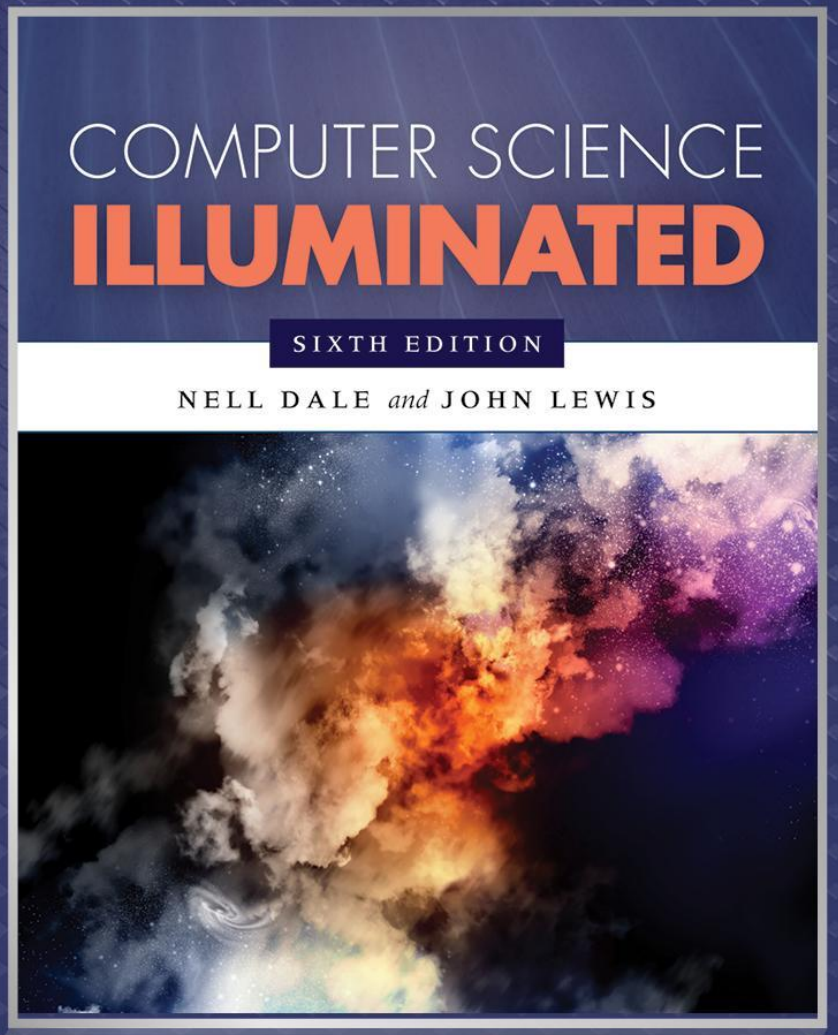
Limitations of Computing



Edits by Michael Edwards

Chapter Goals

- Describe the limits that the hardware places on the solution to computing problems
- Discuss how the finiteness of the computer impacts the solutions to numeric problems
- Discuss ways to ensure that errors in data transmission are detected
- Describe the limits that the software places on the solutions to computing problems

Chapter Goals

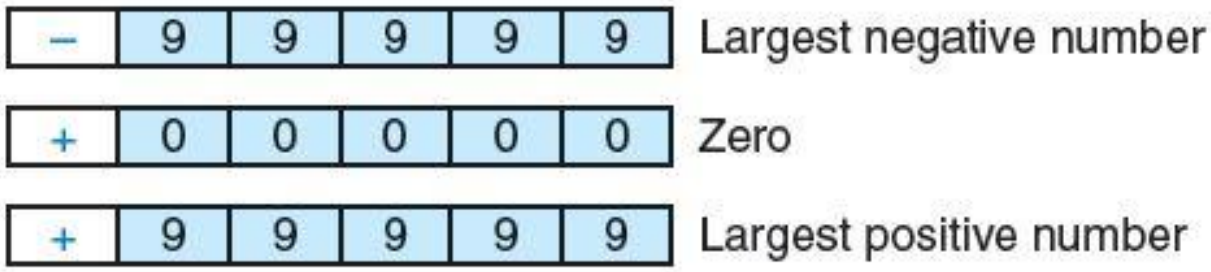
- Discuss ways to build better software
- Describe the limits inherent in computable problems themselves
- Discuss the continuum of problem complexity from problems in Class P to problems that are unsolvable

Limits on Arithmetic

Precision

The maximum number of significant digits that can be represented

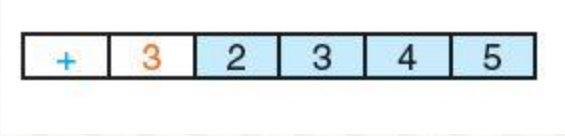
With 5 digits precision, the range of the numbers we can represent is -99,999 through +99,999



Limits on Arithmetic

What happens if we allow one of these digits (let's say the leftmost one, in red) to represent an exponent?

