CS1DM3 DISCRETE MATH

CHAPTER 3 - ALGORITHMS

3.1 ALGORITHMS

INTRO

DEFINITIONS

An ***algorithm*** is a is a finite sequence of mathematically rigorous instructions, typically used to solve a class of specific problems or to perform a computation.

Pseudocode is a description of the steps in an algorithm intended for human reading rather than machine control.

To gain insight into how an algorithm works it is useful to construct a ***trace*** that shows its steps when given specific input

PROPERTIES OF ALGORITHMS

▶ **Input**	An algorithm has input values from a *(possibly empty)* specified set.
▶ **Output**	From each set of input values an algorithm produces output values, which are the solution to the problem, from a *(possibly empty)* specified set.
▶ **Definiteness**	The steps of an algorithm must be defined **precisely**.
▶ **Correctness**	An algorithm should produce the **correct** output values for each set of input values.
▶ **Finiteness**	An algorithm should produce the desired output **after a finite** (perhaps large) **number of steps** for any input in the set.
▶ **Effectiveness**	It must be possible to perform each step of an algorithm **exactly** and in a **finite amount of time**.
▶ **Generality**.	The procedure should be **applicable for all problems** of the desired form, not just for a particular set of input values

sequentially checks each element of the list until a match is found or the whole list has been searched

ALGORITHM 2 The Linear Search Algorithm.

procedure *linear search*(x: integer, $a_1, a_2, \ldots, a_n$: distinct integers)
$i := 1$
while ($i \le n$ and $x \ne a_i$)
    $i := i + 1$
if $i \le n$ **then** $location := i$
else $location := 0$
return $location$ {$location$ is the subscript of the term that equals x, or is 0 if x is not found}

Worst-case	$O(n)$
Best-case	$O(1)$
Worst-case (Space)	$O(1)$

Find 42:

10	25	37	37	59	63	59	63	42

Explanation of time complexity:
p232, 3.3 Ex1&4

Check middle element of the sorted array.
If not found, check **middle element of the upper/lower half** that target must be within.
Repeat until target value is found – or determined not in array.

ALGORITHM 3 The Binary Search Algorithm.

procedure *binary search* (x: integer, $a_1, a_2, \ldots, a_n$: increasing integers)
$i := 1$ {i is left endpoint of search interval}
$j := n$ {j is right endpoint of search interval}
while $i < j$
    $m := \lfloor (i + j)/2 \rfloor$
    if $x > a_m$ **then** $i := m + 1$
    else $j := m$
if $x = a_i$ **then** *location* $:= i$
else *location* $:= 0$
return *location* {*location* is the subscript i of the term a_i equal to x, or 0 if x is not found}

Worst-case $O(\log n)$

Best-case $O(1)$

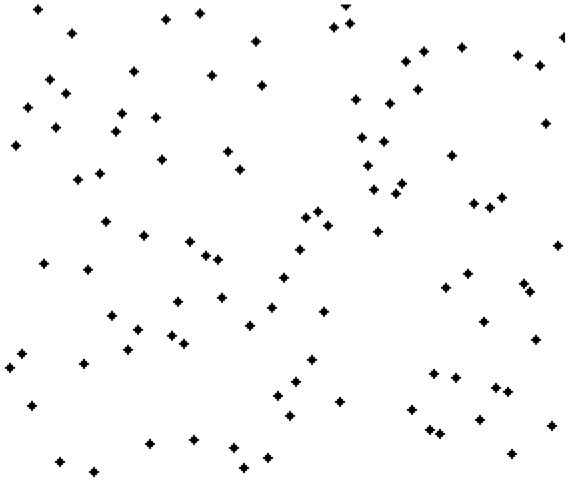
Worst-case (Space) $O(1)$

Find 42:

10	25	37	37	59	63	59	63	42

Explanation: p232, 3.3 Ex2

Compare each element with next one, swapping if needed. Repeat.



