**Chapter 1**

**An Introduction to Computer Science**

**A Guide to this Instructor’s Manual:**

We have designed this Instructor’s Manual to supplement and enhance your teaching experience through classroom activities and a cohesive chapter summary.

This document is organized chronologically, using the same headings that you see in the textbook. Under the headings you will find: lecture notes that summarize the section, Teaching Tips, Class Discussion Topics, and Additional Projects and Resources. Pay special attention to teaching tips and activities geared towards quizzing your students and enhancing their critical thinking skills.

In addition to this Instructor’s Manual, our Instructor’s Resources also contain PowerPoint Presentations, Test Banks, and other supplements to aid in your teaching experience.

|  |
| --- |
| **At a Glance** |

**Instructor’s Manual Table of Contents**

* Overview
* Learning Objectives
* Teaching Tips and Quick Quizzes
* Class Discussion Topics
* Additional Projects
* Additional Resources
* Key Terms
* Solutions to End-of-Chapter Exercises

|  |
| --- |
| Lecture Notes |

**Overview**

Chapter 1 addresses common misconceptions about the field of computer science, carefully examines the Gibbs–Tucker definition of computer science, introduces the definition of computer science, and surveys the history of the field. This definition focuses on the idea of an *algorithm*. The chapter defines the concept of an algorithm, gives examples of algorithms, and explains the importance of algorithmic problem solving. The chapter provides a brief history of computing, beginning with calculating devices that predate modern computers by centuries, and includes key developments from the 1940s onward. The chapter ends by explaining how the organization of the textbook connects directly to the Gibbs–Tucker definition of computer science. This organizational structure is an essential feature of the text, and understanding it is a central goal of the course.

**Learning Objectives**

* Understand the definition of the term algorithm
* Understand the formal definition of computer science
* Write down everyday algorithms
* Determine if an algorithm is ambiguous or not effectively computable
* Understand the roots of modern computer science in mathematics and mechanical machines
* Summarize the key points in the historical development of modern electronic computers

**Teaching Tips**

**1.1 Introduction**

1. Begin by asking students their definition of computer science. Discuss the fact that many people do not have a clear understanding of what computer science professionals do and study.
2. Discuss the following misconceptions about the field of computer science (if students have proposed descriptions, categorize them using these misconceptions):
	1. Computer science is the study of computers.
	2. Computer science is the study of how to write computer programs.
	3. Computer science is the study of the uses and applications of computers and software.

**1.2 The Definition of Computer Science**

1. Introduce the term **algorithm**, and discuss everyday examples of algorithms: recipes, driving directions, instruction manuals, and so on.
2. Present and pick apart the Gibbs and Tucker definition of computer science. Note that the Gibbs and Tucker definition states that it is the task of the computer scientist to design and develop algorithms to solve a range of important problems. Give real-world examples for each part of the definition (Figures 1.1 and 1.2 contain two possible examples).
3. Prepare everyday examples of each of the three types of operations, and use them to illustrate that all operations used to construct algorithms belong to one of the three types:
	1. Sequential operations
	2. Conditional operations
	3. Iterative operations
4. Introduce the term **computing agent**, making clear that a computing agent may be hardware, software, or even human.
5. Describe the different kinds of unsolved problems: those that have no algorithmic solution, those that take too long to solve (see example of chess), and those that we do not yet know how to solve. Emphasize the importance of continuing work to solve new problems.

|  |  |
| --- | --- |
| ***Teaching Tip***  | Refer students to the following site to see current challenges for computing: <http://www.cra.org/ccc/initiatives> |

**1.3 Algorithms**

1. Introduce the formal definition of the term **algorithm.** Discuss each piece of the definition, asking students for their ideas about what each piece means.
2. Introduce the terms **unambiguous operation, primitive**, and **effectively computable**. Discuss the idea that what counts as unambiguous and effectively computable operations depends on the abilities of the computing agent. Take a simple example, and rewrite it for an adult expert, an average adult, a 12-year-old, and a 5-year-old.
3. Note the importance of every algorithm producing an observable result, whether it is a number, text, sound, or some other change to its environment.
4. Introduce the term **infinite loop**. Discuss why it is important that an algorithm stop and produce its result after a finite amount of time. What would happen otherwise?
5. Use Figures 1.3 and 1.4 or some similar set of algorithms to discuss correct, yet different, solutions to a simple problem. Poll students to find out which algorithm they prefer, and why.
6. Compare the automation of physical tasks in the industrial revolution to the automation of repetitive mental tasks in the computer revolution. Let students discuss the similarities and differences and the social pros and cons of such revolutions.