For example



represents the number $+2{,}345 * 10^3$

Limits on Arithmetic

The **range** of numbers we can now represent is much larger

$$-9{,}999 * 10^9 \text{ to } +9{,}999 * 10^9$$

but we can represent only **four significant digits**

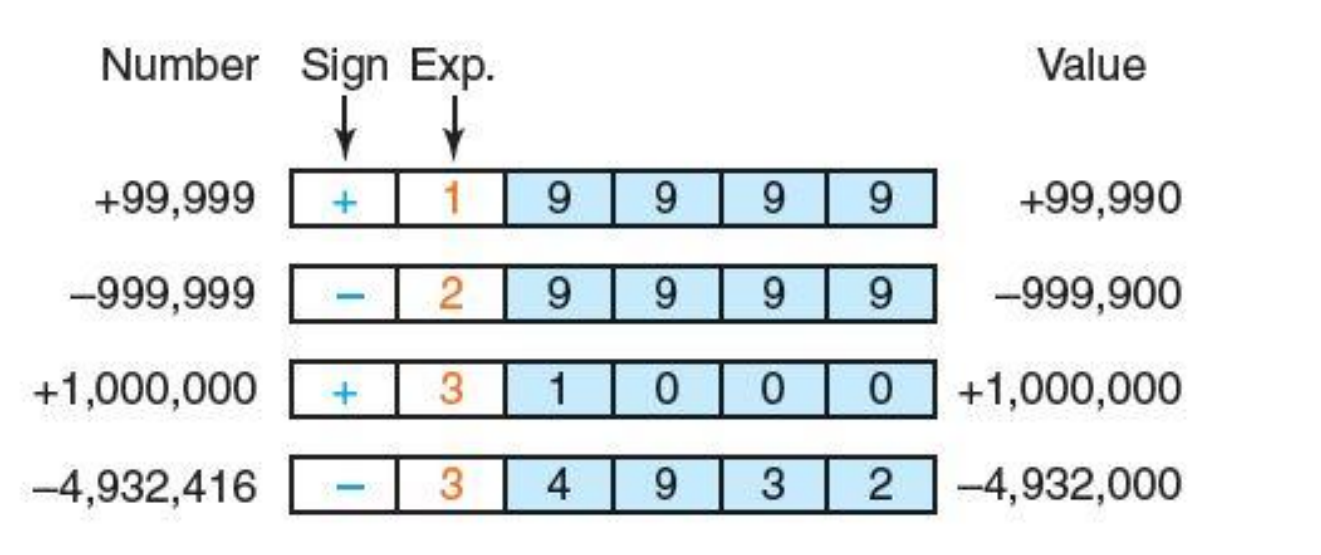
Significant digits

Those digits that begin with the first nonzero digit on the left and end with the last nonzero digit on the right

Limits on Arithmetic

The four leftmost digits are correct, and the balance of the digits are assumed to be zero

We lose the rightmost, or *least significant*, digits



Limits on Arithmetic

To represent real numbers, we extend our coding scheme to represent negative exponents

For example

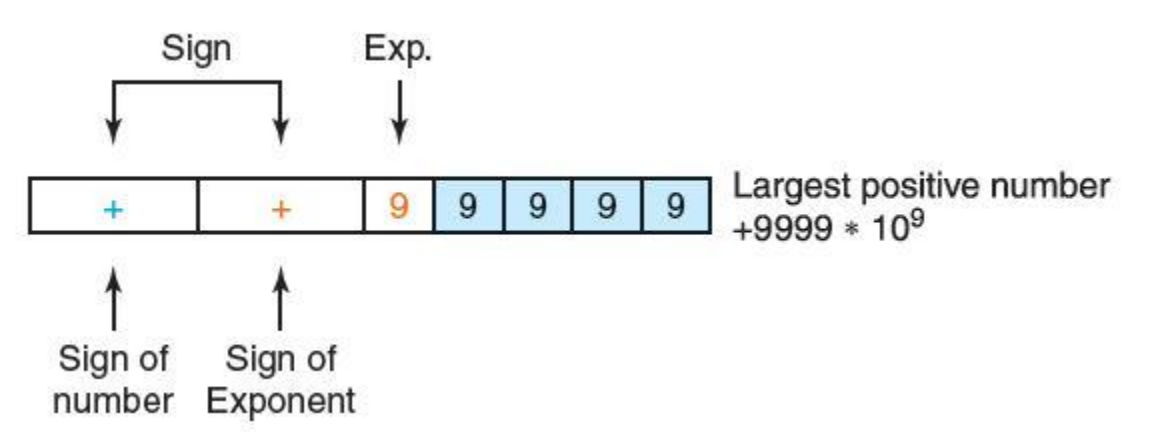
$$4{,}394 * 10^{-2} = 43.94$$

or

$$22 * 10^{-4} = 0.0022$$

Limits on Arithmetic

Let the current sign be the sign of the exponent and add a sign to the left to be the sign of the number itself



What is the largest negative number?
The smallest positive number?
The smallest negative number?

