Introduction to Data Structures and Algorithms

COMP 251

Preliminary Quiz

STATE YOUR NAME AND UFVID and answer the following questions, please:

1. Define *Algorithm.*
2. Define Dijkstra's *Control Structures.*
3. Define *Von Neuman Architecture*.
4. Describe the *Instruction Cycle*.
5. Define the main characteristics of OOP.
6. Define *Recursion*.
7. Write the <u>Java Code</u> for "Is there anybody out there?"
8. What does "*a* mod *b*" mean in words?

Goals of this Lecture

- Briefly summarize the introductory material (Section 1: Chapters 1.- 4 in several PPTs).
- Introduce the following fundamental notions:
  - The *need for data structures* within the context of computing,
  - The *programming paradigms* and to compare and contrast the various approaches to computing within these paradigms,
  - The core ideas behind **data structures** (which are nothing more than [presumably efficient] *methods of information storage, manipulation and retrieval*), and
  - The core ideas behind **algorithms** (*which are nothing more than methods of addressing solvable problems where data structures are an integral part of algorithmic efficiency*).

Goals of this Lecture

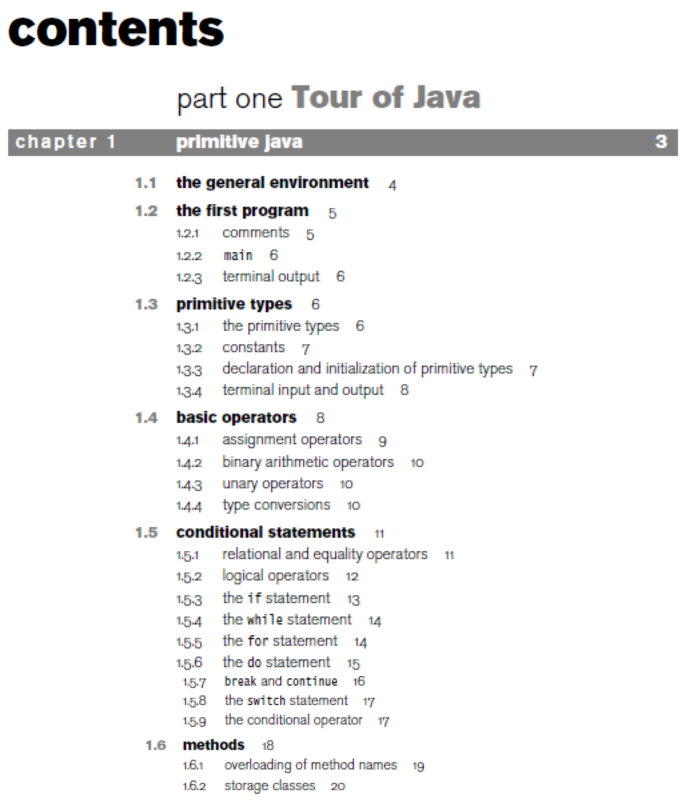
- **NOTE:** This is an **intermediate-level course** with emphasis on OO problem solving in **Java**.
- My approach is to *briefly introduce data structures from the procedural viewpoint* (*i.e.,* a language like C), which is *less abstract* and, therefore, more amenable for developing the core concepts of data structures.
- Then, we increase the level of *abstraction* using OO techniques within the context of Java, which will form the main body of this course.