**Quick Quiz 1**

1. Which kind of operation is “*Add water until the cup is full*”?

Answer: Iterative

1. (True or false) Not all algorithms are known, computer scientists determine the best algorithm to produce a relevant result.

Answer: True

1. In computer science terminology, what is the machine, robot, person, or thing called that carries out the steps of the algorithm?

Answer: Computing agent

1. List at least two flaws in the “algorithm” below.

Given a jar full of jelly beans,

Pick a jelly bean from the jar

Add one to the total count

Repeat until the jar is empty

Answer: Possible answers include ambiguity about which lines to repeat, no initial value given to “total count,” and lack of clarity about what the result is.

**1.4 A Brief History of Computing**

This section contains many facts and details. The instructor must choose which facts to present: avoid extensive lectures that cram in every possible fact. Where possible, encourage students to engage with the material by presenting pieces of it to the class, preparing websites on particular topics, or searching for additional information online.

1. Explain that mathematical algorithms have been created for thousands of years, particularly for applications in building, navigation, and commerce.
2. Discuss a variety of early devices (slide rules, Pascaline, Jacquard looms, Difference and Analytical Engines, etc.) and what features of a modern computing device each device includes. These features include the ability to represent numbers or other kinds of information, operations (such as arithmetic) to manipulate the information, a memory to store information, and programmability—the ability to change the process or algorithm of the device. Discuss how each device implements the features it includes.

|  |  |
| --- | --- |
| ***Teaching Tip***  | Refer students to the following sites to see more about early computing devices (sites include many images): <http://www.computersciencelab.com/ComputerHistory/History.htm>http://www.computerhistory.org/timeline/ |

1. Discuss the importance of Babbage’s Analytical Engine design and its similarity to modern computer designs. Discuss with the class why the Analytical Engine was never completed and what the importance of Ada Augusta’s work with Babbage was.
2. Compare and contrast the Mark 1, ENIAC, Colossus, and other early computers. Describe the various early electronic computers and the kind of tasks they were designed to solve during World War II.

|  |  |
| --- | --- |
| ***Teaching Tip***  | The video *Top Secret Rosies: The Female Computers of World War II* is an excellent resource about early computing and World War II. |

1. Introduce the terms **Von Neumann architecture** and **stored program computer**, and discuss how Von Neumann’s design differed from earlier designs, including the Analytical Engine. The primary innovation is to place the program itself in the computer’s memory, removing the need to physically modify the computer in order to solve new problems. Virtually all modern computers are Von Neumann machines.
2. Survey the first through fifth generations of computing. Introduce the terms **high-level programming languages**, **minicomputer**, and **microcomputer**. Emphasize the memory and capabilities of each generation and the areas of application. Figure 1.8 summarizes this information.

**Quick Quiz 2**

1. What two reasons caused the Pascaline and the Leibnitz’s Wheel to not be considered computers?

Answer: They had no memory to store machine readable information, and they could not be programmed in advance to follow a set of sequential instructions.

1. Name two of the four components Babbage’s Analytical Engine had that make experts consider it, even just on paper, the first computer.

Answers: Mill or arithmetic/logic unit; store or memory; operator or processor; and output unit or input/output

1. Which of the following was the primary innovation of the Von Neumann architecture?
	1. Use of transistors instead of vacuum tubes
	2. Ability to perform floating point (real number) calculations
	3. Storage of program instructions in the internal memory unit
	4. Purely electronic design, no mechanical parts for computation

Answer: c

1. The first computer built for sale was called \_\_\_\_\_\_\_\_\_\_\_\_\_.

Answer: The UNIVAC I

1. (True or false) Microcomputers, the first desktop computers, were developed in the mid-1970s.

Answer: True

1. Herman Hollerith, working for the U.S. Census Bureau in the late 1800s, developed what early computing machine?

Answers: Many possible phrasings, but the answer should center on the “programmable card-processing machines that could automatically read, tally, and sort data entered on punch cards.” (See page 22 of *Invitation*.)