Limits on Arithmetic

Representational error or **round-off error**

An arithmetic error caused by the fact that the precision of the result of an arithmetic operation is greater than the precision of the machine

Underflow

Results of a calculation are too small to represent in a given machine

Overflow

Results of a calculation are too large to represent in a given machine

Cancellation error

A loss of significance during addition or subtraction of numbers of similar magnitude, due to the limits of precision

Give examples of each of these errors

Representational error or *round-off error*

An arithmetic error caused by the fact that the precision of the result of an arithmetic operation is greater than the precision of the machine.

Example:

π = 3.141592653589…

On a machine with 6 significant digit precision:

π = 3.14159

A rounding error of 0.000002653589…

Underflow

Results of a calculation are too small to represent in a given machine.

The value is too close to zero.

Example:

0.000000001 * 0.1 = 0.0000000001 = 1 * 10^-10

However in a machine that can only handle 9 decimal points (1 digit exponent):

0.000000001 * 0.1 = 0

Overflow

Results of a calculation are too large to represent in a given machine.

Example:

Add 1 to the maximum 7 bit binary number:

1111111
0000001 +
10000000

Equals 0, rather than 128 in a 7 bit representation

Cancellation

Loss of significance from calculation involving values of similar magnitude.

Example Subtract the following two numbers:

	12345671×10^{-8}	←8 significant digits
-	12345670×10^{-8}	←8 significant digits
	1×10^{-8}	← Only 1 significant digit

Limits on Arithmetic

There are limitations imposed by the hardware on the representations of both integer numbers and real numbers

- If the word length is 32 bits, the range of integer numbers that can be represented is 22,147,483,648 to 2,147,483,647
- There are software solutions, however, that allow programs to overcome these limitations
- For example, we could represent a very large number as a list of smaller numbers

