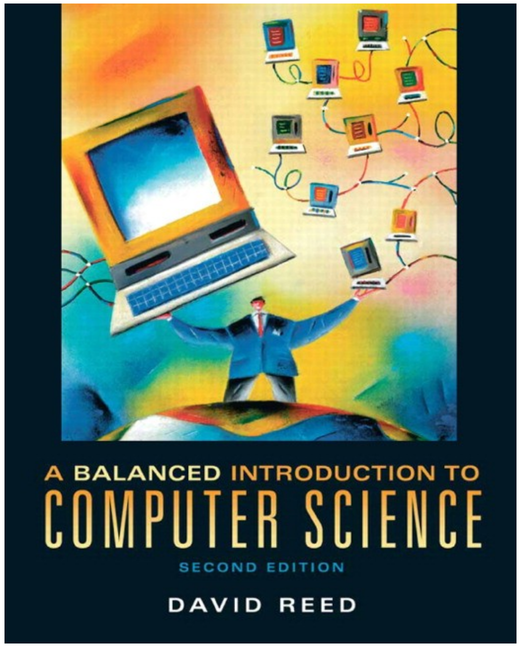
A Balanced Introduction to Computer Science, 2/E

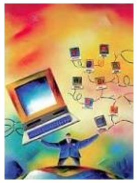
David Reed, Creighton University

©2008 Pearson Prentice Hall
ISBN 978-0-13-601722-6



Chapter 8
Algorithms and Programming Languages

Algorithms



the central concept underlying all computation is that of the *algorithm*

- an algorithm is a step-by-step sequence of instructions for carrying out some task

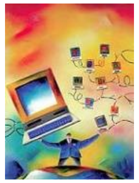
programming can be viewed as the process of designing and implementing algorithms that a computer can carry out

- a programmer's job is to:
  - create an algorithm for accomplishing a given objective, then
  - translate the individual steps of the algorithm into a programming language that the computer can understand

example: programming in JavaScript

- we have written programs that instruct the browser to carry out a particular task
- given the proper instructions, the browser is able to understand and produce the desired results

Algorithms in the Real World

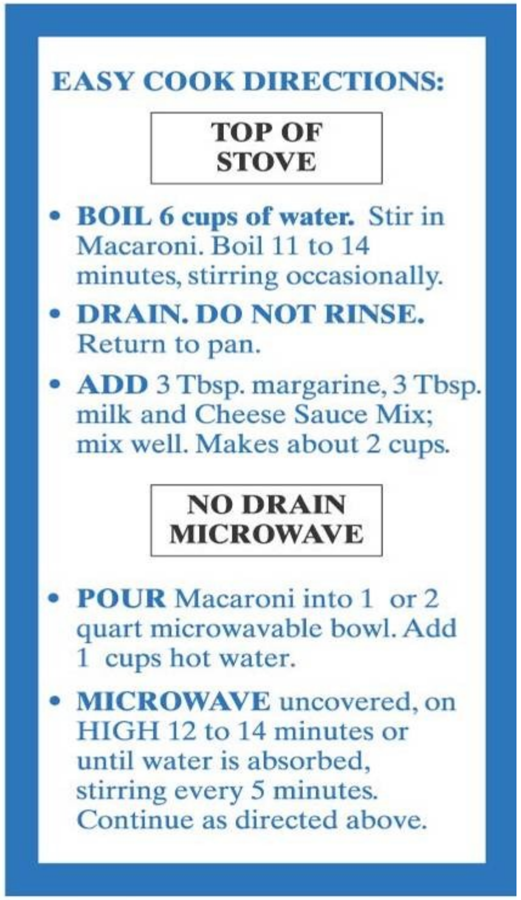


the use of algorithms is not limited to the domain of computing

- e.g., recipes for baking cookies
- e.g., directions to your house

there are many unfamiliar tasks in life that we could not complete without the aid of instructions

