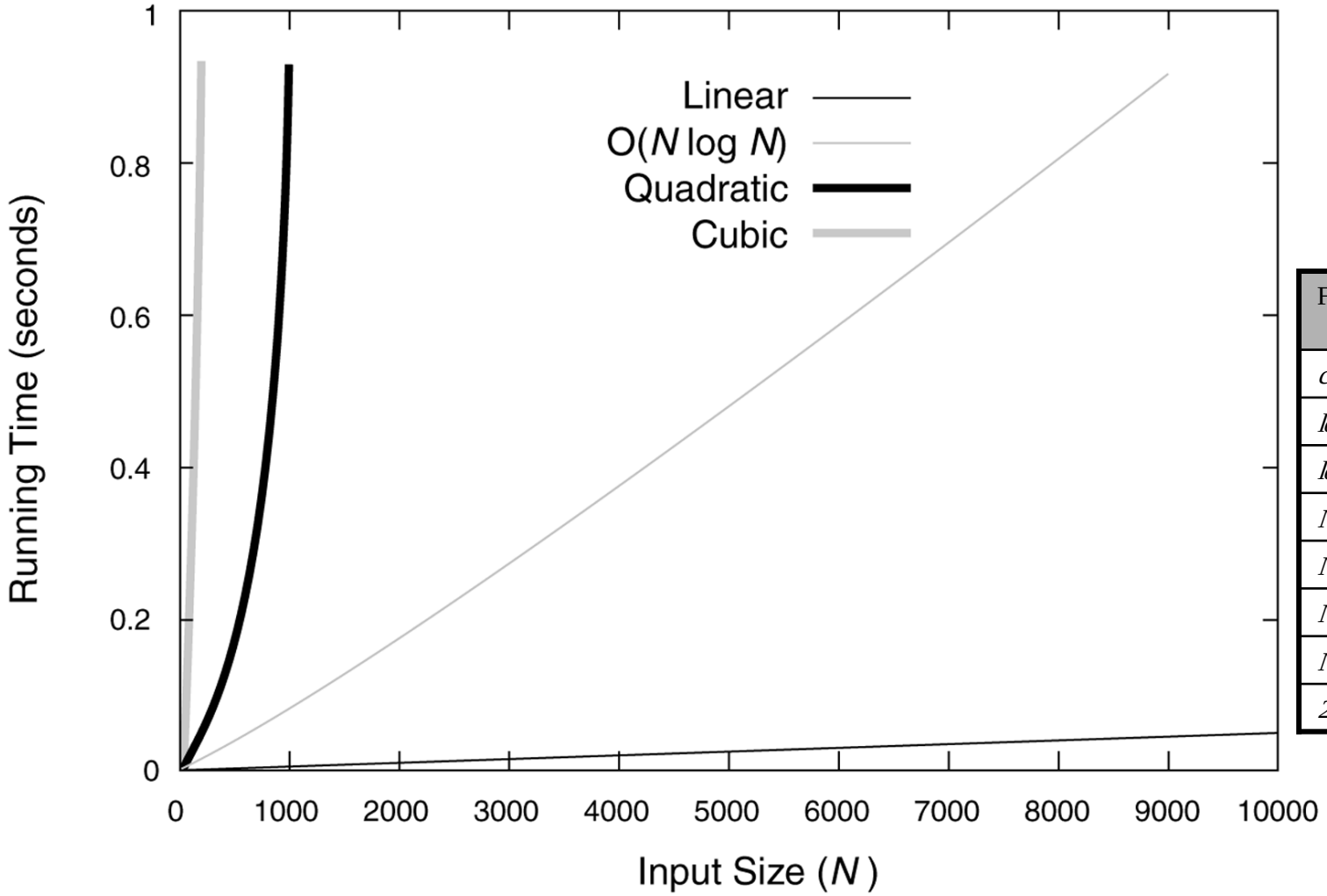
Function	Growth Rate Name
c	Constant
$\log N$	Logarithmic
$\log^2 N$	Log-squared
N	Linear
$N \log N$	
N^2	Quadratic
N^3	Cubic
2^N	Exponential

Running times for small inputs



Function	Growth Rate Name
c	Constant
$\log N$	Logarithmic
$\log^2 N$	Log-squared
N	Linear
$N \log N$	
N^2	Quadratic
N^3	Cubic
2^N	Exponential

Running times for moderate inputs



Function	Growth Rate Name
c	Constant
$\log N$	Logarithmic
$\log^2 N$	Log-squared
N	Linear
$N \log N$	
N^2	Quadratic
N^3	Cubic
2^N	Exponential