Limits on Arithmetic

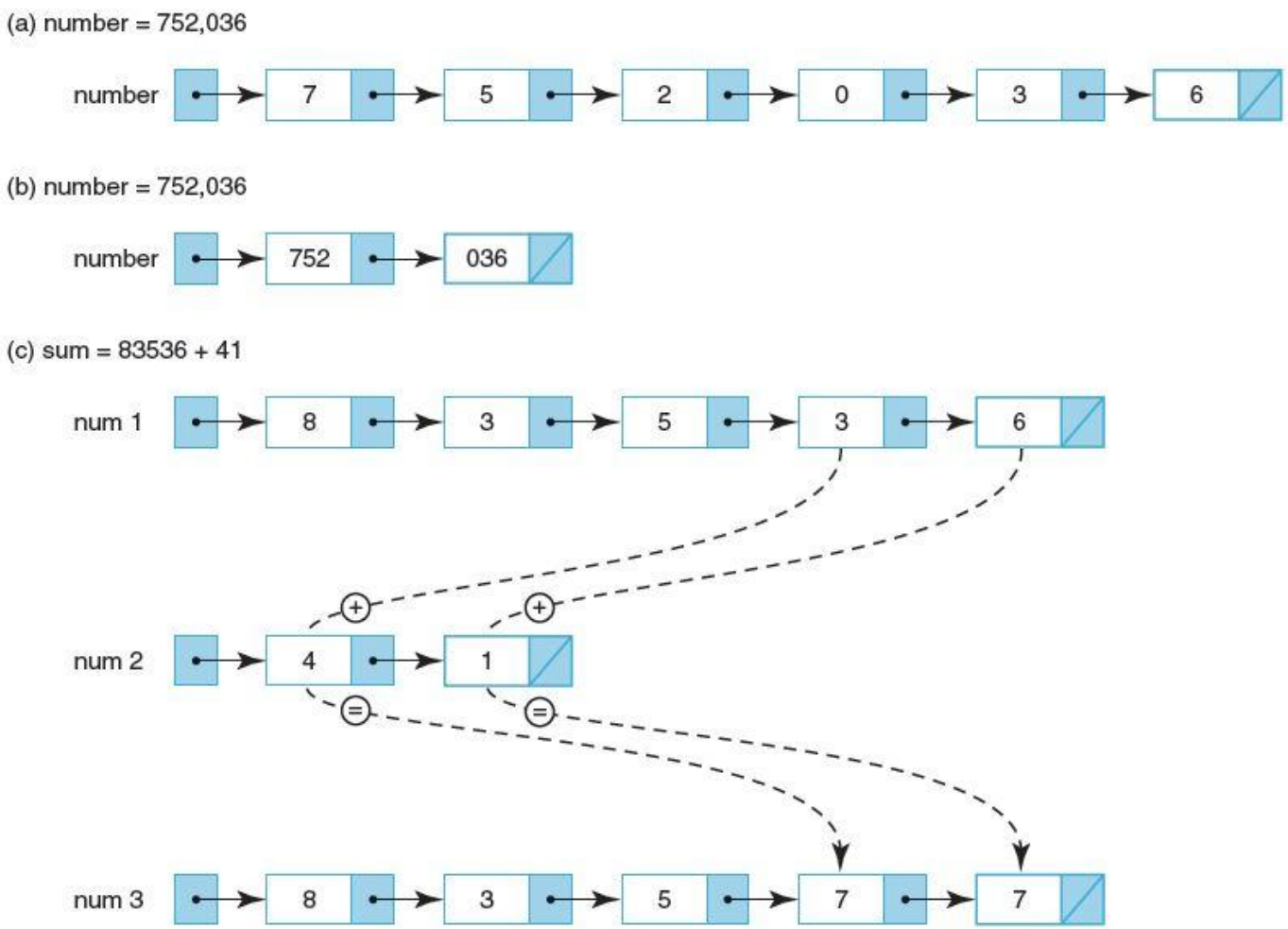


FIGURE 18.1 Representing very large numbers

Limits on Components

Although most errors are caused by software, hardware components do fail

Have you ever had a hardware failure?

Limits on Communications

Error-detecting codes

Techniques to determine if an error has occurred during the transmission of data and then alert the system

Error-correcting codes

Error-detecting codes that detect an error has occurred and try to determine the correct value

Limits on Communications

Parity bit

An extra bit that is associated with each byte, used to ensure that the number of 1 bits in a 9-bit value (byte plus parity bit) is odd (or even) across all bytes

Parity bits are used to detect that an error has occurred between the storing and retrieving of a byte or the sending and receiving of a byte

Limits on Communications

Odd parity requires that the number of 1s in a byte plus the parity bit be odd

For example

- If a byte contains the pattern 11001100, the parity bit would be 1, thus giving an odd number of 1s
- If the pattern were 11110001, the parity bit would be 0, giving an odd number of 1s

Even parity uses the same scheme, but the number of 1 bits must be even

Error-detecting codes

Parity Bit Example: [message, parity bit]

Message:

01011110

In **Even Parity** scheme:

01011110-1

In **Odd Parity** scheme:

01011110-0

When does this scheme fail?

Limits on Communications

Check digits

- A software variation of the same scheme is to sum the individual digits of a number and store the unit's digit of that sum with the number
- For example, given the number 34376, the sum of the digits is 23, so the number would be stored as 34376–3

Error-correcting codes

- If enough information about a byte or number is kept, it is possible to deduce what an incorrect bit or digit must be

Error-detecting codes

Check Digit Example: [message, check digit]

Message:

145678

Sum:

31

Message with check digit:

145678-1

Error-correcting codes

Two-dimensional parity-check (optimal rectangular code)

Goal is to transmit the four-digit message “1234”
Arrange digits in a rectangular pattern:

12
34

Parity digits are calculated by summing each column and row separately:

12 3
34 7
46

12334746 is the message transmitted

If a single error occurs during transmission then this error can not only be detected but can also be corrected as well.

Error-correcting codes

Two-dimensional parity-check (optimal rectangular code)

What we want to see is

12 3
34 7
46

Suppose we receive 92334746 we get the grid

92 3
34 7
46

And it is clear that the error is in first column and row, and can be corrected

In order to handle two errors, a 4-dimensional scheme would be required, at the cost of more parity digits.

Error-correcting codes

Two-dimensional parity-check (optimal rectangular code)

Original Message: 1234

Message with 2D parity:

12334746

Message received:

92334746

Error-correcting codes

Two-dimensional parity-check (optimal rectangular code)

Original Message: 1234

Message with 2D parity:

12334746

Message received:

92334746

9	2	3
3	4	7
4	6	

Error-correcting codes

Two-dimensional parity-check (optimal rectangular code)

Original Message: 1234

Message with 2D parity:

12334746

Message received:

92334746

9	2	3
3	4	7
4	6	

Error-correcting codes

Two-dimensional parity-check (optimal rectangular code)

Original Message: 1234

Message with 2D parity:

12334746

Message received:

92334746

9	2	3
3	4	7
4	6	

Error-correcting codes

Two-dimensional parity-check (optimal rectangular code)

Original Message: 1234

Message with 2D parity:

12334746

Message received:

92334746

9	2	3
3	4	7
4	6	

Error-correcting codes

Two-dimensional parity-check (optimal rectangular code)

Original Message: 1234

Message with 2D parity:

12334746

Message received:

92334746

9	2	3
3	4	7
4	6	

Error-correcting codes

Two-dimensional parity-check (optimal rectangular code)

Original Message: 1234

Message with 2D parity:

12334746

Message received:

92334746

Error found in top left:

9+2 =\= 3, and 9 + 3 =\= 4

9	2	3
3	4	7
4	6	

Error-correcting codes

Two-dimensional parity-check (optimal rectangular code)

Original Message: 1234

Message with 2D parity:

12334746

Message received:

92334746

9	2	3
3	4	7
4	6	

Error found in top left:

9+2 =\= 3, and 9 + 3 =\= 4

Correction:

1 + 2 == 3, and 1 + 3 == 4

top left value should be “1”

Complexity of Software

Commercial software contains errors

- The problem is complexity
- Software testing can demonstrate the presence of bugs but cannot demonstrate their absence
  - As we find problems and fix them, we raise our confidence that the software performs as it should
  - But we can never guarantee that all bugs have been removed

Software Engineering

Remember the four stages of computer problem solving?