ALGORITHM 4 The Bubble Sort.

procedure *bubblesort*($a_1, \ldots, a_n$: real numbers with $n \geq 2$)
for $i := 1$ **to** $n - 1$
    for $j := 1$ **to** $n - i$
        if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}
$\{a_1, \ldots, a_n$ is in increasing order$\}$

6 5 3 1 8 7 2 4

Worst-case	$O(n^2)$ comparisons $O(n^2)$ swaps
Best-case	$O(n)$ comparisons $O(1)$ swaps
Worst-case (Space)	$O(n)$ total $O(1)$ auxillory

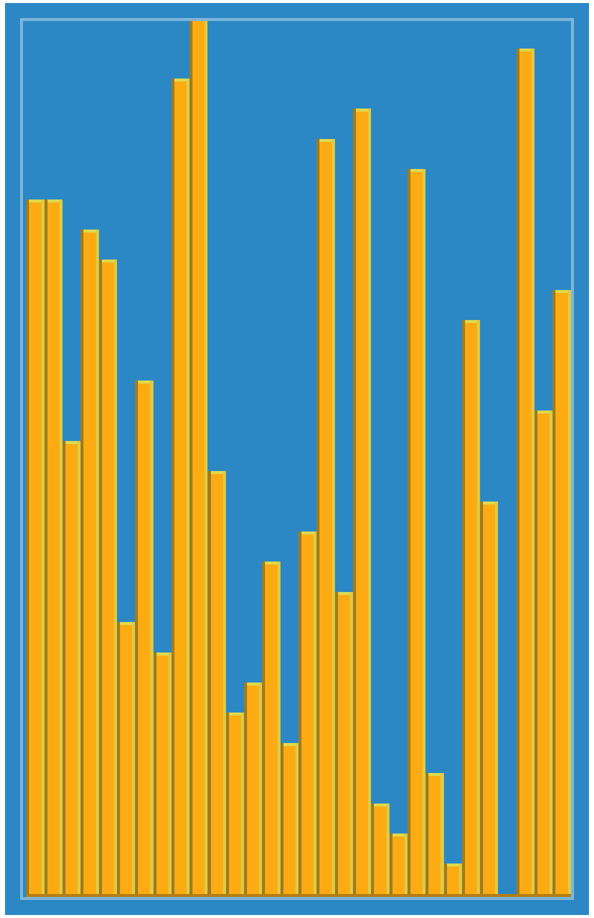
Explanation: p232, 3.3 Ex3&5

Removes each element from array,
Find its appropriate location within the sorted list,
insert it there. Repeat.

ALGORITHM 5 The Insertion Sort.

procedure *insertion sort*($a_1, a_2, \ldots, a_n$: real numbers with $n \geq 2$)
for $j := 2$ **to** n
    $i := 1$
    while $a_j > a_i$
        $i := i + 1$
    $m := a_j$
    for $k := 0$ **to** $j - i - 1$
        $a_{j-k} := a_{j-k-1}$
    $a_i := m$
$\{a_1, \ldots, a_n$ is in increasing order$\}$

6 5 3 1 8 7 2 4



Worst-case	$O(n^2)$ comparisons & swaps
Best-case	$O(n)$ comparisons $O(1)$ swaps
Worst-case (Space)	$O(n)$ total $O(1)$ auxillory

Explanation: p232, 3.3 Ex6

PRACTICE PROBLEM

Identify **Big-O time complexity** of:

ALGORITHM 6 Naive String Matcher.

procedure *string match* (n, m: positive integers, $m \leq n, t_1, t_2, \ldots, t_n, p_1, p_2, \ldots, p_m$: characters)
    for $s := 0$ **to** $n - m$
        $j := 1$
        while ($j \leq m$ and $t_{s+j} = p_j$)
            $j := j + 1$
        if $j > m$ **then print** "s is a valid shift"

Check your answer by searching online or this wiki article:
[Naïve / Brute-Force String Matching algorithm]

3.2 GROWTH OF FUNCTIONS

BIG-O NOTATION

Sometimes written $f(x) = O(g(x))$. However, "=" does not represent a genuine equality.

