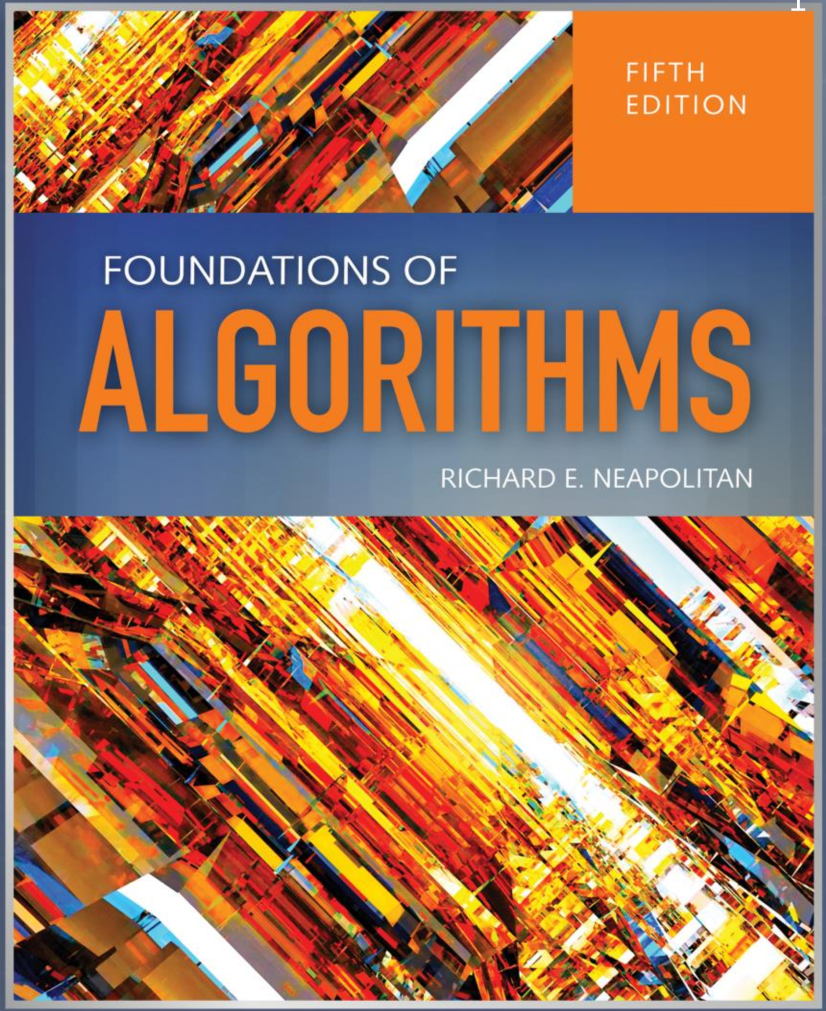


Algorithms:
Efficiency, Analysis, and Order

Chapter 1

Objectives

- Analyze techniques for solving problems
- Define an algorithm
- Define growth rate of an algorithm as a function of input size
- Define Worst case, average case, and best case complexity analysis of algorithms
- Classify functions based on growth rate
- Define growth rates: Big O, Theta, and Omega

Methodology

- Approach to solving a problem
- Independent of Programming Language
- Independent of Style
- Sequential Search versus Binary Search
- Which technique results in the most efficient solution?

Problem?

- A question to which an answer is sought
- Parameters
  - Input to the problem
  - Instance: a specific assignment of values to the input parameters
- Algorithm:
  - Step-by-step procedure
  - Solves the Problem