- Write the specifications
- Develop the algorithm
- Implement the algorithm
- Maintain the program

Moving from small, well-defined tasks to large software projects, we need to add two extra layers on top of these: Software requirements and specifications

Software Engineering

Software requirements

A statement of what is to be provided by a computer system or software product

Software specifications

A detailed description of the function, inputs, processing, outputs, and special features of a software product; it provides the information needed to design and implement the software

Software Engineering

Testing techniques have been a running thread throughout this book

They are mentioned here again as part of software engineering

Can you define walk-throughs and inspections?

Software Engineering

Use of SE techniques can reduce errors, but they will occur

A guideline for the number of errors per lines of code that can be expected

- Standard software: 25 bugs per 1,000 lines of program
- Good software: 2 errors per 1,000 lines
- Space Shuttle software: < 1 error per 10,000 lines

Formal Verification

- The verification of program correctness, independent of data testing, is an important area of theoretical computer science research
- Formal methods have been used successfully in verifying the correctness of computer chips
- It is hoped that success with formal verification techniques at the hardware level can lead eventually to success at the software level

Open Source Movement

- Pre-1970, source code came bundled with the computer.
  - Programmers modified it and shared their improvements.
  - Then companies started withholding source code
- With the growth of the Internet, programmers have once again begun to collaborate, often with a “benevolent dictator” and peer review
- The Linux operating system is the best known example of an open source project
- Now many companies consider open source development to be one of several design choices
- Some studies show that open source software has lower defect rates than commercial software
- One open source defect drew a lot of attention in 2014: the Heartbleed vulnerability in OpenSSL

Notorious Software Errors

AT&T Down for Nine Hours

In January of 1990, AT&T's long-distance telephone network came to a screeching halt for nine hours, because of a software error in an upgrade to the electronic switching systems

Notorious Software Errors

Mariner 1 Venus Probe

This probe, launched in July of 1962, veered off course almost immediately and had to be destroyed

The problem was traced to the following line of Fortran code:

```
DO 5 K = 1. 3
```

The period should have been a comma.

An $18.5 million space exploration vehicle was lost because of this typographical error

Notorious Software Errors

USS Yorktown

- Military "Smartship" left prone for 3 hours after a divide-by-zero error

Eve Online

- Game update deletes the *boot.ini* file on the C drive of thousands of users

Steam deleting all of the Linux

- Intended to recursively remove all of the steam directory
- **rm –rf "$STEAMROOT/"*** could be evaluated as **rm –rf /***
- "error handling" was left to a single comment of "# Scary!"

Numerous Overflow errors in games: Pac-Man

- *Pac-Man*, level 256, half of the screen is unplayable
- Duck Hunt, level 100, ducks are invincible