- in order for an algorithm to be effective, it must be stated in a manner that its intended executor can understand
    - a recipe written for a master chef will look different than a recipe written for a college student
- as you have already experienced, computers are more demanding with regard to algorithm specifics than any human could be



EASY COOK DIRECTIONS:

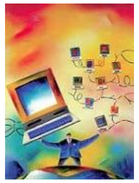
TOP OF STOVE

- **BOIL 6 cups of water.** Stir in Macaroni. Boil 11 to 14 minutes, stirring occasionally.
- **DRAIN. DO NOT RINSE.** Return to pan.
- **ADD** 3 Tbsp. margarine, 3 Tbsp. milk and Cheese Sauce Mix; mix well. Makes about 2 cups.

NO DRAIN MICROWAVE

- **POUR** Macaroni into 1 or 2 quart microwavable bowl. Add 1 cups hot water.
- **MICROWAVE** uncovered, on HIGH 12 to 14 minutes or until water is absorbed, stirring every 5 minutes. Continue as directed above.

Designing & Analyzing Algorithms



4 steps to solving problems (George Polya)

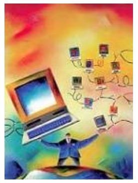
1. understand the problem
2. devise a plan
3. carry out your plan
4. examine the solution

EXAMPLE: finding the oldest person in a room full of people

1. understanding the problem
   - initial condition – room full of people
   - goal – identify the oldest person
   - assumptions
     - ✓ a person will give their real birthday
     - ✓ if two people are born on the same day, they are the same age
     - ✓ if there is more than one oldest person, finding any one of them is okay

1. we will consider 2 different designs for solving this problem

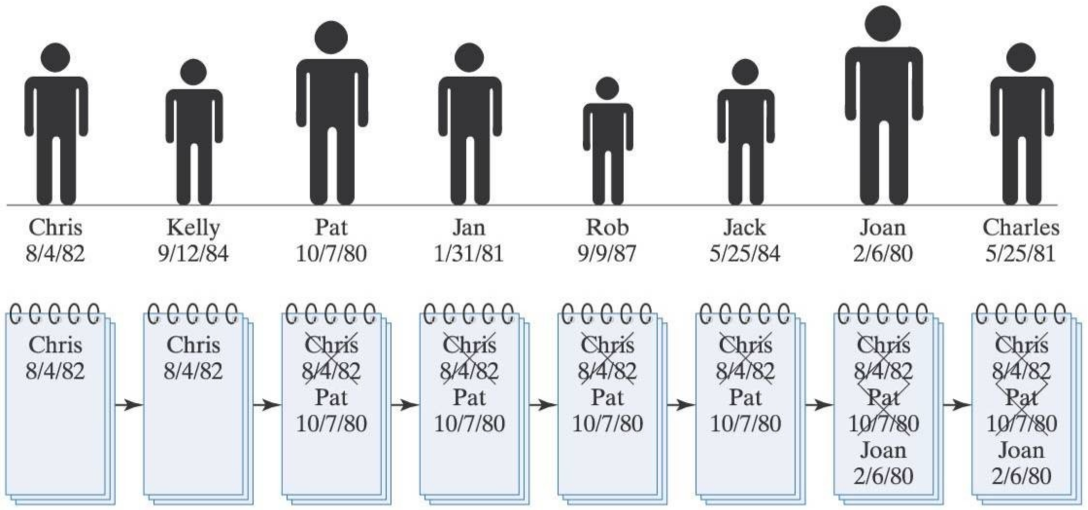
Algorithm 1



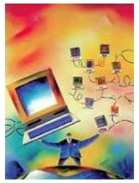
Finding the oldest person (algorithm 1)

1. line up all the people along one wall
2. ask the first person to state his or her name and birthday, then write this information down on a piece of paper
3. for each successive person in line:
   i. ask the person for his or her name and birthday
   ii. if the stated birthday is earlier than the birthday on the paper, cross out old information and write down the name and birthday of this person

when you reach the end of the line, the name and birthday of the oldest person will be written on the paper