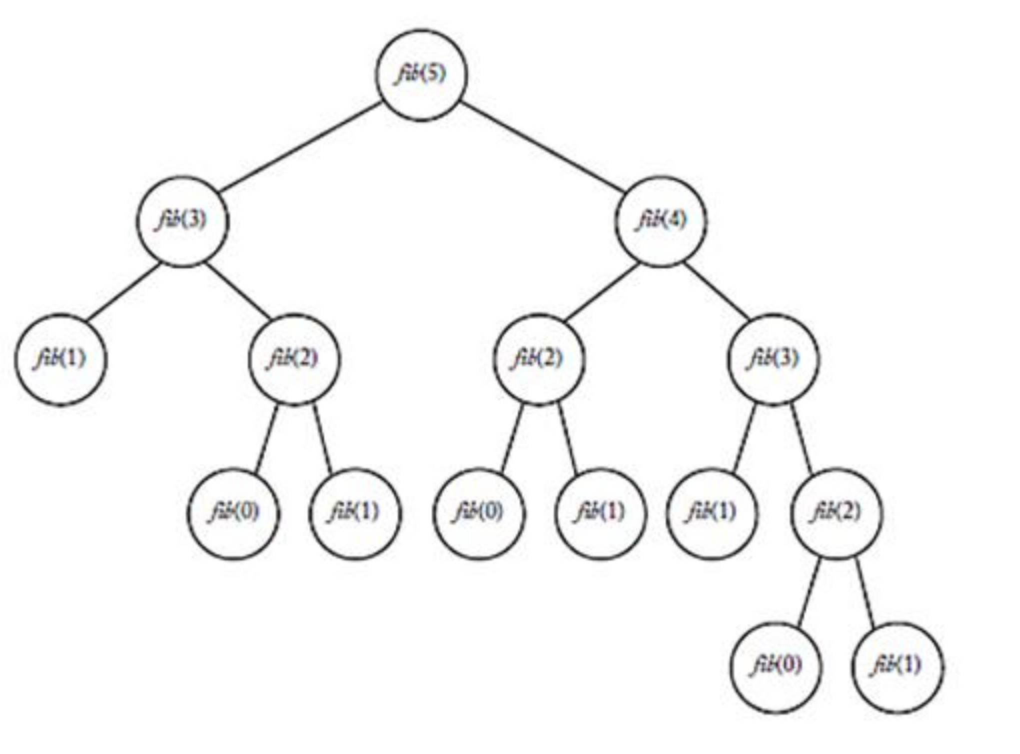
Sequential Search vs Binary Search – Worst Case

- Input Array S size n
- $X \notin S$
- Sequential Search: n operations
- Binary Search: lg n + 1 operations

Fibonacci: Iterative vs Recursive

- $Fib_0 = 0$
- $Fib_1 = 1$
- $Fib_n = Fib_{n-1} + Fib_{n-2}$
- Calculate the nth Fibonacci Term:
  - Recursive calculates $2^{n/2}$ terms
  - Iterative calculates n+1 terms

Recursion Tree for the 5th Fibonacci Term



Comparison of Recursive and Iterative Solutions

n	$n+1$	$2^{n/2}$	Execution Time Using Algorithm 1.7	Lower Bound on Execution Time Using Algorithm 1.6
40	41	1,048,576	41 ns*	1048 μs†
60	61	1.1×10^{9}	61 ns	1 s
80	81	1.1×10^{12}	81 ns	18 min
100	101	1.1×10^{15}	101 ns	13 days
120	121	1.2×10^{18}	121 ns	36 years
160	161	1.2×10^{24}	161 ns	3.8×10^{7} years
200	201	1.3×10^{30}	201 ns	4×10^{13} years

*1 ns $= 10^{-9}$ second.

†1 $\mu s = 10^{-6}$ second.

Complexity Analysis

- Define Basic Operation
- Count the number of times the basic operation executes for each value of the input size
- Maybe dependent on input size (sequential search)
- Every-case time complexity analysis