Comparing Algorithms

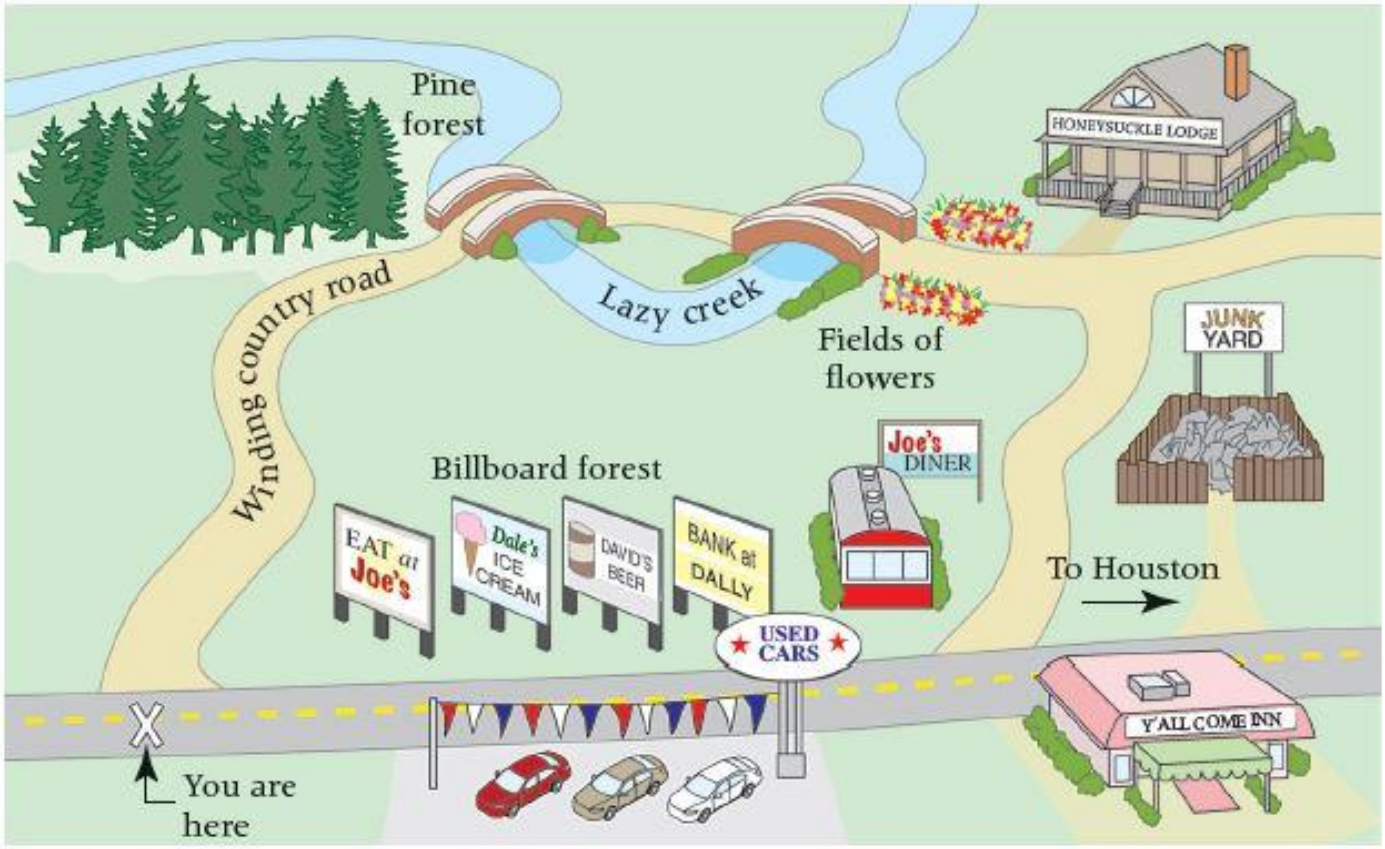


FIGURE 18.2 Equally valid solutions to the same problem

Big-O Analysis

We use a special notation to compare algorithms

Big-O notation

A notation that expresses computing time (complexity) as the term in a function that increases most rapidly relative to the size of a problem

Big-O Analysis

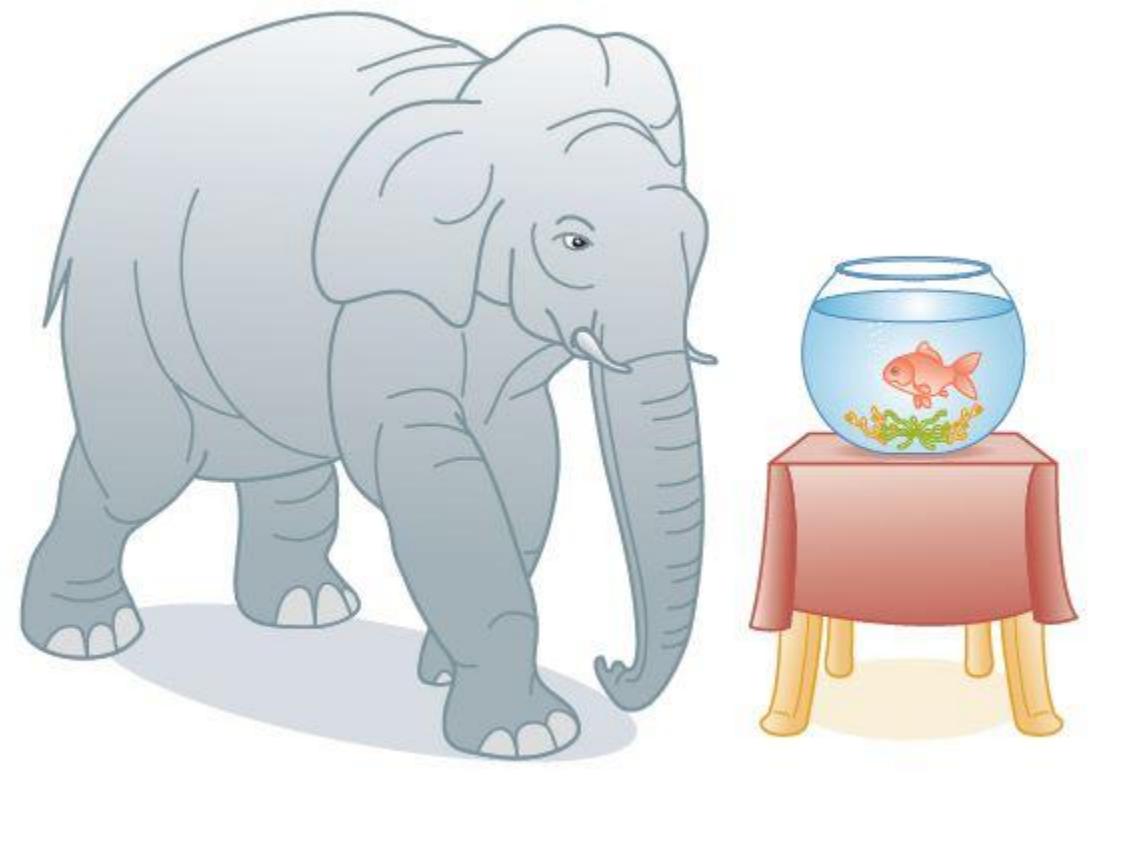
Function of size factor N:

$f(N) = N^4 + 100N^2 + 10N + 50$

Then $f(N)$ is of order N^4—or, in Big-O notation, $O(N^4)$.

For large values of N, N^4 is so much larger than 50, $10N$, or even $100\ N^2$ that we can ignore these other terms

Big-O Analysis



Big-O Analysis

Common Orders of Magnitude

- *O(1) is called bounded time*

  Assigning a value to the ith element in an array of N elements

- $O(log_2 N)$ is called logarithmic time

  Algorithms that successively cut the amount of data to be processed in half at each step typically fall into this category

  Finding a value in a list of sorted elements using the binary search algorithm is $O(log_2 N)$

Big-O Analysis

- $O(N)$ is called linear time

  Printing all the elements in a list of N elements is $O(N)$

- $O(N \log_2 N)$

  Algorithms of this type typically involve applying a logarithmic algorithm N times

  The better sorting algorithms, such as Quicksort, Heapsort, and Mergesort, have $N \log_2 N$ complexity

Big-O Analysis

- $O(N^2)$ is called quadratic time

  Algorithms of this type typically involve applying a linear algorithm N times. Most simple sorting algorithms are $O(N^2)$ algorithms

- $O(2^N)$ is called exponential time
- $O(n!)$ is called factorial time

  The traveling salesperson graph algorithm is a factorial time algorithm

Big-O Analysis

TABLE 18.2 Comparison of rates of growth

Onion: © matka_Wariatka/ShutterStock, Inc.

N	$\log_2 N$	$N \log_2 N$	N^2	N^3	2^N
1	0	1	1	1	2
2	1	2	4	8	4
4	2	8	16	64	16
8	3	24	64	512	256
16	4	64	256	4096	65,536
32	5	160	1024	32,768	4,294,967,296
64	6	384	4096	262,144	About 5 years' worth of instructions on a supercomputer
128	7	896	16,384	2,097,152	About 600,000 times greater than the age of the universe in nanoseconds (for a 6-billion-year estimate)
256	8	2048	65,536	16,777,216	Don't ask!

Big-O Analysis

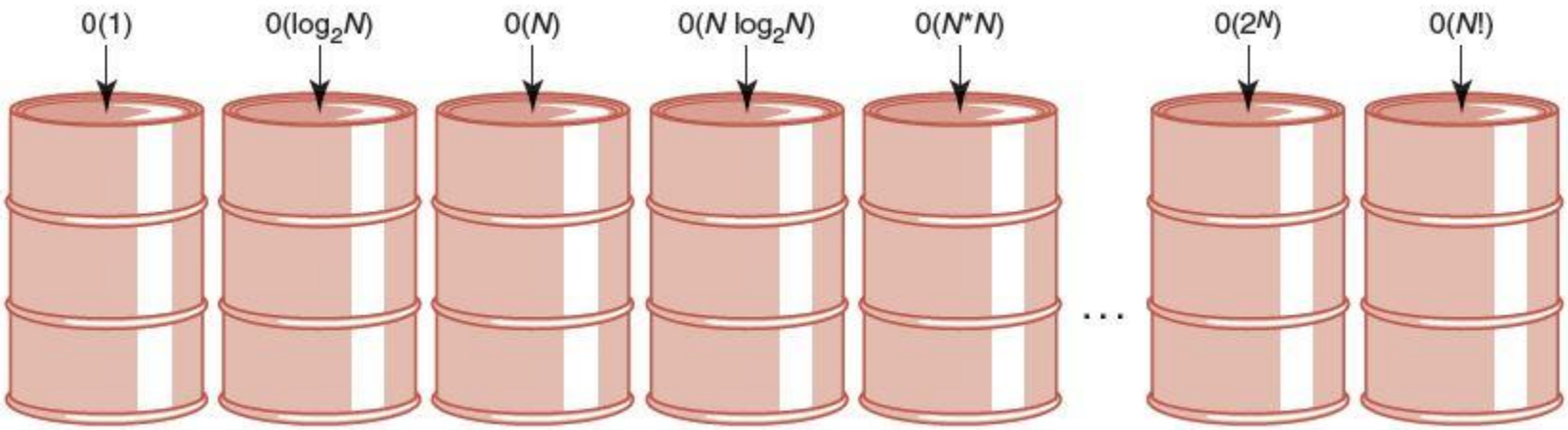


FIGURE 18.3 Orders of complexity

Turing Machines

Turing machine

A model Turing machine was developed in the 1930s and consists of a control unit with a read/write head that can read and write symbols on an infinite tape

