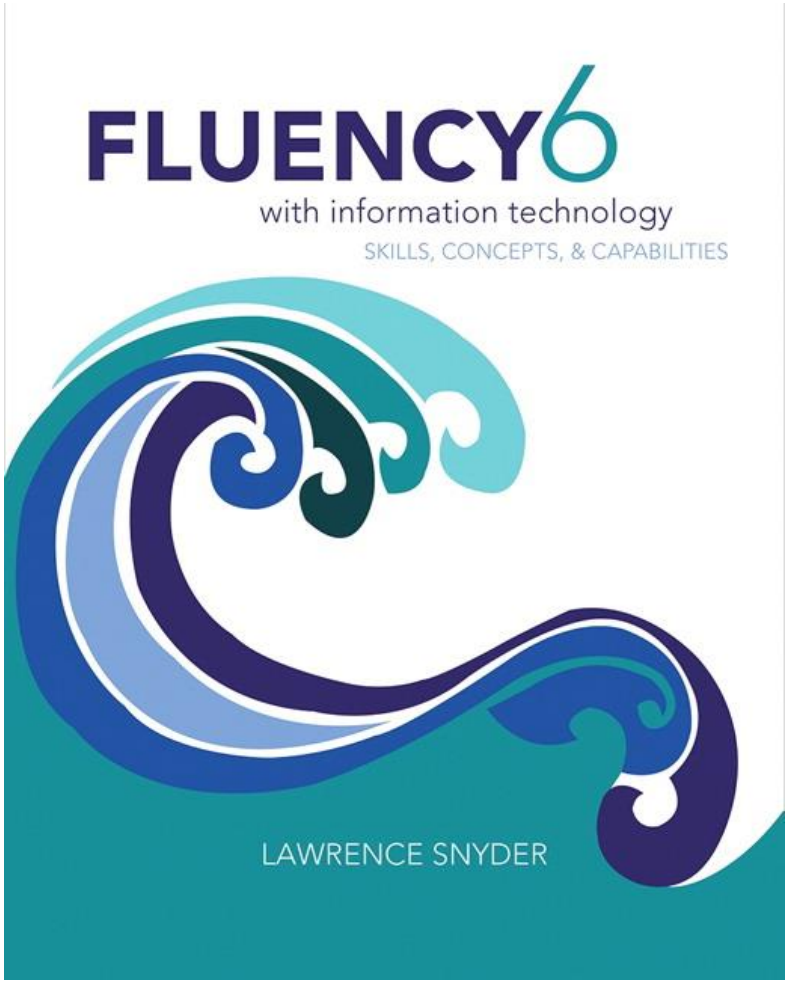


Chapter 10

Algorithmic Thinking

Table of Contents

Learning Objectives

- Explain similarities and differences among algorithms, programs, and heuristic solutions
- List the 5 essential properties of an algorithm
- Use the Intersect Alphabetized List algorithm to do the following:
  - Follow the flow of the instruction execution
  - Follow an analysis to pinpoint assumptions
- Demonstrate algorithmic thinking by being able to do the following:
  - Explain the importance of alphabetical order on the solution
  - Explain the importance of the barrier abstraction for correctness

Algorithms

- An algorithm is "a precise rule (or set of rules) specifying how to solve some problem." (thefreedictionary.com)
- Mohammed al-Khowarizmi (äl-khōwärēz´mē)

  Persian mathematician of the court of Mamun in Baghdad…the word *algorithm* is said to have been derived from his name. Much of the mathematical knowledge of medieval Europe was derived from Latin translations of his works. (encyclopedia.com)
- The study of algorithms is one of the foundations of computer science.

An algorithm is like a function

$$F(x) = y$$

Input

- Input specification
  - Recipes: ingredients, cooking utensils, …
  - Knitting: size of garment, length of yarn, needles …
  - Tax Code: wages, interest, tax withheld, …

- Input specification for computational algorithms:
  - What kind of data is required?
  - In what form will this data be received by the algorithm?