DEFINITIONS

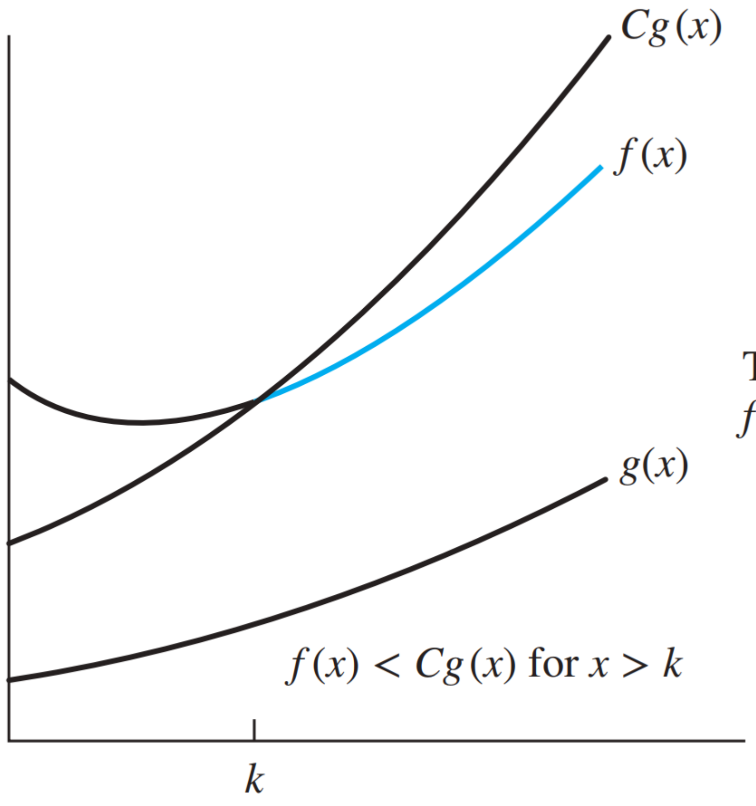
Let f and g be functions from the set of integers or real numbers to the set of real numbers.

We say that $f(x)$ is $\boxed{\boldsymbol{O(g(x))}}$, *read as "**big-oh**"*, if there are constants C and k, whenever $x > k$ such that:

$$\boxed{|f(x)| \leq C|g(x)|}$$

"witnesses"

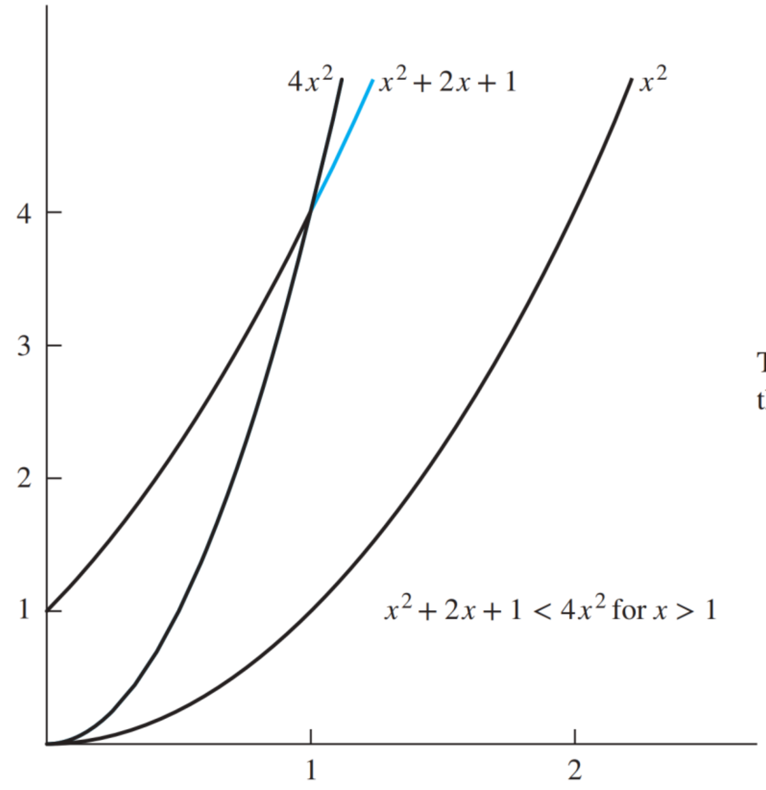
This says $f(x)$ grows **slower** than some **fixed multiple** of $g(x)$ as x grows without bound.



The part of the graph of $f(x)$ that satisfies $f(x) < Cg(x)$ is shown in color.

FIGURE 2 **The function $f(x)$ is $O(g(x))$.**

Note that in Example 1 we have two functions, $f(x) = x^2 + 2x + 1$ and $g(x) = x^2$, such that $f(x)$ is $O(g(x))$ and $g(x)$ is $O(f(x))$—the latter fact following from the inequality $x^2 \leq x^2 + 2x + 1$, which holds for all nonnegative real numbers x. We say that two functions



The part of the graph of $f(x) = x^2 + 2x + 1$ that satisfies $f(x) < 4x^2$ is shown in color.

FIGURE 1 **The function $x^2 + 2x + 1$ is $O(x^2)$.**

EXAMPLE

Show that $7x^2$ is $O(x^3)$.

Solution:

When $x > 7$, by multiplying both sides by x^2, we have $7x^2 < x^3$.