Chris 8/4/82
Kelly 9/12/84
Pat 10/7/80
Jan 1/31/81
Rob 9/9/87
Jack 5/25/84
Joan 2/6/80
Charles 5/25/81
Chris 8/4/82
Chris 8/4/82
~~Chris 8/4/82~~ Pat 10/7/80
~~Chris 8/4/82~~ Pat 10/7/80
~~Chris 8/4/82~~ Pat 10/7/80
~~Chris 8/4/82~~ Pat 10/7/80
~~Chris 8/4/82~~ ~~Pat 10/7/80~~ Joan 2/6/80
~~Chris 8/4/82~~ ~~Pat 10/7/80~~ Joan 2/6/80

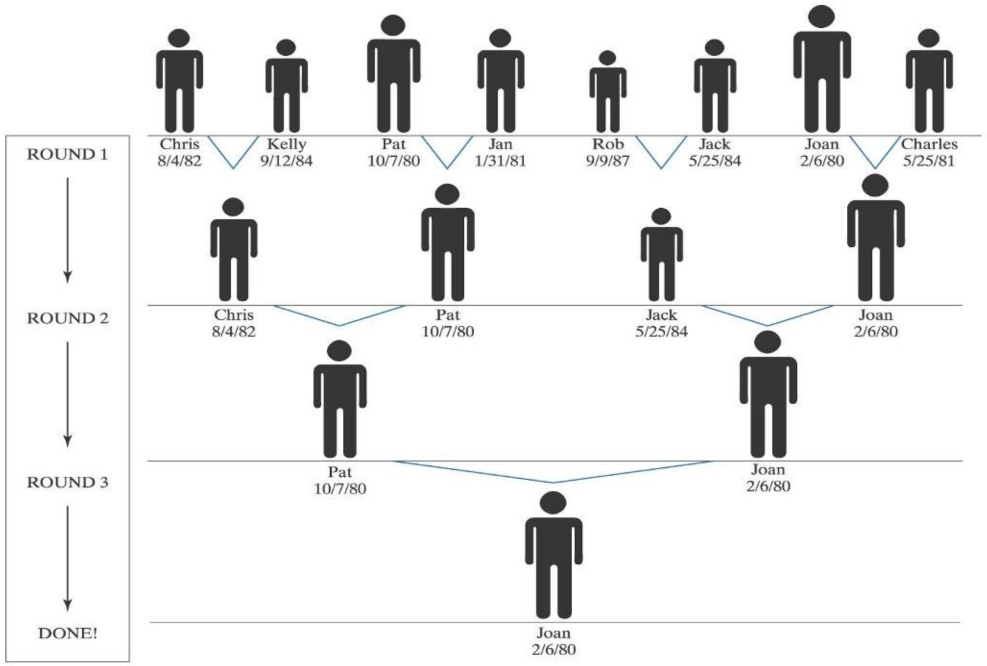
Algorithm 2



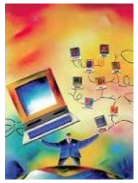
Finding the oldest person (algorithm 2)

1. line up all the people along one wall
2. as long as there is more than one person in the line, repeatedly
   1. have the people pair up (1st with 2nd, 3rd with 4th, etc) – if there is an odd number of people, the last person will be without a partner
   2. ask each pair of people to compare their birthdays
   3. request that the younger of the two leave the line

when there is only one person left in line, that person is the oldest



Algorithm Analysis



determining which algorithm is "better" is not always clear cut

- it depends upon what features are most important to you
  - if you want to be sure it works, choose the clearer algorithm
  - if you care about the time or effort required, need to analyze performance

algorithm 1 involves asking each person's birthday and then comparing it to the birthday written on the page