Computation

- An algorithm requires clear and precisely stated steps that express how to perform the operations to yield the desired results.
- Algorithms assume a basic set of ***primitive operations*** that are assumed to be understood by the executor of the algorithm.
  - Recipes: beat, stir, blend, bake, ...
  - Knitting: casting on, slip loop, draw yarn through, ...
  - Tax code: deduct, look up, check box, ...
  - Computational: add, set, modulo, output, ...

Output

- Output specification
  - Recipes: number of servings, how to serve
  - Knitting: final garment shape
  - Tax Code: tax due or tax refund, where to pay
- Output specification for computational algorithms:
  - What results are required?
  - How should these results be reported?
  - What happens if no results can be computed due to an error in the input? What do we output to indicate this?

Is this a "good" algorithm?

- Input: slices of bread, jar of peanut butter, jar of jelly

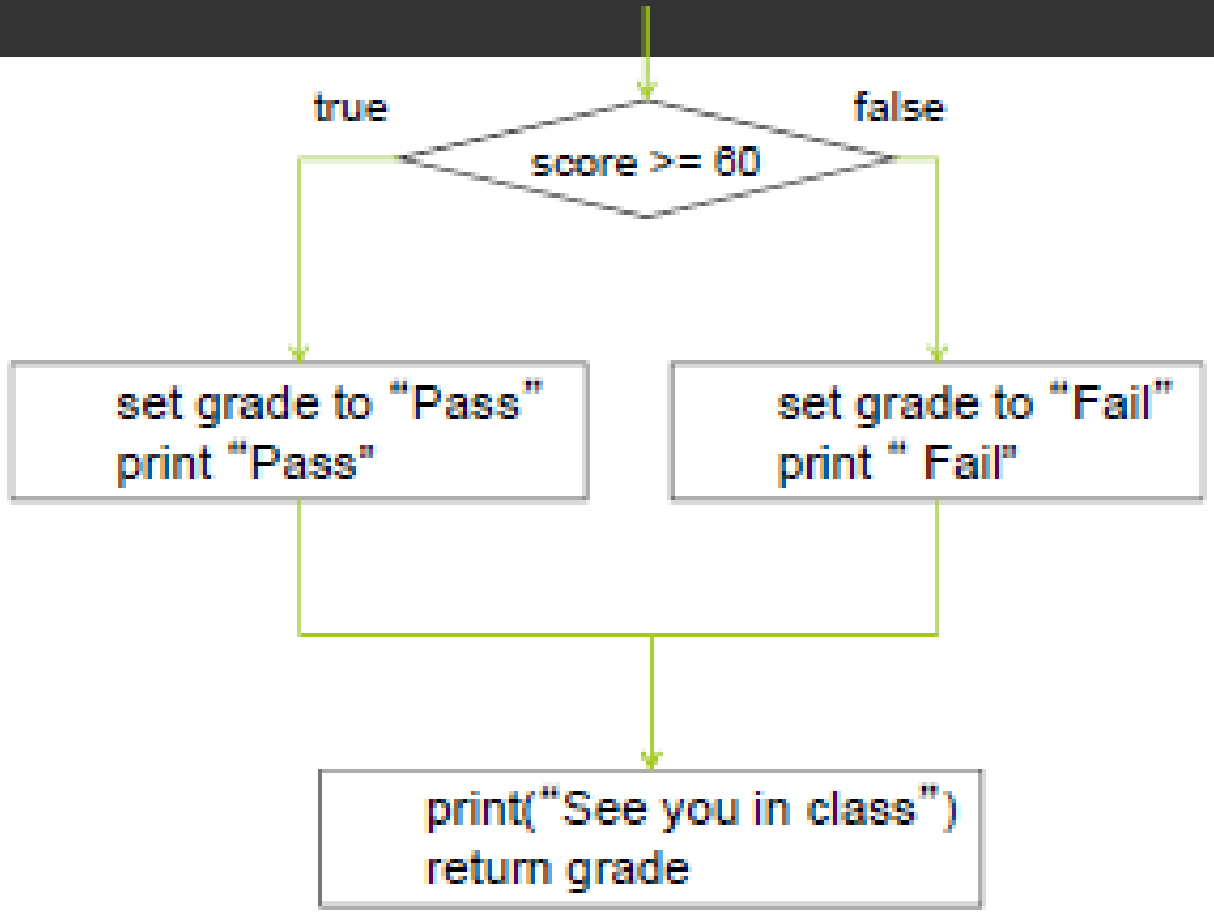
1. Pick up some bread.
2. Put peanut butter on the bread.
3. Pick up some more bread.
4. Open the jar of jelly.
5. Spread the jelly on the bread.
6. Put the bread together to make your sandwich.

