Chapter 5: Algorithms

Computer Science: An Overview
Eleventh Edition

by
J. Glenn Brookshear
Dennis Brylow

Altered by N. Harris

Addison-Wesley is an imprint of PEARSON

Chapter 5: Algorithms

- 5.1 The Concept of an Algorithm
- 5.2 Algorithm Representation
- 5.3 Algorithm Discovery
- 5.4 Iterative Structures
- 5.5 Recursive Structures
- 5.6 Efficiency and Correctness

What do you already know about Algorithms?

This activity can be used in a topic where you think
students have misconceptions or where a topic flows from another so
you want to help students with recall.

Initial Activity

- This activity will let you explore POGIL (Process Oriented Guided Inquiry Learning)
- And lays the groundwork for further discussion of algorithms.

Brookshear’s Definition of Algorithm

An algorithm is an **ordered** set of **unambiguous**, **executable** steps that defines a **terminating** process.

So, what was hard about describing the process of calculating averages?

English as a way of representing algorithms is lousy. Why?

- Looking at these algorithms, which one is the “best”. Why?

English as a way of representing algorithms is lousy. Why?

- What makes for a good algorithm?
  - SCRAP (thanks Alan Crouch)
    - Simple
    - Complete (terminates)
    - Right (produces correct result)
    - Abstraction (hides unnecessary detail – or separates solution into submodules)
    - Precise (unambiguous language)

Algorithm Representation

- Requires well-defined primitives
- A collection of primitives constitutes a programming language.
- Helps with the clarity of the algorithm.

Pseudo code or code?

- Pseudo code is a shorthand way of expressing an algorithm.
- It is frequently a combination of code but without some of the detail…“a miracle occurs here” for example.

- Brookshear spends a section on pseudo code… teaching students a separate “language” for pseudo code can be confusing.

Polya’s Problem Solving Steps

- 1. Understand the problem.
- 2. Devise a plan for solving the problem.
- 3. Carry out the plan.
- 4. Evaluate the solution for accuracy and its potential as a tool for solving other problems.

Polya’s Steps in the Context of Program Development

- 1. Understand the problem.
  - This is where examples come in.
  - If you can come up with your own examples, you understand the nature of the problem.
- 2. Get an idea of how an algorithmic function might solve the problem.
- 3. Formulate the algorithm and represent it as a program.
  - These often are different steps. Design and then implementation.
- 4. Evaluate the solution for accuracy and its potential as a tool for solving other problems.

Getting a Foot in the Door

- Try working the problem backwards
- Solve an easier related problem
  - Relax some of the problem constraints
  - Solve pieces of the problem first (bottom up methodology)
- Stepwise refinement: Divide the problem into smaller problems (top-down methodology)

Figure 5.6 **The sequential search algorithm in pseudocode**

```
def Search (List, TargetValue):
    if (List is empty):
        Declare search a failure (“a miracle occurs”)
    else:
        Select the first entry in List to be TestEntry
        while (TargetValue > TestEntry and entries remain):
            Select the next entry in List as TestEntry
        if (TargetValue == TestEntry):
            Declare search a success
        else:
            Declare search a failure
```

Note, this algorithm is using Python primitives for decision looping and definition of a procedure.

Primitives

- Assignment

  `variable = value` (variable is assigned value)

- Decision

```
if variable < value:
   do something
elif variable > value:
   do something else
else:
   final alternative
```

Primitives

- Iteration

```
while variable < value:
   do something
   update variable
```

- Procedure

```
def foo(bar):
   do something
   return something
```