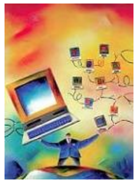
- the amount of time to find the oldest person is *proportional to the number of people*
- if you double the amount of people, the time needed to find the oldest person will also double

algorithm 2 allows you to perform multiple comparisons simultaneously

- the time needed to find the oldest person is *proportional to the number of rounds it takes to shrink the line down to one person*
  - which turns out to be the logarithm (base 2) of the number of people
- if you double the amount of people, the time needed to find the oldest person increases by the cost of one more comparison

the words *algorithm* and *logarithm* are similar – do not be confused by this
algorithm: a step-by-step sequence of instructions for carrying out a task
logarithm: the exponent to which a base is raised to produce a number
e.g., $2^{10} = 1024$, so $\log_2(1024) = 10$

Algorithm Analysis (cont.)



when the problem size is large, performance differences can be dramatic

for example, assume it takes 5 seconds to compare birthdays

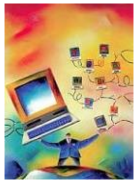
- for algorithm 1:
  - 100 people → 5*100 = 500 seconds
  - 200 people → 5*200 = 1000 seconds
  - 400 people → 5*400 = 2000 seconds

    . . .
  - 1,000,000 people → 5*1,000,000 = 5,000,000 seconds
- for algorithm 2:
  - 100 people → 5*$\lceil \log_2 100 \rceil$ = 35 seconds
  - 200 people → 5*$\lceil \log_2 200 \rceil$ = 40 seconds
  - 400 people → 5*$\lceil \log_2 400 \rceil$ = 45 seconds

    . . .
  - 1,000,000 people → 5*$\lceil \log_2 1{,}000{,}000 \rceil$ = 100 secor

N	$\lceil \log_2(N) \rceil$
100	7
200	8
400	9
800	10
1,600	11
. . .	. . .
10,000	14
20,000	15
40,000	16
. . .	. . .
1,000,000	20

Big-Oh Notation



to represent an algorithm's performance in relation to the size of the problem, computer scientists use what is known as *Big-Oh* notation

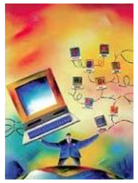
- executing an O(N) algorithm requires time proportional to the size of problem
  - given an O(N) algorithm, doubling the problem size doubles the work

- executing an O(log N) algorithm requires time proportional to the logarithm of the problem size
  - given an O(log N) algorithm, doubling the problem size adds a constant amount of work

based on our previous analysis:

- algorithm 1 is classified as O(N)
- algorithm 2 is O(log N)

Another Algorithm Example



SEARCHING: a common problem in computer science involves storing and maintaining large amounts of data, and then searching the data for particular values

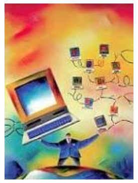
- data storage and retrieval are key to many industry applications
- search algorithms are necessary to storing and retrieving data efficiently

- e.g., consider searching a large payroll database for a particular record
  - if the computer selected entries at random, there is no assurance that the particular record will be found
  - even if the record is found, it is likely to take a large amount of time
  - a systematic approach assures that a given record will be found, and that it will be found more efficiently

there are two commonly used algorithms for searching a list of items

- sequential search – general purpose, but relatively slow
- binary search – restricted use, but fast

Sequential Search



sequential search is an algorithm that involves examining each list item in sequential order until the desired item is found

<u>sequential search for finding an item in a list</u>

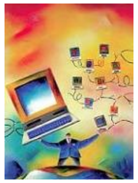
1. start at the beginning of the list
2. for each item in the list
    1. examine the item - if that item is the one you are seeking, then you are done
    2. if it is not the item you are seeking, then go on to the next item in the list

if you reach the end of the list and have not found the item, then it was not in the list

sequential search guarantees that you will find the item if it is in the list

- but it is not very practical for very large databases
- *worst case:* you may have to look at every entry in the list

Binary Search



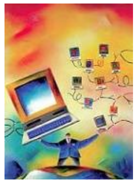
binary search involves continually cutting the desired search list in half until the item is found