- Output?

What makes a "good" algorithm?

- A good algorithm should produce the **correct outputs for any set of legal inputs.**
- A good algorithm should **execute efficiently with the fewest number of steps as possible.**
- A good algorithm should be designed in such a way that **others will be able to understand it and modify it to specify solutions to additional problems.**

Control Flow



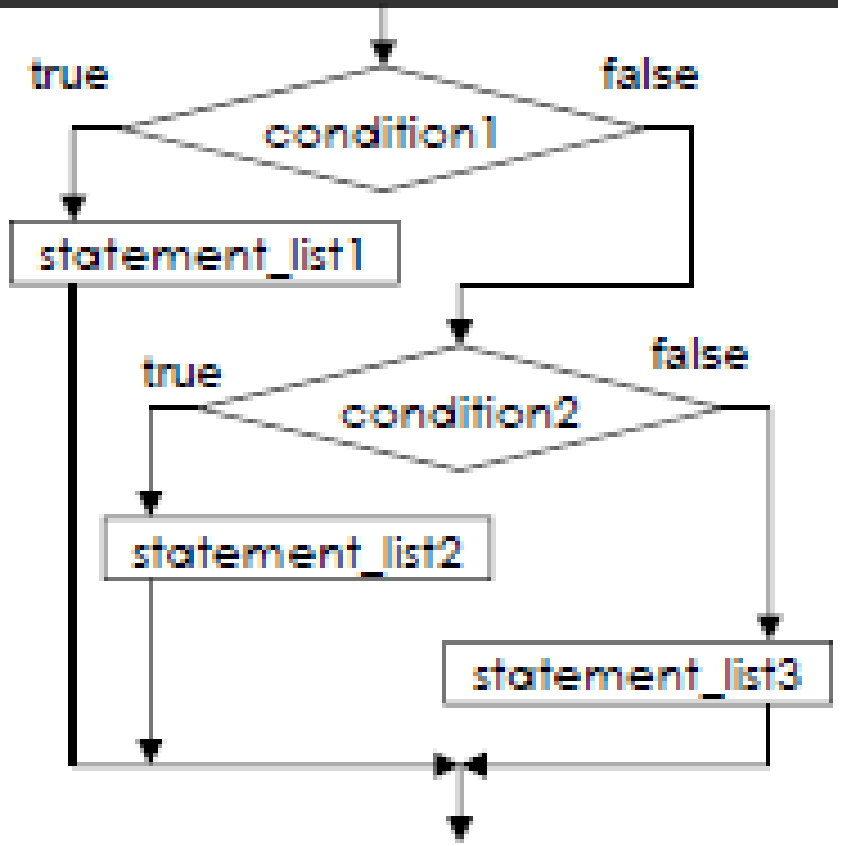
Flow chart: `if/elif/else` statement

Format:

```
if condition1 :
      statement_list1

elif condition2 :
      statement_list2
else:

      statement_list3
```