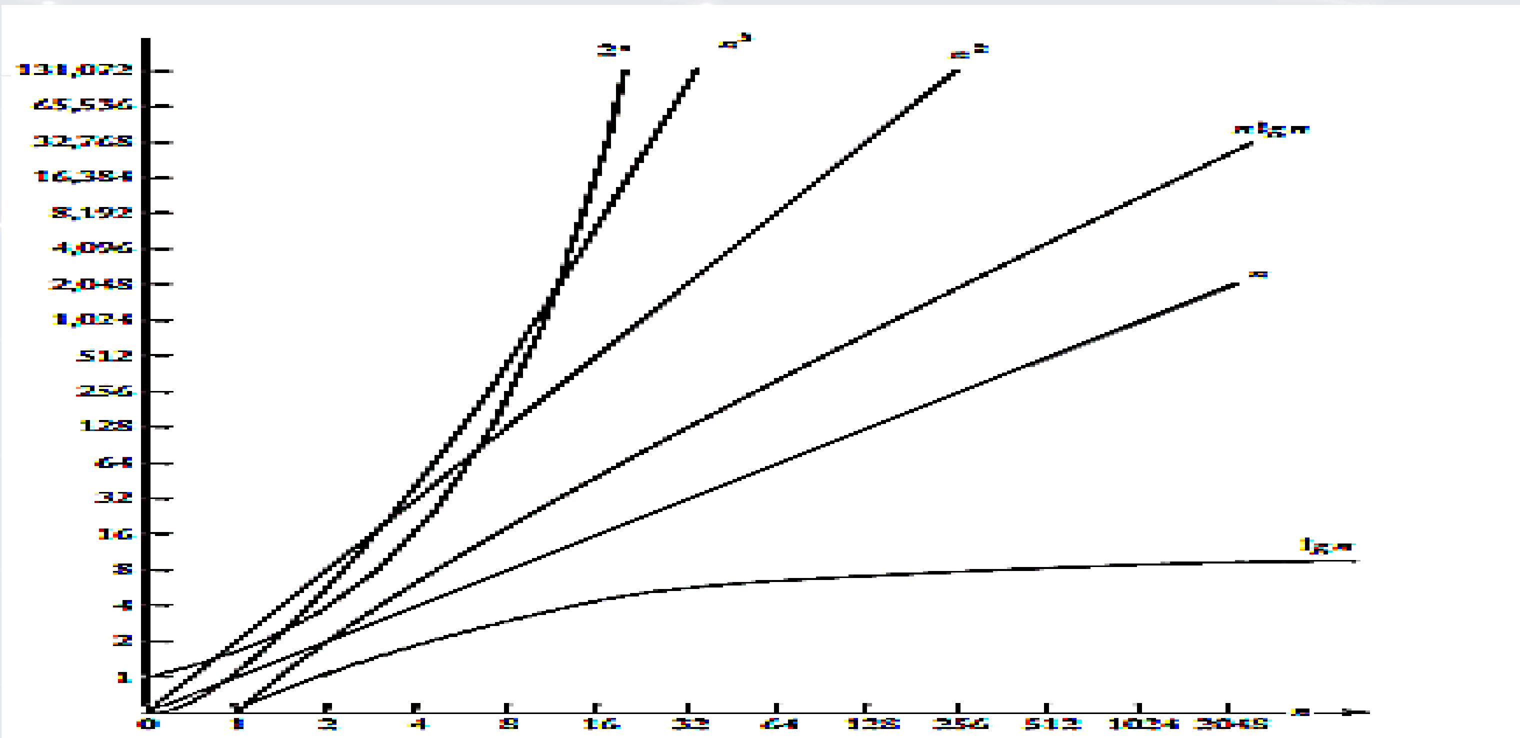
Complexity Analysis – Large n

- Worst Case
- Every Case
- Average Case
- Best Case

Order – Classes of Functions – Growth Rate

- Θ(f) – At the rate of f
- O(f) – At most as fast as f
- Ω(f) - At least as fast as f
- n grows more slowly than n^3 => n ε O(n^3)
- n^3 grows faster than n => n^3 ε Ω(n)
- By definition n and 2n grow at the same rate => 2n ε Θ(n)

Growth Rates of Common Complexity Functions



Big O

- For a given complexity function f (n), O(f(n)) is the set of complexity functions g(n) for which there exists some positive real constant c and some nonnegative integer N such that for all $n \geq N$,
- $g(n) \leq c \times f(n)$
- $g(n) \in O(f(n))$