Order-of-Magnitude Analysis and Big O Notation

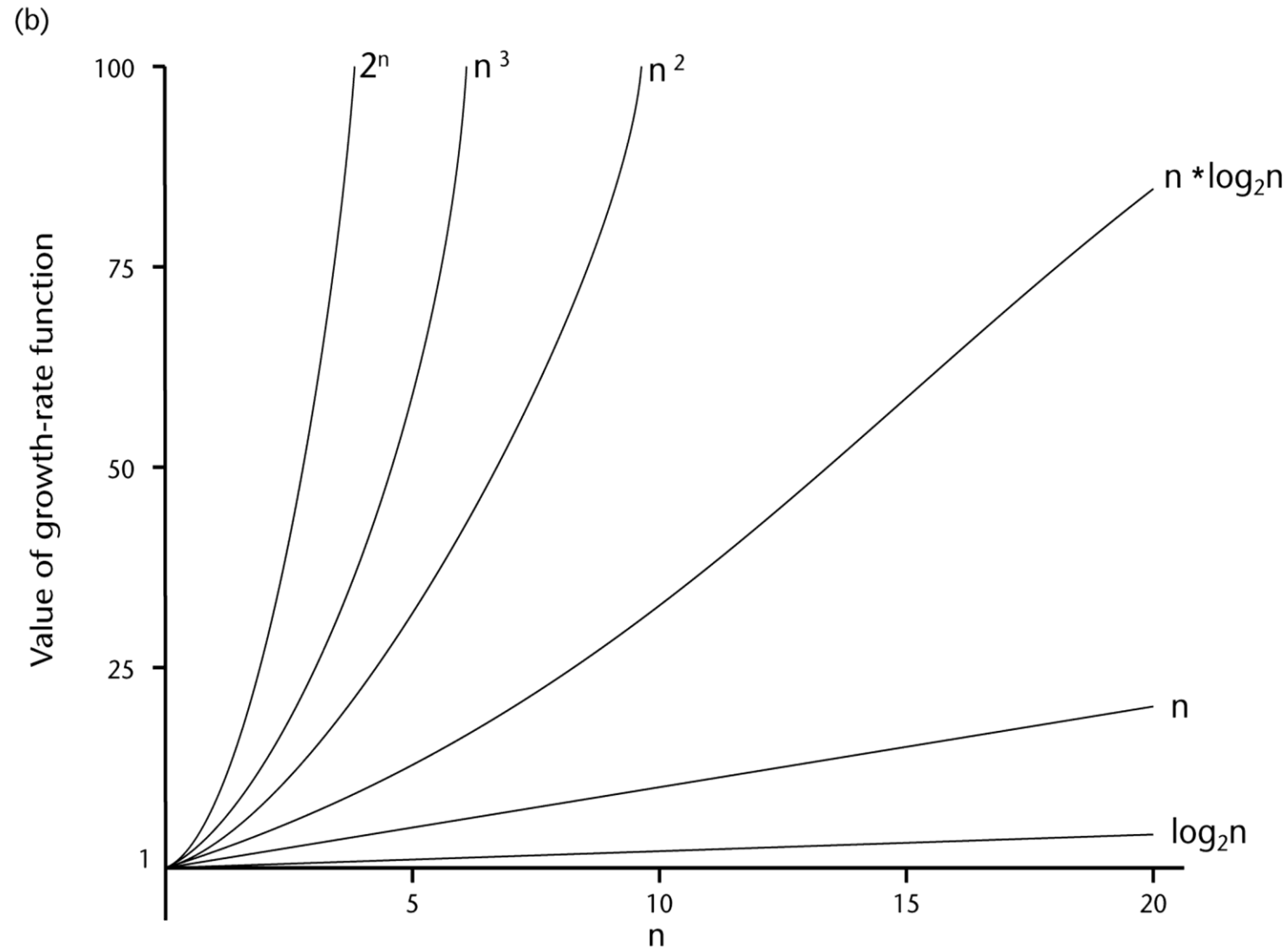
- If *Algorithm A requires time proportional to f(n),* Algorithm A is said to be **order f(n),** and it is denoted as **O(f(n)).**
- The **function f(n)** is called the algorithm's **growth-rate function**.
- Since the capital O is used in the notation, this notation is called the **Big O notation**.
- If Algorithm A requires time proportional to $\mathbf{n^2}$, it is $\mathbf{O(n^2)}$.
- If Algorithm A requires time proportional to **n**, it is **O(n).**

A Comparison of Growth-Rate Functions

(a)

	n					
Function	10	100	1,000	10,000	100,000	1,000,000
1	1	1	1	1	1	1
$\log_2 n$	3	6	9	13	16	19
n	10	10^2	10^3	10^4	10^5	10^6
$n * \log_2 n$	30	664	9,965	10^5	10^6	10^7
n^2	10^2	10^4	10^6	10^8	10^{10}	10^{12}
n^3	10^3	10^6	10^9	10^{12}	10^{15}	10^{18}
2^n	10^3	10^{30}	10^{301}	$10^{3,010}$	$10^{30,103}$	$10^{301,030}$

A Comparison of Growth-Rate Functions (cont.)



Growth-Rate Functions

O(1) Time requirement is **constant**, and it is independent of the problem's size.

O($\log_2 n$) Time requirement for a **logarithmic** algorithm increases slowly as the problem size increases.

O(n) Time requirement for a **linear** algorithm increases directly with the size of the problem.

O($n*\log_2 n$) Time requirement for a **$n*\log_2 n$** algorithm increases more rapidly than a linear algorithm.

O(n^2) Time requirement for a **quadratic** algorithm increases rapidly with the size of the problem.

O(n^3) Time requirement for a c**ubic** algorithm increases more rapidly with the size of the problem than the time requirement for a quadratic algorithm.

O(2^n) As the size of the problem increases, the time requirement for an **exponential** algorithm increases too rapidly to be practical.

Growth-Rate Functions

- If an algorithm takes 1 second to run with the problem size 8, what is the time requirement (approximately) for that algorithm with the problem size 16?
- If its order is:

O(1) ➔ T(n) = 1 second

O($\log_2 n$) ➔ T(n) = (1*$\log_2 16$) / $\log_2 8$ = 4/3 seconds

O(n) ➔ T(n) = (1*16) / 8 = 2 seconds

O(n*$\log_2 n$) ➔ T(n) = (1*16*$\log_2 16$) / 8*$\log_2 8$ = 8/3 seconds

O(n^2) ➔ T(n) = (1*16^2) / 8^2 = 4 seconds

O(n^3) ➔ T(n) = (1*16^3) / 8^3 = 8 seconds

O(2^n) ➔ T(n) = (1*2^{16}) / 2^8 = 2^8 seconds = 256 seconds

END