Coding the Grader in Python

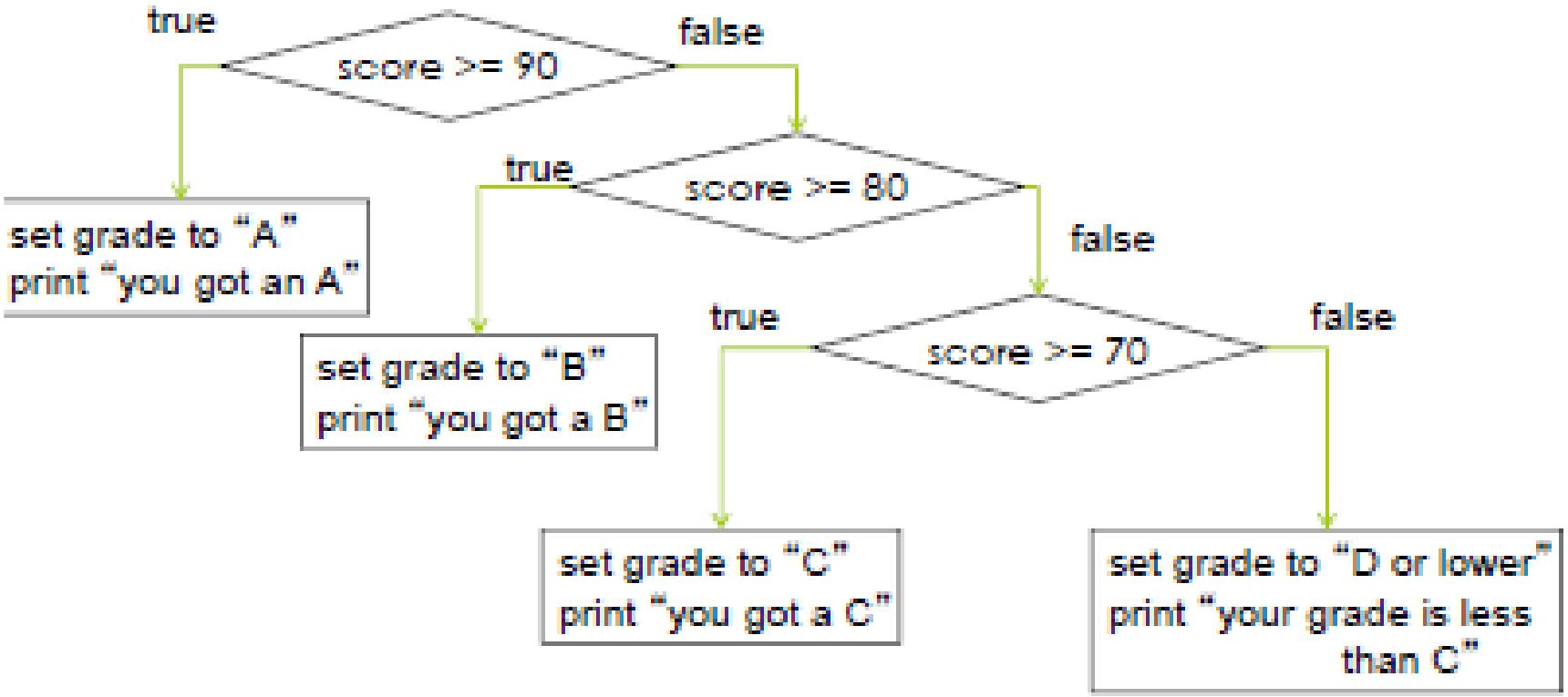
Algorithm:

1. If *score* >= 60
   a. Set *grade* to "Pass"
   b. Print " Pass"
2. Otherwise,
   a. Set *grade* to "Fail"
   b. Print "Fai"
3. Print "See you in class "
4. Return *grade*

```
def grader(score):
    if score >= 60:
        grade = "Pass"
        print("Pass")
    else:
        grade = "Fail"
        print("Fail")
    print("See you in class")
    return grade
```

Grader for Letter Grades

ALGORITHM



Equivalently

PYTHON PROGRAMMING

```
def grader3(score):
    if score >= 90:
        grade = "A"
        print("You got an A")
    elif score >= 80:
        grade = "B"
        print("You got a B")
    elif score >= 70:
        grade = "C"
        print("You got a C")
    else:
        grade = "D or lower"
        print("Your grade is less than C")
    return grade
```

Everyday Situations need Problem Solving

- The movie: The Driving Bell and the Butterfly
- The man Jean-Dominique Baudy
  - His body is paralyzed, but his mind is active
  - Only blinking his left eye, but he wants to write a book…. How to solve it?