Preamble

Section One Summary and Preamble

Data Structures & Problem Solving Using Java

fourth edition

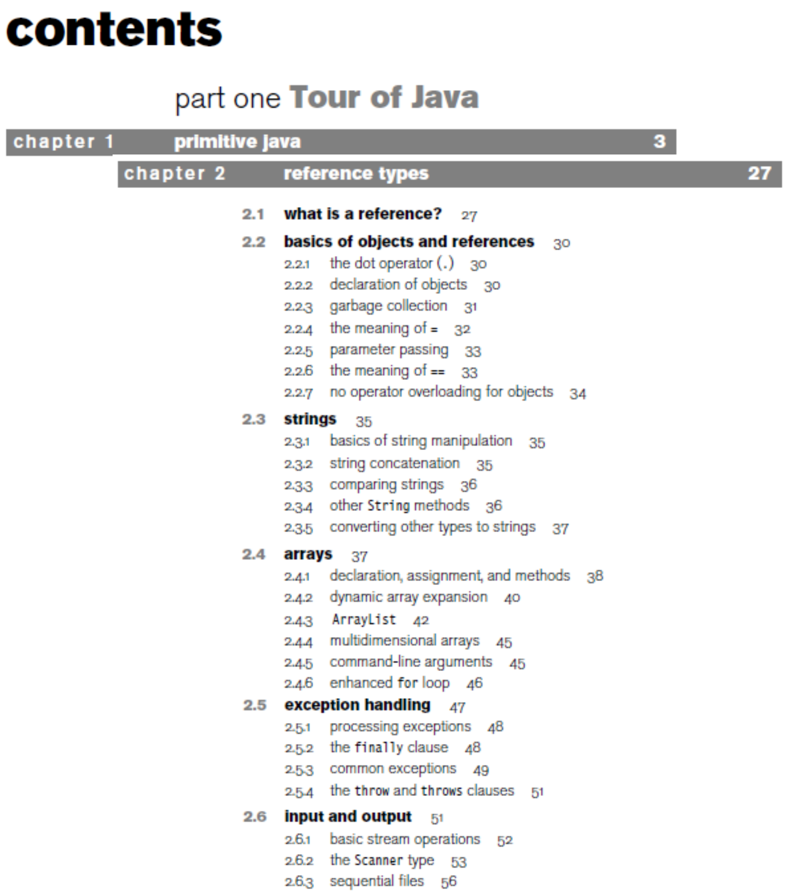


contents

Section One Summary and Preamble

Data Structures & Problem Solving Using Java

fourth edition

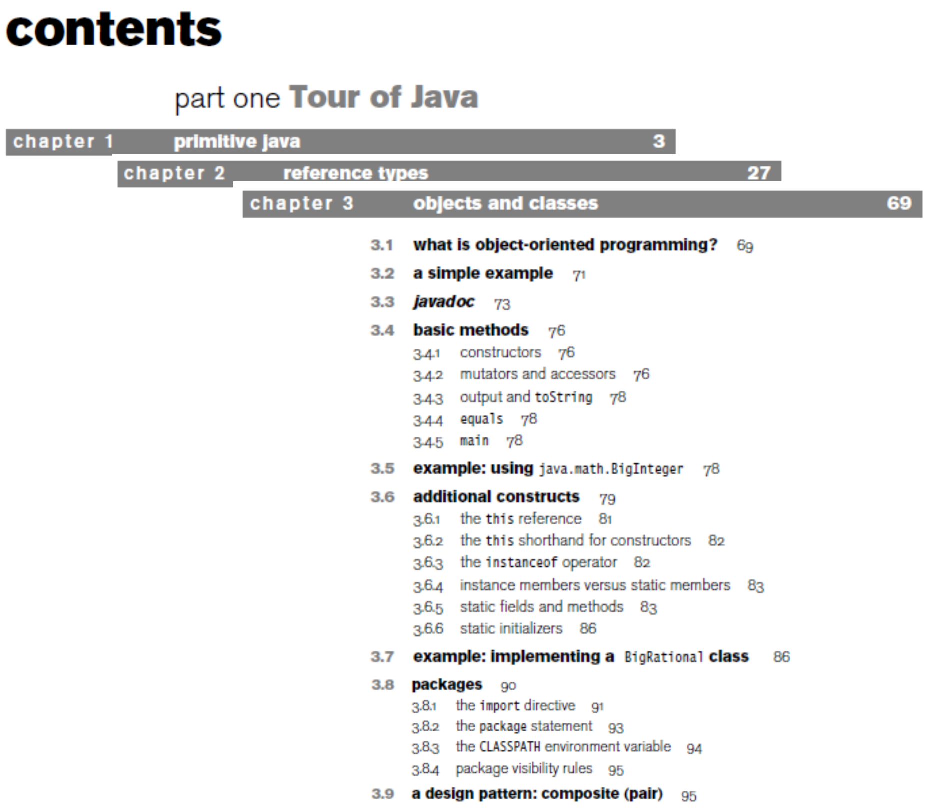


contents

Section One Summary and Preamble

Data Structures & Problem Solving Using Java

fourth edition

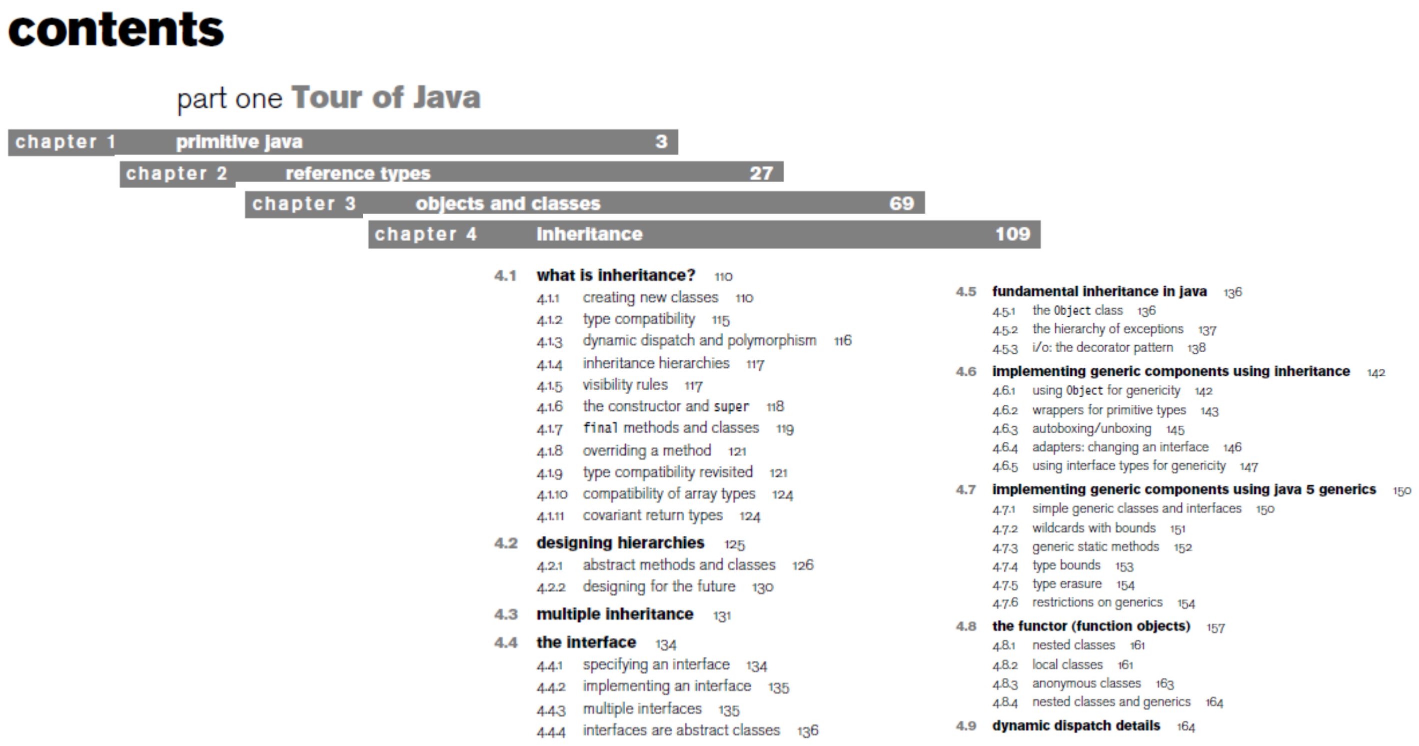


contents

Section One Summary and Preamble

Data Structures & Problem Solving Using Java

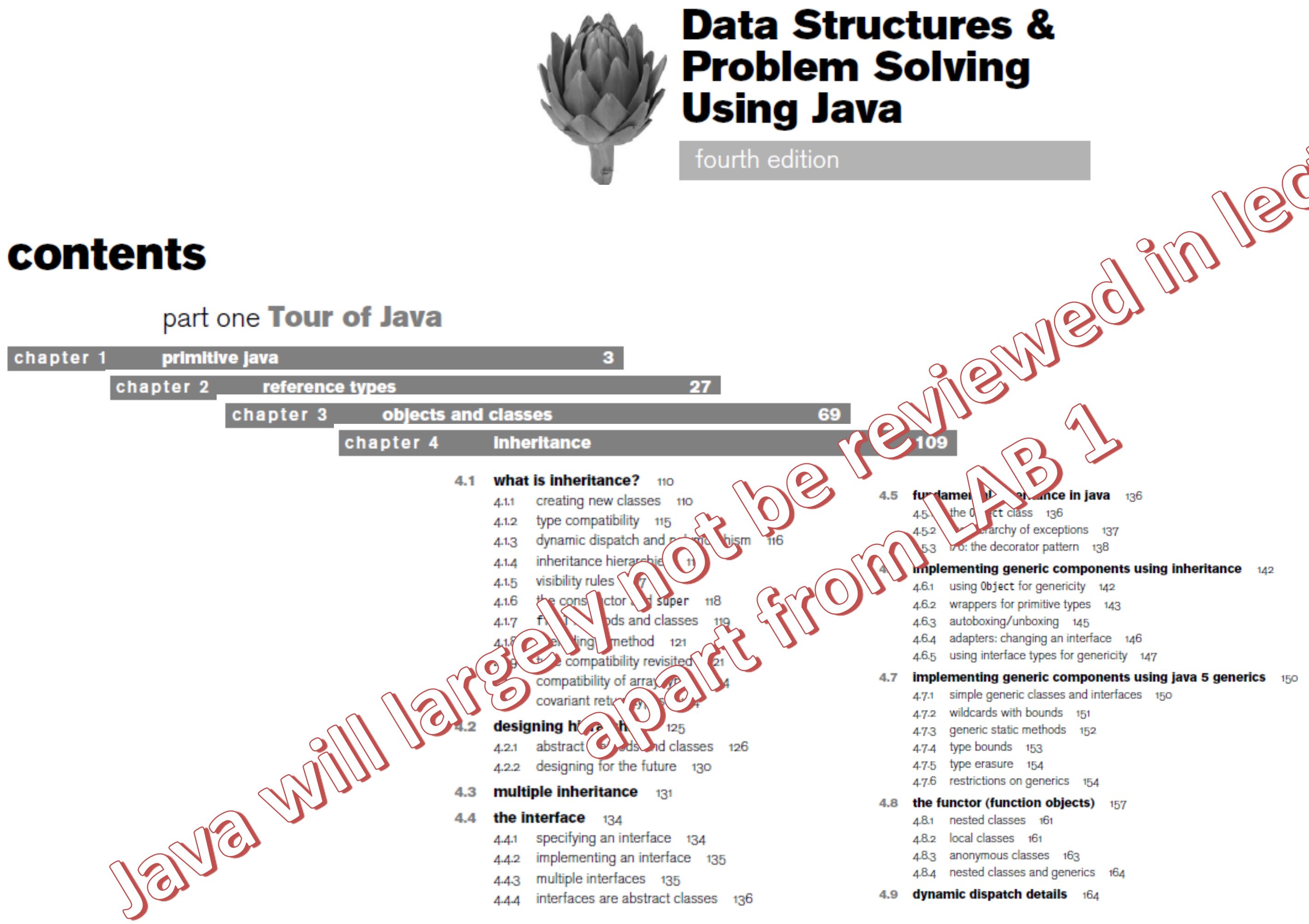
fourth edition



contents

part one **Tour of Java**

Section One Summary and Preamble



Data Structures & Problem Solving Using Java

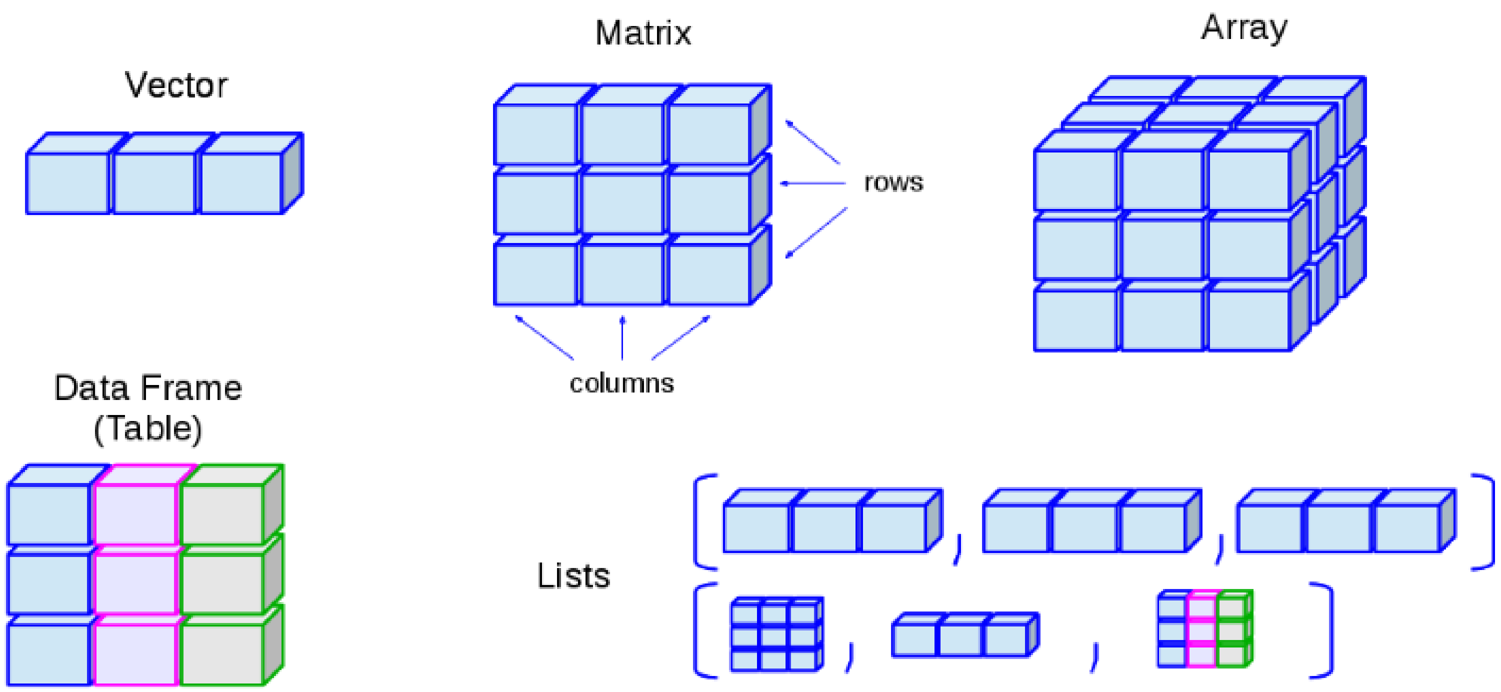
fourth edition

contents

part one **Tour of Java**

Java will largely not be reviewed in lecture, apart from LAB 1

Beyond Java: Introductory Notions of Data Structures and Algorithms



DEFINITION: Computing Science

DEFINITION: Computing Science

The study of *algorithms*, including

DEFINITION: Computing Science

The study of *algorithms*, including

- Their formal and mathematical properties,

DEFINITION: Computing Science

The study of *algorithms*, including

- Their formal and mathematical properties,
- Their hardware realizations,

DEFINITION: Computing Science

The study of *algorithms*, including

- Their formal and mathematical properties,
- Their hardware realizations,
- Their linguistic realizations,

DEFINITION: Computing Science

The study of *algorithms*, including

- Their formal and mathematical properties,
- Their hardware realizations,
- Their linguistic realizations,
- Their applications.

Schneider and Gersting, 1999

Definition: Algorithm

Definition: Algorithm

A well-ordered collection of unambiguous and effectively computable operations that, when executed, produces a result and halts in a finite amount of time.

- Schneider and Gersting, 1999

Exercise

Apply this definition to the quadratic equation an consider how to implement it with pseudocode.

Exercise

Apply this definition to the quadratic equation an consider how to implement it with pseudocode.