**1.5 Organization of the Text**

1. The organization of the book is based on the Gibbs and Tucker definition of computer science. Connect each “level” of the book to the relevant portion of the definition and explain the connection. Figure 1.9 gives a diagrammatic view of the book’s organization. Otherwise you could connect them as follows:

|  |  |
| --- | --- |
| **Gibbs and Tucker definition:** | **Levels of *Invitation to Computer Science*** |
|  Computer science is the study of  algorithms, including: |  |
| 1. Their formal and mathematical

properties | Level 1: The Algorithmic Foundations of Computer Science (Ch. 2, 3) |
| 1. Their hardware realizations
 | Level 2: The Hardware World (Ch. 4, 5)Level 3: The Virtual Machine (Ch. 6–8) |
| 1. Their linguistic realizations
 | Level 4: The Software World (Ch. 9–12) |
| 1. Their applications
 | Level 5: Applications (Ch. 13–16)Level 6: Social Issues (Ch. 17) |

1. Introduce the term **virtual machine**/**environment**. A virtual machine is composed only of the resources that the user perceives rather than all the resources that exist. The virtual machine is an abstract view of the machine.

**Class Discussion Topics**

1. What are devices in your home that appear to use computers or algorithms? Can you name at least one device for every room in your house? Describe one algorithm each device performs.
2. Think of a problem or task in your life that you wish had an algorithmic solution. How difficult is this problem to solve? What might an algorithmic solution require?
3. Discuss the social changes caused by the computer revolution of the past hundred years. What are the downsides to automating routine mental tasks? What are the upsides? Are there times when we might want to decide *not* to use a technical innovation? When and why?

**Additional Projects**

1. Working with one or two teammates, collect algorithms that you use in your everyday life, both those that are written down and those that you “just know.” Organize your algorithms into categories. What features are shared by all algorithms in a given category?
2. (This project is a kinesthetic in-class activity, and thus will be described from the instructor’s point of view.) When introducing the notion of an algorithm and the importance of it being unambiguous and effectively computable, ask the students to work in teams to develop an algorithm for an everyday problem. Examples others have used include making a peanut butter sandwich, counting a jar of M&Ms or buttons, or making a paper airplane. You, the instructor, are the computing agent. Place all necessary materials on a table in front of you. Perform *exactly* the algorithms given to you, without inferring any steps. For instance, if the algorithm says to “Put peanut butter on the bread,” then place the jar of peanut butter on top of the bread. Have students refine their algorithms until they get one that works.
3. Search the web for articles about the future of computing. Collect a list of the new innovations in computing in the next few years. How will our daily lives change as a result?

**Additional Resources**

1. Computational Tales: <http://computationaltales.blogspot.com/p/posts-by-topic.html>

This link reflects the work of Dr. Jeremy Kubica, software engineer and manager at Google. Dr. Kubica has gathered a set of computer science concepts written as fairy tales in order to provide students with an accessible overview of key concepts before diving into technical details. These fun and quirky stories cover a broad range of topics, from algorithms and data structures to general programming concepts.

1. Careers in Computing: <http://computingcareers.acm.org/>
2. Algorithm: <http://www.techterms.com/definition/algorithm>
3. Charles Babbage: <http://www.computerhistory.org/babbage/>
4. Computing history time line: <http://www.computerhistory.org/timeline/>
5. Early computing machines: <http://learn.fi.edu/learn/case-files/hci.html>

**Key Terms**

* **Algorithm:** Informally, an ordered sequence of instructions that is guaranteed to solve a problem; formally, a well-ordered collection of unambiguous and effectively computable operations that, when executed, produces a result and halts in a finite amount of time.
* **Analytical Engine:** A machine designed by Charles Babbage in the 1830s. Many consider it to be the first computer, although he never completed it.
* **Computer science:** The study of algorithms, including their mathematical properties, hardware and linguistic realizations, and their applications.
* **Computing agent:** The entity (machine, robot, person, or thing) that executes the steps of an algorithm.
* **Conditional operations:** Algorithmic operations that ask a question and select the next step based on the answer to that question.
* **Difference Engine**: A mechanical calculator that could do addition, subtraction, multiplication, and division to six significant digits and could solve polynomial equations and other complex mathematical problems as well.
* **Effectively computable:** There exists a method for actually carrying out the intent of the operation.
* **ENIAC:** The first fully electronic general-purpose programmable computer, completed in 1946; it contained 18,000 vacuum tubes and nearly filled a building.
* **High-level programming language:** A programming language that uses both natural language constructs and mathematical notation.
* **Infinite loop:** The repetitive execution of a block of operations that will never end. This is a fatal error when it occurs in an algorithm.
* **Iterative operations:** Algorithmic operations that repeat a block of instructions.
* **Luddites:** People who fear and are opposed to the use of new technologies.
* **Microcomputer:** Desktop computer that uses integrated circuit technology, developed in the mid-1970s, smaller than a minicomputer.
* **Minicomputer:** Smaller than mainframe computer, less expensive, developed in the mid-1960s.
* **Primitive:** When an operation is unambiguous for the agent carrying out the algorithm.
* **Sequential operation:** An algorithmic operation that carries out a single task and then moves on to the next operation in sequence.
* **Stored program computer:** A model of computation in which the instructions to be executed are represented as binary strings and stored in the memory of the computer.
* **Unambiguous operation:** An operation is unambiguous if it can be understood by the computing agent without having to be further defined or simplified.
* **Virtual machine (virtual environment)**: The computer system as perceived by the user as opposed to the hardware that actually exists; the set of services and resources created by the software and seen by the user.
* **Von Neumann architecture**: The computational model designed by John Von Neumann and first implemented in the EDSAC computer of 1947; the structure and organization of virtually all modern computers.