Take $C = 1$ and $k = 7$ as witnesses to establish:

$$\text{Whenever } x > 7 \text{ such that } |7x^2| \leq 1 \cdot |x^3|$$

Alternatively, when $x > 1$, we have $7x^2 < 7x^3$, so $C = 7$ and $k = 1$ are also witnesses.

BIG O COMPLEXITY

Time / Space used

TERRIBLE

$O(n!)$

$O(2^n)$

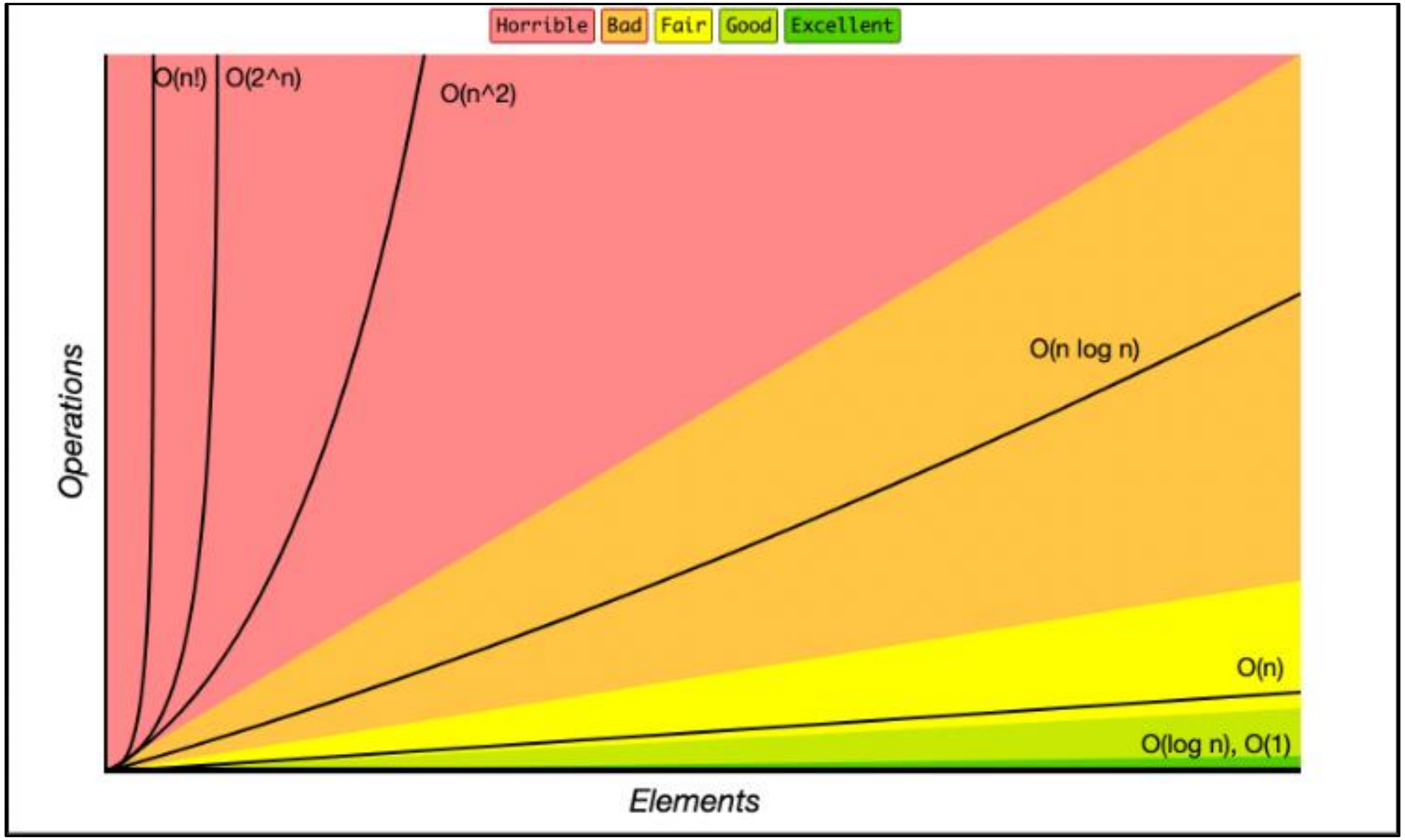
$O(n^2)$

$O(n \log n)$ Bad

$O(n)$ Fair

$O(\log n)$ Good

$O(1)$ Best



$O(n!)$

$O(2^n)$ **TERRIBLE**

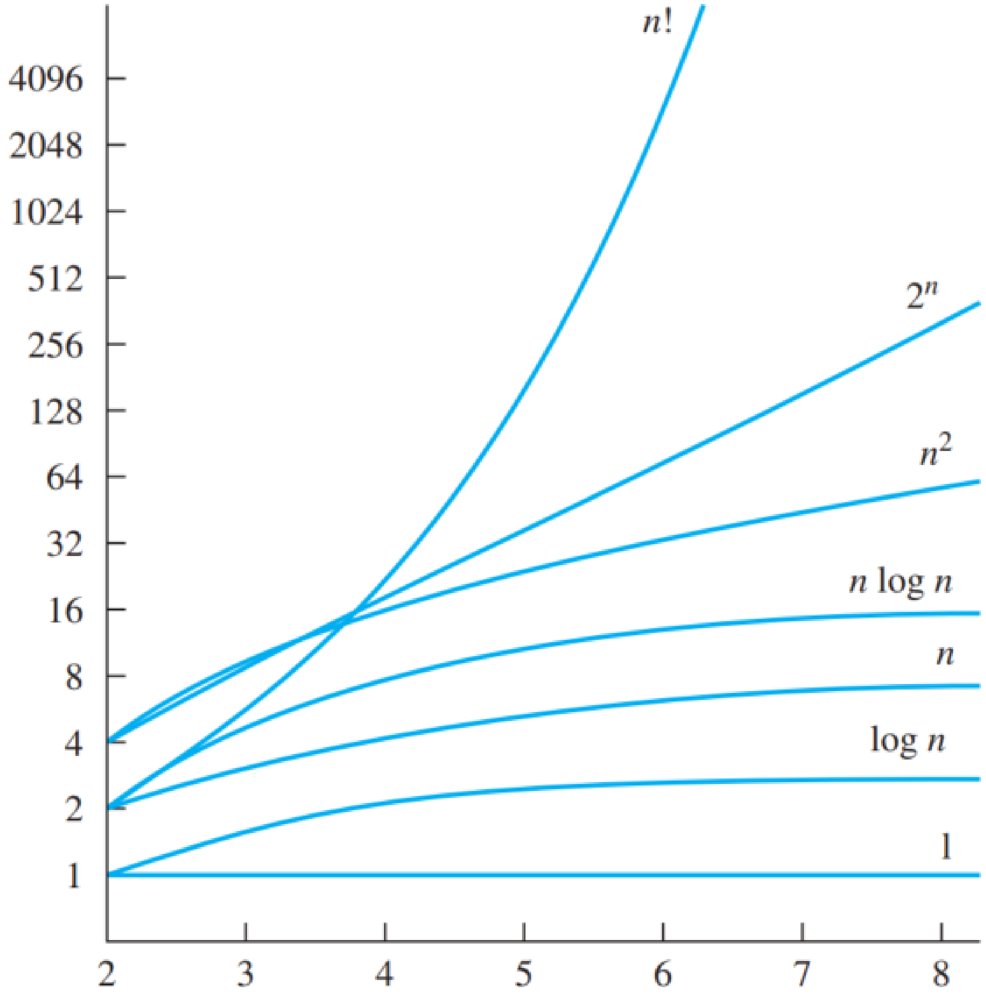
$O(n^2)$

$O(n \log n)$ **Bad**

$O(n)$ Fair

$O(\log n)$ Good

$O(1)$ **Best**



BIG O COMPLEXITY

DEFINITIONS

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

We say that $f(x)$ is $\boldsymbol{\Omega(g(x))}$, *read as "**big-omega**",*

if there are constants C and k, with $C > 0$ such that:

$$|f(x)| \geq C|g(x)|$$

$f(x) = \Omega(g(x))$ if and only if $g(x) = O(f(x))$

$$C_1|g(x)| \leq |f(x)| \leq C_2|g(x)|$$

DEFINITIONS

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

We say that $f(x)$ is $\boxed{\Theta(g(x))}$, *read as "**big-theta**"*,

$$\boxed{\text{if } g(x) = O(f(x)) \text{ and } f(x) = \Omega(g(x))}.$$

If $f(x) = \Theta(g(x))$, we say that $\boxed{f(x) \text{ is of order } g(x)}$,
and that $\boxed{f(x) \text{ and } g(x) \text{ are of the same order}}$.