- To be *effectively computable*, the discriminant must not be less than zero AND the leading coefficient must not be zero.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

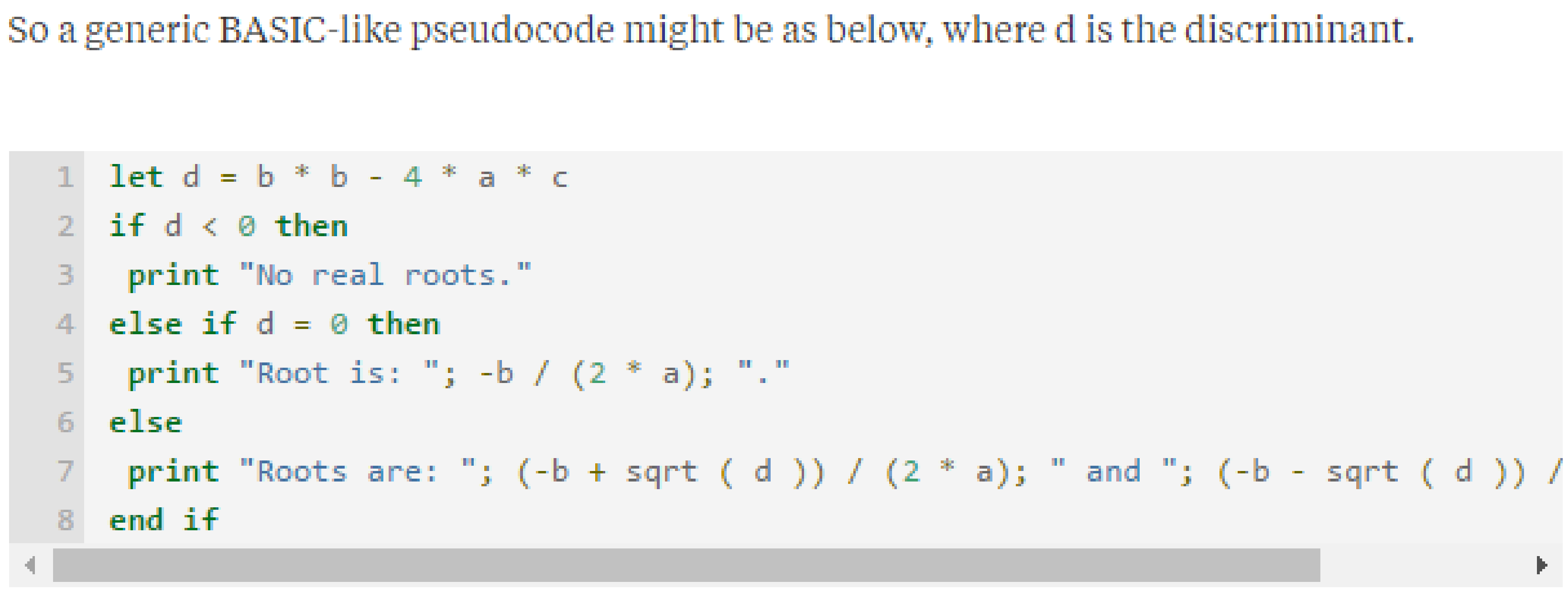
Exercise

Apply this definition to the quadratic equation an consider how to implement it with pseudocode.

- To be *effectively computable*, the discriminant must not be less than zero AND the leading coefficient must not be zero.
- Note that software development begins with defining the problem (requirements of SWE).

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Exercise

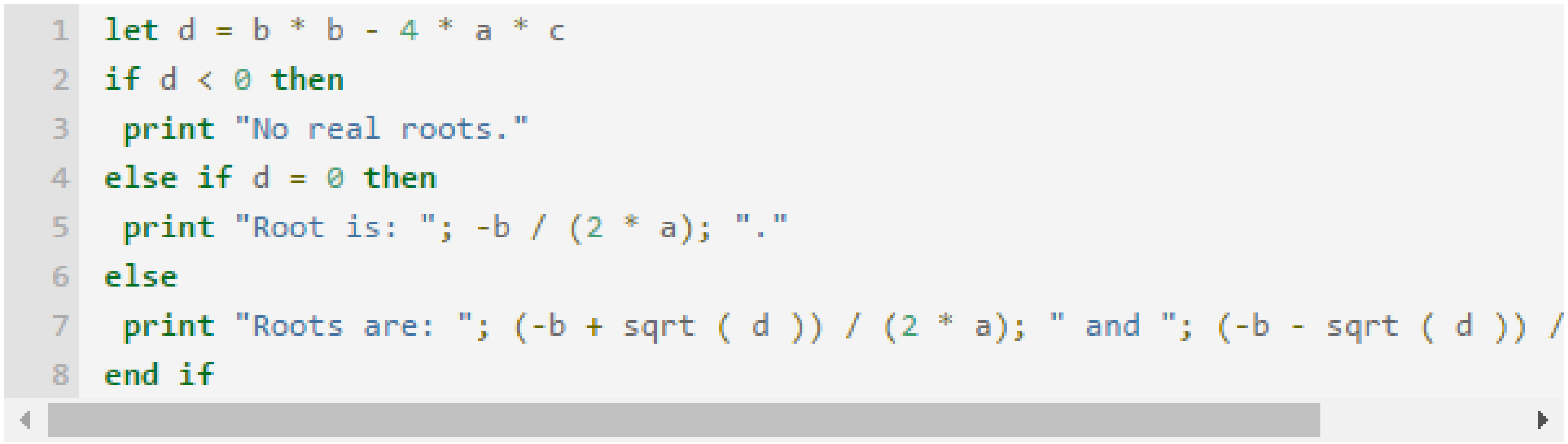


So a generic BASIC-like pseudocode might be as below, where d is the discriminant.

```
let d = b * b - 4 * a * c
if d < 0 then
 print "No real roots."
else if d = 0 then
 print "Root is: "; -b / (2 * a); "."
else
 print "Roots are: "; (-b + sqrt ( d )) / (2 * a); " and "; (-b - sqrt ( d )) /
end if
```

Exercise

So a generic BASIC-like pseudocode might be as below, where d is the discriminant.



```
let d = b * b - 4 * a * c
if d < 0 then
 print "No real roots."
else if d = 0 then
 print "Root is: "; -b / (2 * a); "."
else
 print "Roots are: "; (-b + sqrt ( d )) / (2 * a); " and "; (-b - sqrt ( d )) /
end if
```

Note that this is best done using Procedural Programming – OO is an unnecessary abstraction for most simple problems in mathematics.