**Solutions to End-of-Chapter Exercises**

**1.** There is no one correct answer. Common examples are the instructions for using a voice mail system, the instructions for opening a mail box lock, and the instructions for doing laundry.

**2.** A *heuristic* is a method for finding a reasonably close, “good enough” solution to a problem. It can be viewed as a rule-of-thumb, a method of approximation, an informal technique, or even a way to make an “educated guess.” It differs from the concept of an algorithm in that it does not guarantee to produce an optimal solution, just to make a good faith attempt to locate a reasonable one. Heuristics are often used when executing an algorithm might be too time-consuming, and we only need an approximation to the correct answer.

An example of a heuristic for adding two 3-digit numbers, such as 234 + 567, might be:

 1. Set the one and tens digit of both operands to 0

 2. Increase the hundreds digit of the second operand by 1. These two

 steps result in changing the problem to the simpler one 200 + 600.

 3. Add the hundreds digits, resulting in a final “answer” of 800.

Now, of course, this is not the correct answer, which is 801. But the result we get may be close enough for our needs, and it is certainly a lot easier to add a single column of numbers rather than three columns of numbers.

**3.** One may argue that the instruction is not well-ordered, since it is unclear whether one should enter the channel first or press CHAN first. Also, it may not be effectively computable if you desire to enter a channel that is out of the DVR’s range.

**4.** (a) Sequential

 (b) Conditional

 (c) Sequential

 (d) Iterative

**5.** Step 1: *carry* = 0, *c*3 = ??, *c*2 = ??, *c*1 = ??, and *c*0 = ??

 Step 2: *i = 0,* all others unchanged

 Step 4: *c*0 = 18, all others unchanged

 Step 5: *c*0 = 8 and *carry* = 1, all others unchanged

 Step 6: *i* = 1, *carry* = 1, *c*3 = ??, *c*2 = ??, *c*1 = ??, and *c*0 = 8

 Step 4: *c*1 = 7, all others unchanged

 Step 5: *carry* = 0, all others unchanged

 Step 6: *i* = 2, *carry* = 0, *c*3 = ??, *c*2 = ??, *c*1 = 7, and *c*0 = 8

 Step 4: *c*1 = 1, all others unchanged

 Step 5: *carry* = 0, all others unchanged

 Step 6: *i*= 3, *carry* = 0, *c*3 = ??, *c*2 = 1, *c*1 = 7, and *c*0 = 8

 Step 7: *c*3 = 0, *c*2 = 1, *c*1 = 7, and *c*0 = 8

 Step 8: Print out 0178.

**6.** Replace Step 8 with the following steps:

 Step 8: Set the value of *i* to *m*

 Step 9: Repeat step 10 until either *ci* is not equal to 0 or *i* < 0

 Step 10: Subtract 1 from *i*, moving one digit to the right

 Step 11: If *i* > 0 then print *c*i*c*i-1 . . . *c*0

**7**. Assume that *a* has *n* digits an-1, … , a0, and *b* has *m* digits, bm-1, … , b0, with *n* not necessarily equal to *m*. Add an operation at the beginning of the algorithm that resets the two numbers to the same number of digits by adding non-significant leading zeros to the shorter one. We can then reuse the algorithm of Figure 1.2.

 If (m > n) then

 Set i to 0

 While (n+i < m)

 Add a leading zero to the number at position an+i

 Increment i by 1

 End of the loop

 Else

 If (n > m)

 Set i to 0

 While (m+i < n)

 Add a leading zero to the number at position bm+i

 Increment i by 1

 End of the loop

We have now made the two numbers equal in length. All we need do now is set the variable *m* to the larger of the two values:

 Set m to the larger of m and n.

The addition algorithm in Figure 1.2 will now work correctly. Note that if m and n are equal in value, neither of the Boolean expressions will be true, and neither of the conditional statements will be executed.