- the algorithm is only applicable if the list is ordered
  - e.g., a list of numbers in increasing order
  - e.g., a list of words in alphabetical order

<u>binary search for finding an item in an ordered list</u>

1. initially, the potential range in which the item could occur is the entire list
2. as long as items remain in the potential range and the desired item has not been found, repeatedly
   i. examine at the middle entry in the potential range
   ii. if the middle entry is the item you are looking for, then you are done
   iii. if the middle entry is greater than the desired item, then reduce the potential range to those entries left of the middle
   iv. if the middle entry is less than the desired item, then reduce the potential range to those entries right of the middle

by repeatedly cutting the potential range in half, binary search can hone in on the value very quickly



Binary Search Example

suppose you have a sorted list of state names, and want to find *Illinois*

1. start by examining the middle entry (*Missouri*)

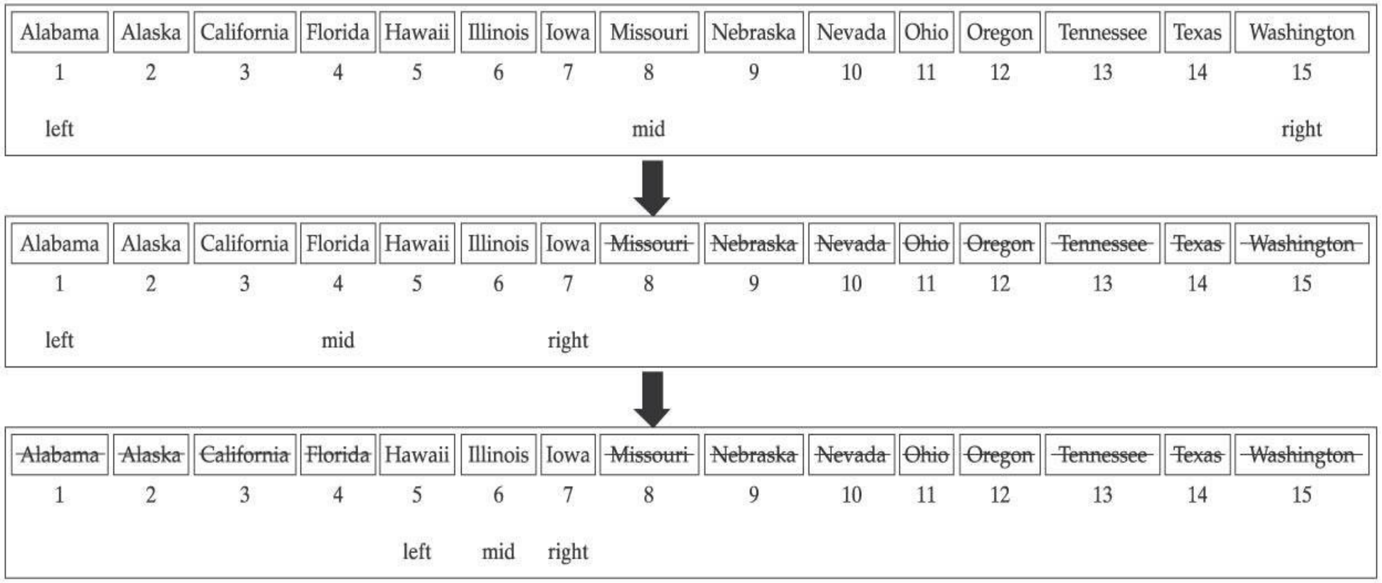
   since *Missouri* comes after *Illinois* alphabetically, can eliminate it and all entries that appear to the right

1. next, examine the middle of the remaining entries (*Florida*)

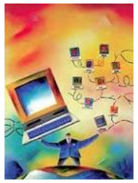
   since *Florida* comes before *Illinois* alphabetically, can eliminate it and all entries that appear to the left