Areas of Computing Science

Areas of Computing Science

- Data Structures (Efficient Organization of Data)

Areas of Computing Science

- Data Structures (Efficient Organization of Data)
- Operating Systems (Managing the Computer)

Areas of Computing Science

- Data Structures (Efficient Organization of Data)
- Operating Systems (Managing the Computer)
- Programming Languages (Software)

Areas of Computing Science

- Data Structures (Efficient Organization of Data)
- Operating Systems (Managing the Computer)
- Programming Languages (Software)
- Computational Complexity (Limitations of Computing)

Areas of Computing Science

- Data Structures (Efficient Organization of Data)
- Operating Systems (Managing the Computer)
- Programming Languages (Software)
- Computational Complexity (Limitations of Computing)
- Architecture (Design of Computers)

Areas of Computing Science

- Data Structures (Efficient Organization of Data)
- Operating Systems (Managing the Computer)
- Programming Languages (Software)
- Computational Complexity (Limitations of Computing)
- Architecture (Design of Computers)
- Intelligent Systems (AI)

Areas of Computing Science

- Data Structures (Efficient Organization of Data)
- Operating Systems (Managing the Computer)
- Programming Languages (Software)
- Computational Complexity (Limitations of Computing)
- Architecture (Design of Computers)
- Intelligent Systems (AI)
- Automata Theory (Abstract Computers)

Areas of Computing Science

- Data Structures (Efficient Organization of Data)
- Operating Systems (Managing the Computer)
- Programming Languages (Software)
- Computational Complexity (Limitations of Computing)
- Architecture (Design of Computers)
- Intelligent Systems (AI)
- Automata Theory (Abstract Computers)
- Information Storage and Retrieval (DBMS)

Areas of Computing Science

- Data Structures (Efficient Organization of Data)
- Operating Systems (Managing the Computer)
- Programming Languages (Software)
- Computational Complexity (Limitations of Computing)
- Architecture (Design of Computers)
- Intelligent Systems (AI)
- Automata Theory (Abstract Computers)
- Information Storage and Retrieval (DBMS)
- Software Engineering (generally large systems designed with professional engineering practices)

Post-Preamble Summary

- We have defined fundamental terminology:
  - Computing Science,
  - Algorithms,
- We have also provided an overview of the various areas of research in Computing Science and *have seen where Data Structures fits into this taxonomy*.

The Fundamental Motivation for Data Structures

The Philosophy of Data Structures

- The primary goal of this lecture is to *motivate the <u>need</u> to study data structures*.
- This involves understanding that **unlimited computing power has logical barriers** and that, hence, there is a fundamental requirement to understand the nature of how data may be manipulated by any computer and, at the very least, for the sake of **efficient storage and retrieval of data**.
- **Computer programs are formal mathematical structures and, hence, these issues begin with the mathematical nature of algorithms.**

Different Types of “Well-Known” Functions

Given a constant, *n*, and a variable, *x*, we can construct the following:

- x^n – **Mono**mial

Different Types of “Well-Known” Functions

Given a constant, *n*, and a variable, *x*, we can construct the following:

- x^n – **Mono**mial
- $P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$ – **Poly**nomial
  - $P_2(x) = a_2 x^2 + a_1 x + a_0$ – Quadratic
  - $P_1(x) = a_1 x + a_0$ – Linear
  - $P_0(x) = a_0$ – Constant

Different Types of “Well-Known” Functions

Given a constant, *n*, and a variable, *x*, we can construct the following:

- x^n – **Mono**mial
- $P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$ – **Poly**nomial
  - $P_2(x) = a_2 x^2 + a_1 x + a_0$ – Quadratic
  - $P_1(x) = a_1 x + a_0$ – Linear
  - $P_0(x) = a_0$ – Constant

- n^x – Exponential

Different Types of “Well-Known” Functions

Given a constant, *n*, and a variable, *x*, we can construct the following:

- x^n – **Mono**mial
- $P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$ – **Poly**nomial
  - $P_2(x) = a_2 x^2 + a_1 x + a_0$ – Quadratic
  - $P_1(x) = a_1 x + a_0$ – Linear
  - $P_0(x) = a_0$ – Constant
- n^x – Exponential
- $n!$ – Factorial

Different Types of "Well-Known" Functions

Given a constant, *n*, and a variable, *x*, we can construct the following:

- x^n – **Mono**mial
- $P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$ – **Poly**nomial
  - $P_2(x) = a_2 x^2 + a_1 x + a_0$ – Quadratic
  - $P_1(x) = a_1 x + a_0$ – Linear
  - $P_0(x) = a_0$ – Constant
- n^x – Exponential
- $n!$ – Factorial
- And their inverse functions (such as $x^{1/n}$ or $\log x$)

Computational Time of Various Classes of Problems According to Input Size

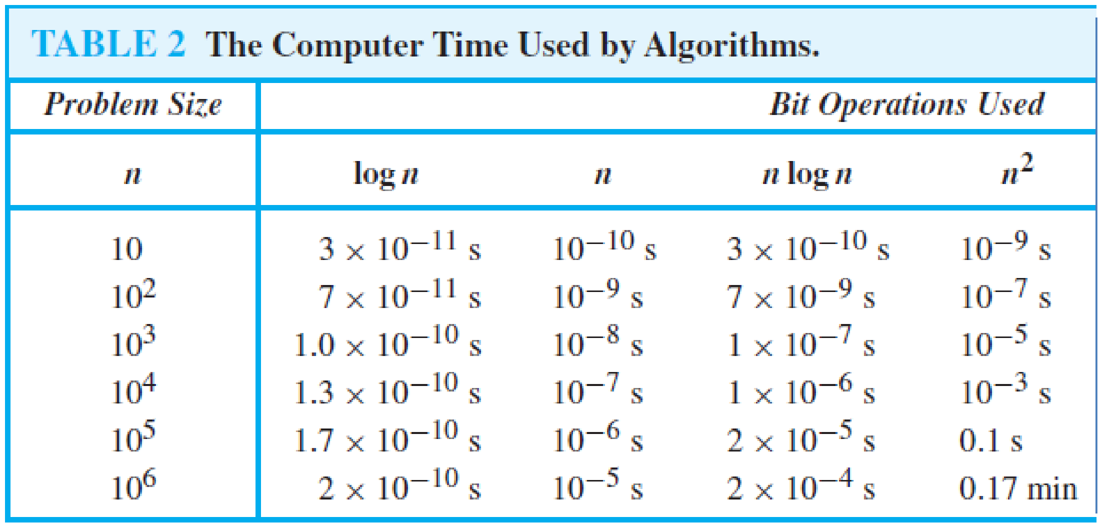


TABLE 2 The Computer Time Used by Algorithms.