Figure 10.1 Jean-Dominique Bauby "dictates" to his assistant Cla
Mendibil.

Figure 10.2 Anne Consigny as Claude Mendibil, J-DB's assistant, in the movie *The Diving Bell and the Butterfly*. She shows J-DB the alphabet; when she points to the right letter, he blinks his left eyelid.

Important Points about Algorithms

- Algorithm: a precise, systematic method for producing a specified result
- Everyone uses and invents algorithms all the time to solve problems
  - The agent running the algorithm does not need to be a computer
  - There are better and worse versions of many algorithms
- Programs are algorithms that have been specialized to a specific set of conditions and assumptions usually written in a specific programming language
  - Usually the words "program" and "algorithm" are used interchangeably

Algorithms we have learned

- Placeholder technique (ch 2)
  - protect “good” letter sequences by replacing them with a placeholder
  - edit the “bad” letter sequences
  - restore the “good” letter sequences
- Binary to Decimal Conversion (ch 7)
  - if there is a 1, write down the place value for its position in decimal
  - add up those place values
- Binary Addition (ch 8)
  - add as in decimal but limit digit positions to less than two

Algorithms vs. Heuristic Processes

- Not all processes given in the book are algorithms
- The process to find information on the web using a search engine was not an algorithm
  - not systematic
  - not guaranteed to find it (process could fail)
  - called a heuristic process: helpful procedure for finding a result
- Algorithm Properties
  - Input specified / Output specified
  - Definiteness
  - Effectiveness
  - Finiteness

1. Input Specified

- The input is the data to be transformed during the computation to produce the output
- What data do you need to begin to get the result you want?
- Input precision requires that you know what kind of data, how much and what form the data should be

2. Output Specified

- The output is the data resulting from the computation (your intended result)
- Frequently the name of the algorithm contains the output:
  - "Algorithm to compute batting average"
- Output precision also requires that you know what kind of data, how much and what form the output should be (or even if there will be any output at all!)

3. Definiteness

- Algorithms must specify every step and the order the steps must be taken in the process
  - Details of each step must be spelled out (including how to handle errors)
- Definiteness ensures that if the algorithm is performed at different times or different computers, the output is same

4. Effectiveness

- For an algorithm to be effective, all of its steps must be doable
- The computer could execute the algorithm mechanically without any further input, special talent, clairvoyance, etc

5. Finiteness

- The algorithm must stop, eventually!
  - Computer algorithms often repeat instructions with different data and finiteness may be a problem
- Stopping may mean that either you get the expected output or you get a response that no solution is possible
  - Finiteness is not usually an issue for non-computer algorithms

Algorithm Facts

- [1] Algorithms can be specified at different levels of detail
  - Algorithms use functions to simplify the algorithmic description
  - These functions (such as scan) may have their own algorithms
- [2] Algorithms always build on functionality previously defined and known to the user
  - Assume the familiar functions and algorithms
  - Ex: Searching an element in a sorted list is faster than searching an element in a unsorted list
- [3] Different algorithms can solve the same problem differently, and the different solutions can take different amounts of time (or space)

Query Evaluation of Web Search Engine

- Larry Page and Sergey Brin "Page Rank Algorithm"
  - The query processor (web search engine!) makes an ordered list of the pages that hit on the keywords of your search query (keywords!)
- The algorithm in Figure 10.4 is not written in a programming language, but does use "tech speak" instead just everyday English
  - For technically trained people
- The actual program (SW) would be too detailed to explain to people how it works
- Essentially the query processor is executing the intersect an alphabetized list (IAL) algorithm

1. Parse the query.
2. Convert words into wordIDs.
3. Seek to the start of the doclist in the short barrel for every word.
4. Scan through the doclists until there is a document that matches all the search terms.
5. Compute the rank of that document for the query.
6. If we are in the short barrels and at the end of any doclist, seek to the start of the doclist in the full barrel for every word and go to step 4.
7. If we are not at the end of any doclist go to step 4.