From the definition of big-O notation, there are constants C_1, C_2, k_1, and k_2 such that

$|f_1(x)| \leq C_1|g_1(x)|$ when $x > k_1$, and $|f_2(x)| \leq C_2|g_2(x)|$ when $x > k_2$.

To estimate the sum of $f_1(x)$ and $f_2(x)$, note that

$$\begin{aligned} |(f_1 + f_2)(x)| &= |f_1(x) + f_2(x)| \\ &\leq |f_1(x)| + |f_2(x)| \quad \text{using the triangle inequality } |a + b| \leq |a| + |b|. \end{aligned}$$

When x is greater than both k_1 and k_2, it follows from the inequalities for $|f_1(x)|$ and $|f_2(x)|$ that

$$\begin{aligned} |f_1(x)| + |f_2(x)| &\leq C_1|g_1(x)| + C_2|g_2(x)| \\ &\leq C_1|g(x)| + C_2|g(x)| \\ &= (C_1 + C_2)|g(x)| \\ &= C|g(x)|, \end{aligned}$$

where $C = C_1 + C_2$ and $g(x) = \max(|g_1(x)|, |g_2(x)|)$. [Here $\max(a, b)$ denotes the maximum, or larger, of a and b.]

This inequality shows that $|(f_1 + f_2)(x)| \leq C|g(x)|$ whenever $x > k$, where $k = \max(k_1, k_2)$.

Suppose $f_1(x) = O\big(g_1(x)\big),\ f_2(x) = O(g_2(x))$. Then

$$\boldsymbol{(f_1 + f_2)(x) = O(g(x))}$$

where $\forall x, g(x) =\ (\max(|g_1(x)|, |g_2(x)|)$.

Suppose that $f_1(x)$ and $f_1(x)$ are both $O(g(x))$. Then

$$\boldsymbol{(f_1 + f_2)(x) = O(g(x))}$$

Suppose that $f_1(x) = O\big(g_1(x)\big),\ f_2(x) = O(g_2(x))$. Then

$$\boldsymbol{(f_1 f_2)(x) \text{ is } O(g_1(x) g_2(x)).}$$

EXAMPLE

Give a big-O estimate for $f(x) = (x + 1)\log(x^2 + 1) + 3x^2$.

Solution: First, a big-O estimate for $(x + 1)\log(x^2 + 1)$ will be found. Note that $(x + 1)$ is $O(x)$. Furthermore, $x^2 + 1 \leq 2x^2$ when $x > 1$. Hence,

$$\log(x^2 + 1) \leq \log(2x^2) = \log 2 + \log x^2 = \log 2 + 2\log x \leq 3\log x,$$

if $x > 2$. This shows that $\log(x^2 + 1)$ is $O(\log x)$.

From Theorem 3 it follows that $(x + 1)\log(x^2 + 1)$ is $O(x \log x)$. Because $3x^2$ is $O(x^2)$, Theorem 2 tells us that $f(x)$ is $O(\max(x \log x, x^2))$. Because $x \log x \leq x^2$, for $x > 1$, it follows that $f(x)$ is $O(x^2)$. ◀

3.3 COMPLEXITY OF FUNCTIONS

BIG-O NOTATION

TABLE 1 Commonly Used Terminology for the Complexity of Algorithms.

Complexity	*Terminology*
$\Theta(1)$	Constant complexity
$\Theta(\log n)$	Logarithmic complexity
$\Theta(n)$	Linear complexity
$\Theta(n \log n)$	Linearithmic complexity
$\Theta(n^b)$	Polynomial complexity
$\Theta(b^n)$, where $b > 1$	Exponential complexity
$\Theta(n!)$	Factorial complexity

PRACTICE PROBLEMS

Explain **Best**, **Worst**, and **Average** *Big-Oh* ***time complexities*** of these Algorithms:

- ❑Find max element in a finite set of integers (3.1Alg1, Solution: 3.3 Ex1)
- ❑Linear Search (3.1Alg2, Solution: 3.3Ex2, Ex4)
- ❑Binary Search (3.1Alg3, Solution: 3.3Ex3)
- ❑Bubble Sort (3.1Alg4, Solution: 3.3Ex5)
- ❑Insertion Sort (3.1Alg5, Solution 3.3Ex6)
- ❑**Matrix Multiplication (3.3Alg1**, Solution 3.3Ex7)
- ❑**Brute-Force Alg for Closest Pair of Points (3.3Alg3**, Solution 3.3Ex8)

DEFINITIONS

- algorithmic paradigm
- tractable,

P VS NP PROBLEM