Problem Size	*Bit Operations Used*			
n	$\log n$	n	$n \log n$	n^2
10	3×10^{-11} s	10^{-10} s	3×10^{-10} s	10^{-9} s
10^2	7×10^{-11} s	10^{-9} s	7×10^{-9} s	10^{-7} s
10^3	1.0×10^{-10} s	10^{-8} s	1×10^{-7} s	10^{-5} s
10^4	1.3×10^{-10} s	10^{-7} s	1×10^{-6} s	10^{-3} s
10^5	1.7×10^{-10} s	10^{-6} s	2×10^{-5} s	0.1 s
10^6	2×10^{-10} s	10^{-5} s	2×10^{-4} s	0.17 min

Rosen

Computational Time of Various Classes of Problems According to Input Size

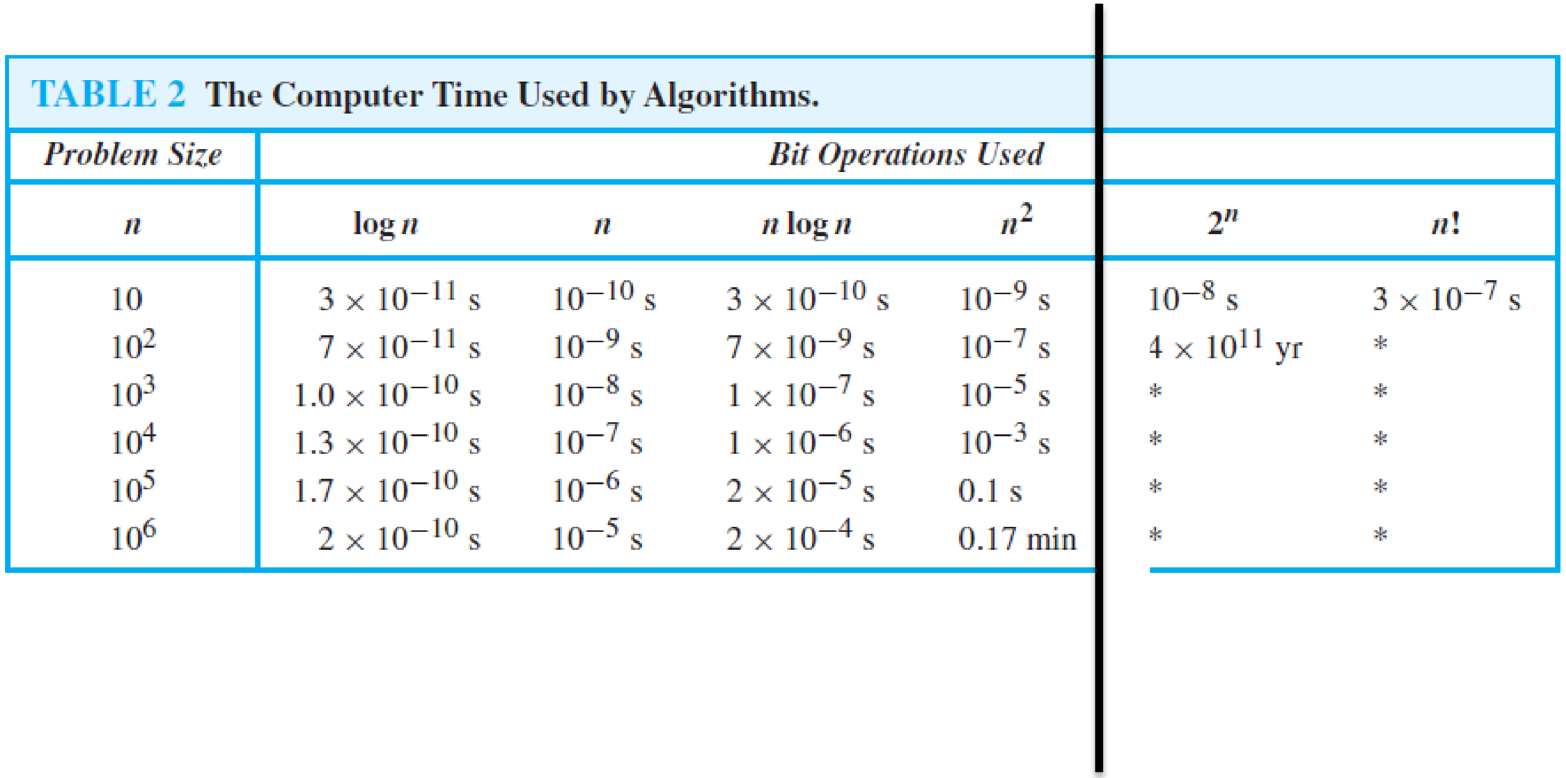


TABLE 2 The Computer Time Used by Algorithms.

Problem Size	*Bit Operations Used*					
n	$\log n$	n	$n \log n$	n^2	2^n	$n!$
10	3×10^{-11} s	10^{-10} s	3×10^{-10} s	10^{-9} s	10^{-8} s	3×10^{-7} s
10^2	7×10^{-11} s	10^{-9} s	7×10^{-9} s	10^{-7} s	4×10^{11} yr	*
10^3	1.0×10^{-10} s	10^{-8} s	1×10^{-7} s	10^{-5} s	*	*
10^4	1.3×10^{-10} s	10^{-7} s	1×10^{-6} s	10^{-3} s	*	*
10^5	1.7×10^{-10} s	10^{-6} s	2×10^{-5} s	0.1 s	*	*
10^6	2×10^{-10} s	10^{-5} s	2×10^{-4} s	0.17 min	*	*

Rosen

Computational Time of Various Classes of Problems According to Input Size

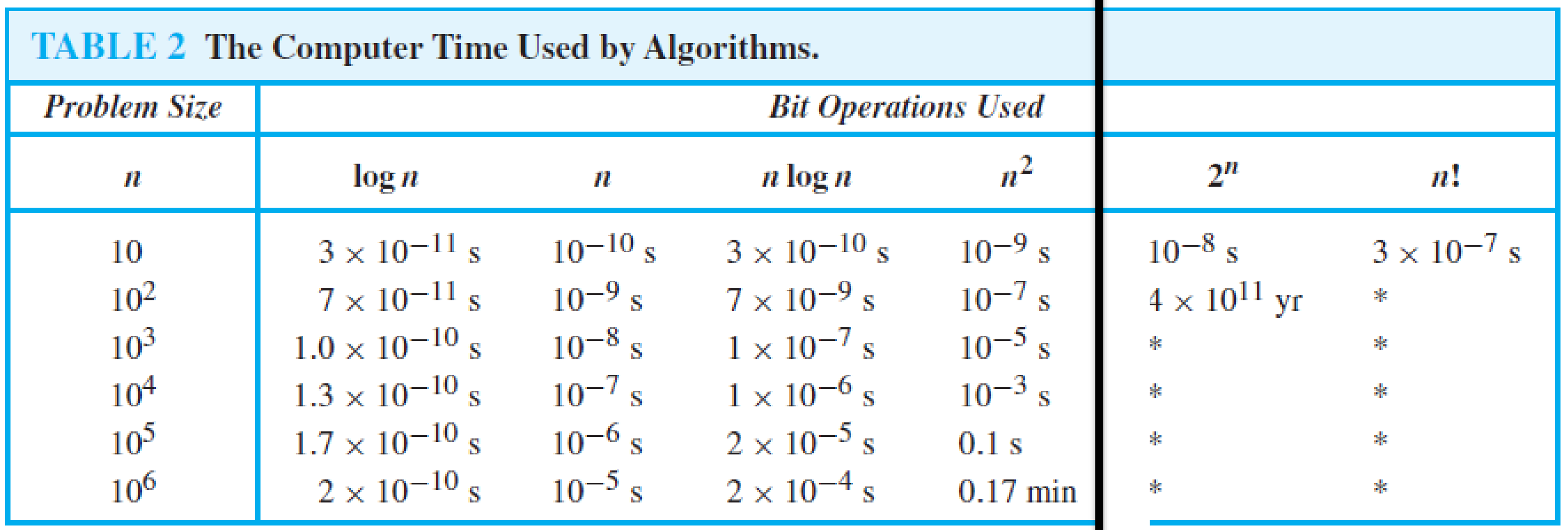


TABLE 2 The Computer Time Used by Algorithms.