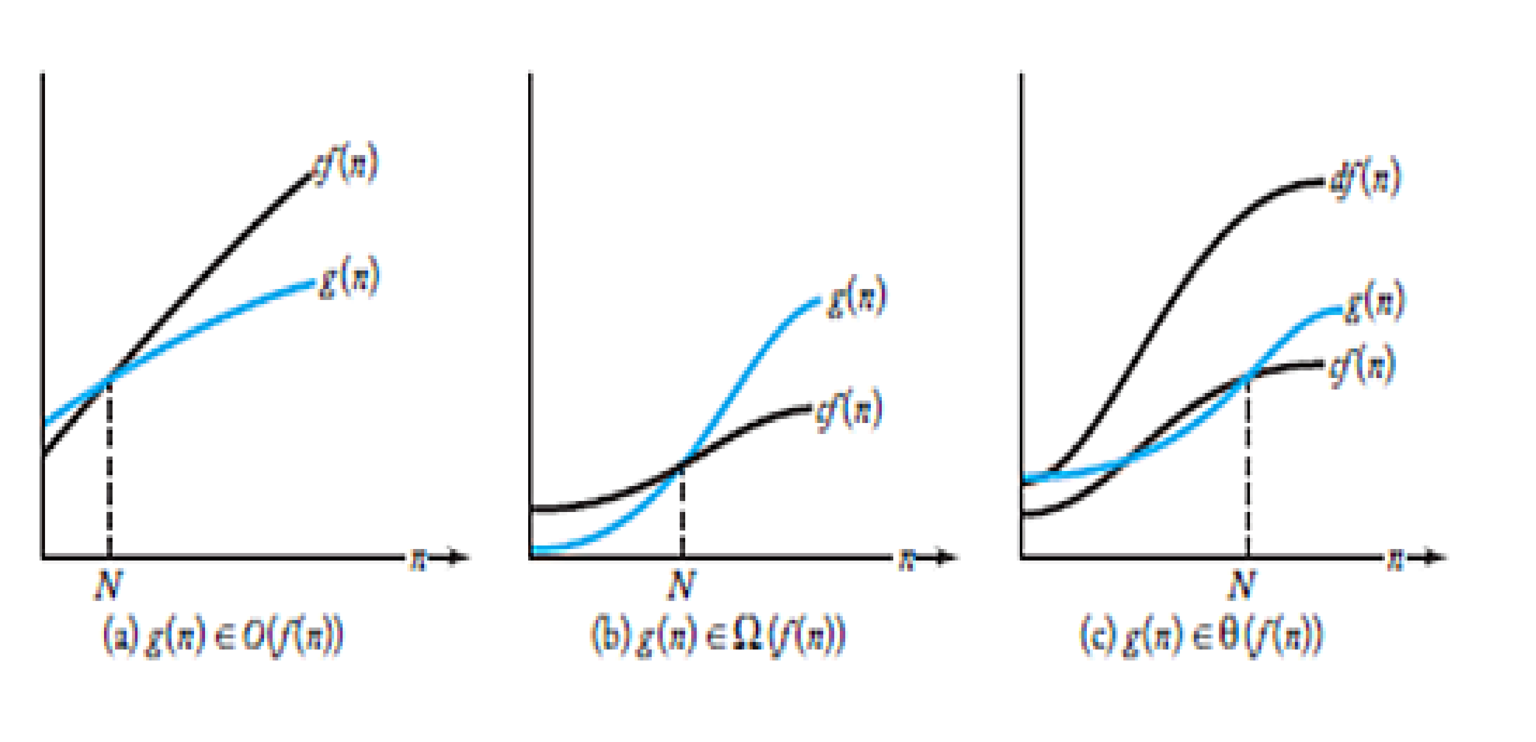
Omega

- For a given complexity function $f(n)$, $\Omega(f(n))$ is the set of complexity functions $g(n)$ for which there exists some positive real constant c and some nonnegative integer N such that, for all $n \geq N$,
- $g(n) \geq c \times f(n)$.
- $g(n) \in \Omega(f(n))$

Theta

- For a given complexity function $f(n)$,
- $\theta(f(n)) = O(f(n)) \cap \Omega(f(n))$
- This means that $\theta(f(n))$ is the set of complexity functions $g(n)$ for which there exists some positive real constants c and d and some nonnegative integer N such that, for all $n \geq N$,
- $c \times f(n) \leq g(n) \leq d \times f(n)$.
- $g(n) \in \theta(f(n))$

Big O, Omega, Theta



Limit Definitions for Big O, Theta, and Omega

- $\text{Lim}_{n\text{->}\infty}$ g(n)/f(n) = c => g(n) ε θ(f(n)) for c > 0 and c != ∞
- $\text{Lim}_{n\text{->}\infty}$ g(n)/f(n) = c => g(n) ε O(f(n)) where c ε Set of all positive real numbers union 0
- $\text{Lim}_{n\text{->}\infty}$ g(n)/f(n) =∞ OR $\text{Lim}_{n\text{->}\infty}$ g(n)/f(n) = c >0 => g(n) ε Ω(f(n))