**8.** It is not effectively computable if *b*2 – 4*ac* < 0 (since we cannot take the square root of a negative number if we are limited to real numbers) or if *a* = 0 (since we cannot divide by 0).

**9.** The first algorithm (Figure 1.3(a)) is a better general purpose algorithm. If you want to shampoo your hair any number *n* times you can change the 2 to *n*. You could even ask the shampooer to input the desired number *n* of washings. For the second algorithm you would have to rewrite the algorithm to repeat steps 4 and 5 998 more times.

**10.** (a) Trace:

 Step 1: *I* = 32, *J* = 20, and *R* = ??

 Step 2: *I* = 32, *J* = 20, and *R* = 12

 Step 3: *I* = 20, *J* = 12, and *R* = 12

 Step 2: *I* = 20, *J* = 12, and *R* = 8

 Step 3: *I* = 12, *J* = 8, and *R* = 8

 Step 2: *I* = 12, *J* = 8, and *R* = 4

 Step 3: *I* = 8, *J* = 4, and *R* = 4

 Step 2: *I* = 8, *J* = 4, and *R* = 0

 Step 4: Print *J* = 4

 (b) At Step 2 we are asked to divide *I* = 32 by *J* = 0, which cannot be done. We can fix the problem by adding a step between Step 1 and Step 2 that says: If *J* = 0, then print “ERROR: division by 0” and Stop.

**11.** There are 25! possible paths to be considered. That is approximately 1.5 x 1025 different paths. The computer can analyze 10,000,000, or 107, paths per second. The number of seconds required to check all possible paths is about 1.5 x 1025/107, or about 1.5 x 1018 seconds. That’s roughly 1012 years: about a trillion years. This would not be a feasible algorithm.

**12.** A Multiplication Algorithm.

 Given: Two positive numbers *a* and *b*

 Wanted: A number *c* which contains the result of multiplying *a* and *b*

 Step 1: Set the value of *c* equal to 0

 Step 2: Set the value of *i* equal to *b*

 Step 3: Repeat steps 4 and 5 until the value of *i* is 0

 Step 4: Set the value of *c* to be *c* + *a*

 Step 5: Subtract 1 from *i*

 Step 6: Print out the final answer *c*

 Step 7: Stop

 This algorithm assumes that we know how to add two multiple-digit numbers together. We may assume this because we have the algorithm from the book which does exactly that.

**13**. The algorithm will work correctly only if all three numbers are unique. If two or more numbers are identical, none of the Boolean expressions will be true and nothing will be output. To make this a correct solution you either have to specify in the problem statement that the three numbers provided must all be distinct or (better) change all of the comparison operations to ≥ in place of >.

**14.**  This is an essay question. Students may find excellent resources on the Internet.

**15.** If this problem is assigned, be sure to coordinate with your computing staff ahead of time for students to get the required information.

**16.** This is an essay question. Because this is a “hot” topic, a great deal of hype and hyperbole is available, as well as useful information. It might be a good opportunity to teach students about finding *reliable* sources on the Internet, and evaluating online and print source materials.

**17.** Like question 16 this is an essay question. Students may be familiar with Apple iCloud services for iPhone and iPad devices, so it might be a good opportunity to relate their answers to the services provided by Apple.

**18.** About 130 feet ((((700,000,000 chars/5 chars per word)/300 words per page)/300 pages per inch)/12 inch per foot)

**Discussion of Challenge Work**

**1.** We may perform subtraction, like addition, by subtracting one column at a time, starting with the rightmost column and working to the left. Since we know that the first number is larger than the second one, we know that we can always borrow from columns to the left of the current one. Therefore, if the upper number in the column (*ai*) is smaller than the lower, we automatically borrow from the next column. We can do this by subtracting one from the *ai+1* value of the column to the left. If the *ai+*1 value were already zero, then it would become -1. This automatically causes a borrow to occur on the next step. Here is the algorithm:

 Step 1: Set the value of *i* equal to the value of 0

 Step 2: Repeat steps 3 to 6 until the value of *i* is *m*

Step 3: If *bi* < *ai* then

 Step 4: Set *ci* equal to *ai - bi*

 Otherwise (*bi > ai*)

 Step 5: Set *ci* equal to (*ai +* 10) – *bi* and replace *ai+*1 with *ai+*1 – 1

 (This amounts to a borrow of 1 from *ai*+1 which adds 10 to *ai*)

 Step 6: Add 1 to *i* (moving us one column to the left)

 Step 7: Print out the final answer *cm-*1*cm-*2 . . . *c0*

Step 8: Stop.

**2**. Students may need assistance finding or understanding other definitions from other sources.