Sort the documents that have matched by rank and return the top k.

Figure 4. Google Query Evaluation

Figure 10.4 Original query evaluation algorithm developed by Brin and Page to find the hits for a Google search; from their original paper, The Anatomy of a Large-Scale Hypertextual Web Search Engine (infolab.stanford.edu/~backrub/google.html).

Different Levels of Details and Pre-defined Functions

Repeating the algorithm and figure from Chapter 5, and connecting it with the Brin/Page version shown in Figure 10.4, we have:

1. Put a marker such as an arrow at the start of each token's index list.	*This is B/P Step 3.*
2. If all markers point to the same URL, include it in the hit list, because all tokens are associated with the page.	*This is B/P Step 4; they then compute page rank (Step 5) for a hit and, when necessary, switch to full lists (Step 6).*
3. Move the marker(s) to the next position for whichever URL is earliest in the alphabet.	*B/P don't have this step; it is the "scan" that makes the algorithm advance through the lists efficiently.*
4. Repeat Steps 2–3 until some marker reaches the end of the list, then quit.	*This is B/P Step 7.*

** B/P means Brin and Page

Intersecting Alphabetalized List (IAL) Execution

Figure 10.5 Progress of the Intersecting Alphabetized List computation for six keywords. For columns A through F, the activity described by each row is: (a) initial configuration, (b) pass amiss.com, (c) pass bmiss.com, (d) pass cmiss.com, (e) pass dmiss.com, (f) pass emiss.com, (g) pass fmiss.com.

** marker가 지명하는 string중에 가장 작은값의 string으로 모든 list를 확인
** 6개 list에 전부 같은 string이 있으면 그 string은 AnswerSet으로 이동
** 작은값이 있었던 List의 marker만 1칸을 progress

What if non-sorted URL lists?

How Not to Match

Another way to implement "scan" to find the needed matches—we call it No Alphabetized Lists (NAL)—starts with any lists and works like an odometer. (The lists can be alphabetized or not; the algorithm will ignore the ordering.) It places the arrow pointers at the start of each list, and then advances the pointer of one of the lists all the way to the end looking for a match at each step; it then advances the next list one position, and repeats what it has done up to this point. (Think of the odometer as starting at 00000 and finding the 10 hits, 00000, 11111, . . . , 99999.) This is an algorithm. It checks every combination of URLs, and so it finds the identical information that the IAL does. But it's MUCH slower.

How much slower? An example will convince you that it's a lot slower. Consider five 10-item lists. The IAL moves the pointer past each item of each list *once*, giving a worst-case number of steps as

IAL: 10 + 10 + 10 + 10 + 10 = 50

(We don't know what order the pointers will be advanced in, but it doesn't matter when we're only counting up how many advancements there are: At most ten for each list.)

But the non-alphabetized lists solution repeatedly visits the same items. Because it works like an odometer, our sample lists will require as many steps as sweeping through a five-digit odometer:

NAL: 10 × 10 × 10 × 10 × 10 = 100,000

Both solutions work. One solution is very efficient compared to the other.

How Do We Know it Works?

- Algorithm solution should be clear, simple and efficient
- Then, how do we know it works? ➔ Correctness!
  - If there is no loop, the program gets to an end, and we can check the result
  - What if there is a loop? (too many possible cases)
    - Programs with loops cannot be absolutely verified that it works
- The way to know that an algorithm works is to know why it works.
  - Find the algorithm's correctness-preserving properties
  - Explain, using the program, why they make it work

Summary

- We use algorithms daily, but everyday algorithms can be sometimes be unclear because natural language is imprecise
- Problems can be solved by different algorithms in different ways
- Algorithms can be given at different levels of detail depending on the abilities of the agent
- Computer algorithms should have five fundamental properties
- Algorithms should always work — either they give the answer, or say no answer is possible — and they are evaluated on their use of resources such as space and time
- Intersect Alphabetized Lists algorithm is used in Web search, and is preferred over other solutions
- Two properties of the Intersect Alphabetized Lists algorithm tell us why it works: the Alpha-Ordering and Barrier properties