1. next, examine the middle of the remaining entries (*Illinois*)

   the desired entry is found


Alabama Alaska California Florida Hawaii Illinois Iowa Missouri Nebraska Nevada Ohio Oregon Tennessee Texas Washington
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
left mid right
Alabama Alaska California Florida Hawaii Illinois Iowa ~~Missouri~~ ~~Nebraska~~ ~~Nevada~~ ~~Ohio~~ ~~Oregon~~ ~~Tennessee~~ ~~Texas~~ ~~Washington~~
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
left mid right
~~Alabama~~ ~~Alaska~~ ~~California~~ ~~Florida~~ Hawaii Illinois Iowa ~~Missouri~~ ~~Nebraska~~ ~~Nevada~~ ~~Ohio~~ ~~Oregon~~ ~~Tennessee~~ ~~Texas~~ ~~Washington~~
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
left mid right

Search Analysis



sequential search

- in the worst case, the item you are looking for is in the last spot in the list (or not in the list at all)
  - as a result, you will have to inspect and compare every entry in the list
- the amount of work required is proportional to the list size

  → sequential search is an O(N) algorithm

binary search

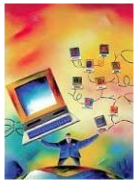
- in the worst case, you will have to keep halving the list until it gets down to a single entry
  - each time you inspect/compare an entry, you rule out roughly half the remaining entries
- the amount of work required is proportional to the logarithm of the list size

  → binary search is an O(log N) algorithm

imagine searching a phone book of the United States (300 million people)

- sequential search requires at most 300 million inspections/comparisons
- binary search requires at most $\lceil \log_2(300,000,000) \rceil = 29$ inspections/comparisons

Another Algorithm Example

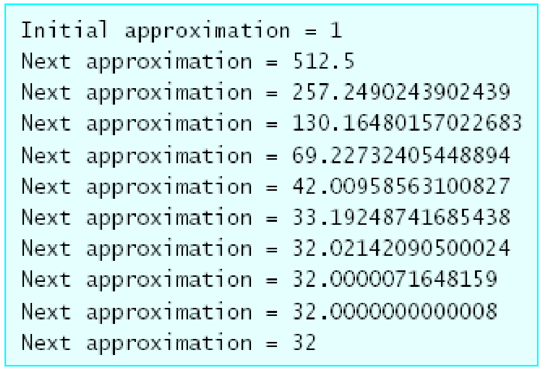


<u>Newton's Algorithm for finding the square root of N</u>

1. start with an initial approximation of 1
2. as long as the approximation isn't close enough, repeatedly
   i. refine the approximation using the formula:

      newApproximation = (oldApproximation + N/oldApproximation)/2

example: finding the square root of 1024



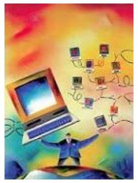
```
Initial approximation = 1
Next approximation = 512.5
Next approximation = 257.2490243902439
Next approximation = 130.16480157022683
Next approximation = 69.22732405448894
Next approximation = 42.00958563100827
Next approximation = 33.19248741685438
Next approximation = 32.02142090500024
Next approximation = 32.0000071648159
Next approximation = 32.0000000000008
Next approximation = 32
```

algorithm analysis:

- Newton's Algorithm does converge on the square root because each successive approximation is closer than the previous one
  - however, since the square root might be a non-terminating fraction it is difficult to define the exact number of steps for convergence
- in general, the difference between the given approximation and the actual square root is roughly cut in half by each successive refinement

  → demonstrates O(log N) behavior

Algorithms and Programming



programming is all about designing and coding algorithms for solving problems

- the intended executor is the computer or a program executing on that computer
- instructions are written in programming languages which are more constrained and exact than human languages

the level of precision necessary to write programs can be frustrating to beginners

- but it is much easier than it was 50 years ago
- early computers (ENIAC) needed to be wired to perform computations
- with the advent of the von Neumann architecture, computers could be programmed instead of rewired
  - an algorithm could be coded as instructions, loaded into the memory of the computer, and executed