Problem Size	*Bit Operations Used*					
n	$\log n$	n	$n \log n$	n^2	2^n	$n!$
10	3×10^{-11} s	10^{-10} s	3×10^{-10} s	10^{-9} s	10^{-8} s	3×10^{-7} s
10^2	7×10^{-11} s	10^{-9} s	7×10^{-9} s	10^{-7} s	4×10^{11} yr	*
10^3	1.0×10^{-10} s	10^{-8} s	1×10^{-7} s	10^{-5} s	*	*
10^4	1.3×10^{-10} s	10^{-7} s	1×10^{-6} s	10^{-3} s	*	*
10^5	1.7×10^{-10} s	10^{-6} s	2×10^{-5} s	0.1 s	*	*
10^6	2×10^{-10} s	10^{-5} s	2×10^{-4} s	0.17 min	*	*

$\longleftarrow$ **TRACTABLE** $\longrightarrow$ $\longleftarrow$ **INTRACTABLE** $\longrightarrow$

Rosen

Informal Definition

Tractable Problem:

An algorithm that does not have a mathematical structure that is worse than a polynomial.

Informal Definition

Tractable Problem:

An algorithm that does not have a mathematical structure that is worse than a polynomial.

Note: The idea here is that a tractable problem can be solved in *a reasonable time* with reasonable *storage* (*i.e.,* you can solve it in less than the age of the known universe).

Computational Complexity

- We will deal with this important subject in greater detail further on in this course.
- In particular, we will provide a formal definition of an *intractable function* which will require auxiliary definitions and an analysis of the **growth of functions**.
- Moreover, this will lead to a discussion of the **logical limitations of computation and of science itself** where, in the latter case, algorithms are non-existent for certain types of problems.

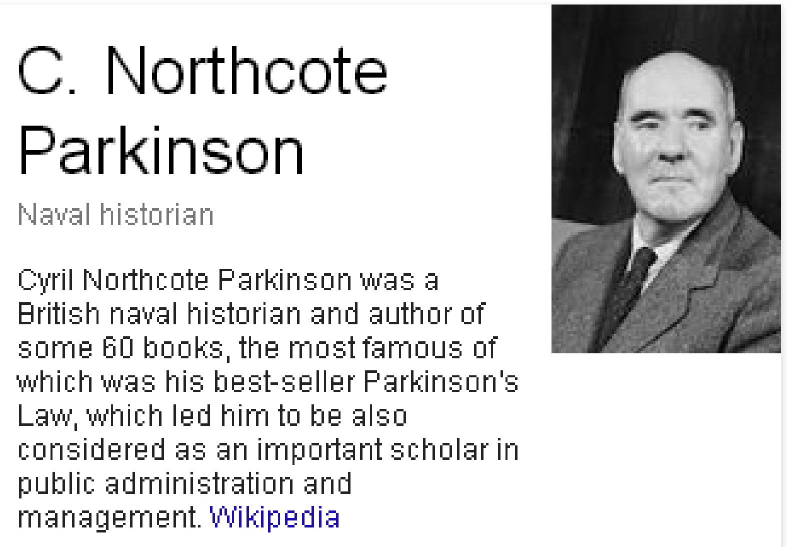
The Philosophy of Data Structures

PARKINSON’S LAW

The consequences of the forgoing subject-matter that are germane to this course.

PARKINSON'S LAW

"Work expands so as to fill the time available for its completion."



C. Northcote Parkinson

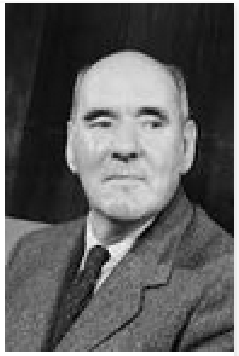
Naval historian

Cyril Northcote Parkinson was a British naval historian and author of some 60 books, the most famous of which was his best-seller Parkinson's Law, which led him to be also considered as an important scholar in public administration and management. Wikipedia

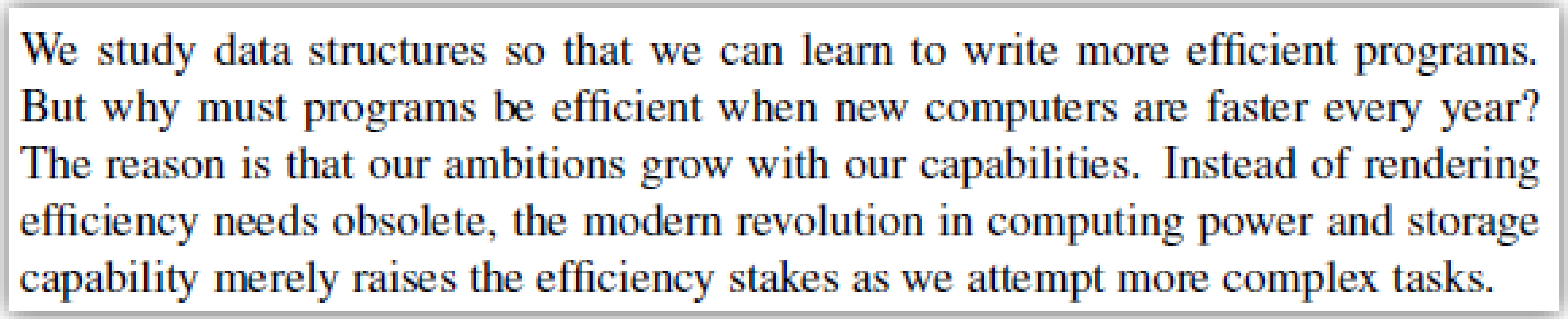
“Parkinson’s Law of Algorithmic Complexity”

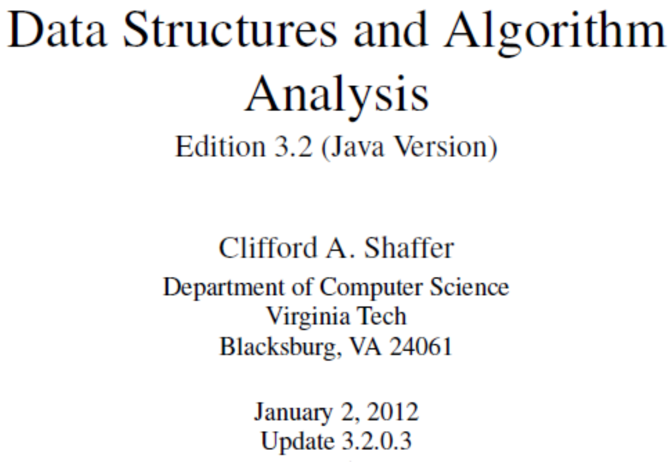
The complexity of algorithms expands at a rate that is roughly proportional to the increase in the available computational power.

C. Northcote Parkinson



“Parkinson’s Law of Algorithmic Complexity”


We study data structures so that we can learn to write more efficient programs. But why must programs be efficient when new computers are faster every year? The reason is that our ambitions grow with our capabilities. Instead of rendering efficiency needs obsolete, the modern revolution in computing power and storage capability merely raises the efficiency stakes as we attempt more complex tasks.


Data Structures and Algorithm Analysis

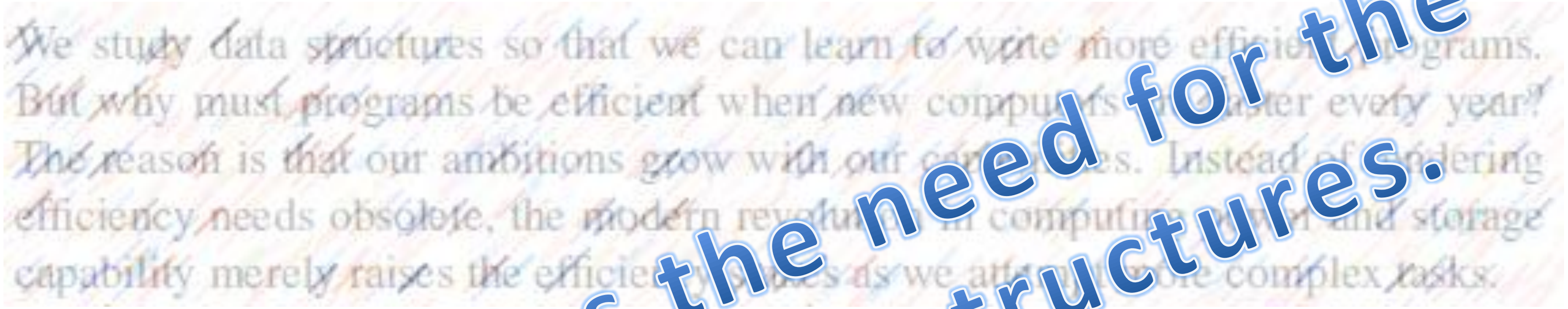
Edition 3.2 (Java Version)

Clifford A. Shaffer
Department of Computer Science
Virginia Tech
Blacksburg, VA 24061

January 2, 2012
Update 3.2.0.3

Our Online Reference
https://people.cs.vt.edu/shaffer/Book/Java3e20120102.pdf

"Parkinson's Law of Algorithmic Complexity"



We study data structures so that we can learn to write more efficient programs. But why must programs be efficient when new computers are faster every year? The reason is that our ambitions grow with our capabilities. Instead of rendering efficiency needs obsolete, the modern revolution in computing power and storage capability merely raises the efficiency stakes as we attempt more complex tasks.

Herein lies the need for the study of data structures.

Data Structures and Algorithm Analysis

Edition 3.2 (Java Version)

Clifford A. Shaffer

Department of Computer Science

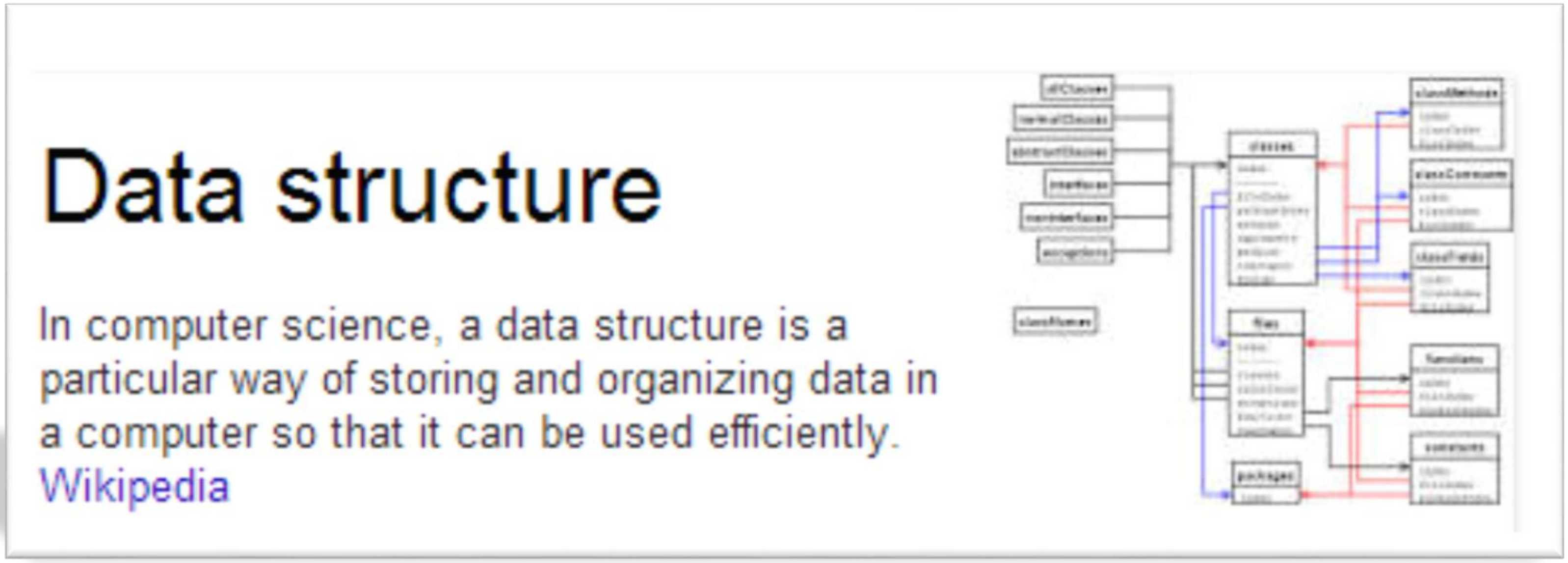
Virginia Tech

Blacksburg, VA 24061

January 2, 2012

Update 3.2.0.3

Definition: Data Structure



Data structure

In computer science, a data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently.
Wikipedia

Presentation Terminated