Turing Machines

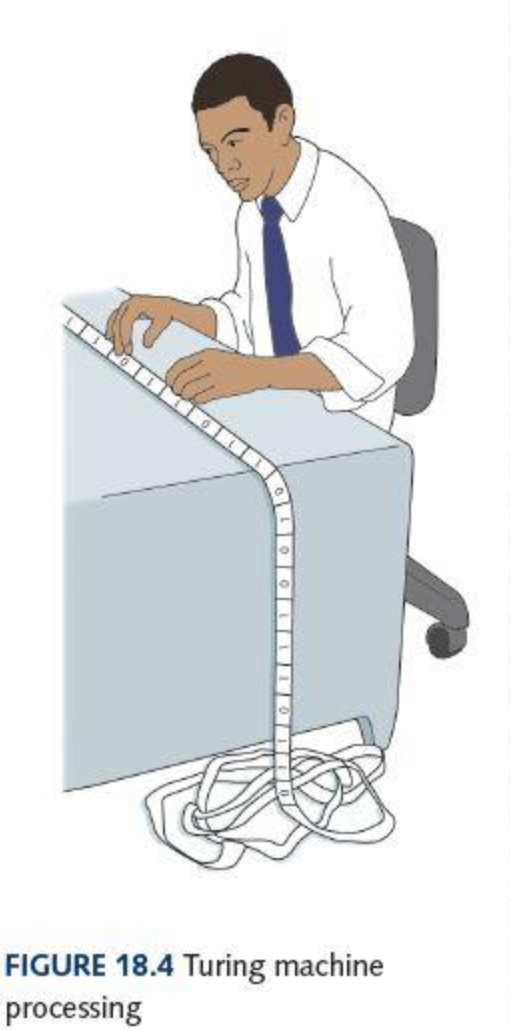


FIGURE 18.4 Turing machine processing

Why is such a simple machine (model) of any importance?

- It is widely accepted that anything that is intuitively computable can be computed by a Turing machine
- If we can find a problem for which a Turing-machine solution can be proven not to exist, then the problem must be unsolvable

Halting Problem

The Halting problem

Given a program and an input to the program, determine if the given program will eventually stop with this particular input.

If the program doesn't stop, then it is in an infinite loop and this problem is unsolvable

Halting Problem

Assume that there exists a Turing-machine program, called **SolvesHaltingProblem** that determines for any program **Example** and input **SampleData** whether program **Example** halts given input **SampleData**

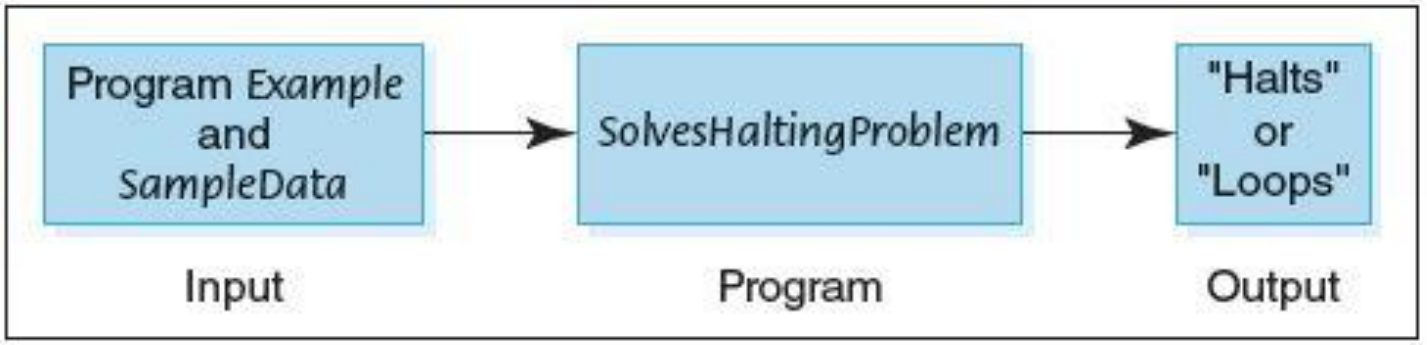


FIGURE 18.5 Proposed program for solving the halting problem

Halting Problem

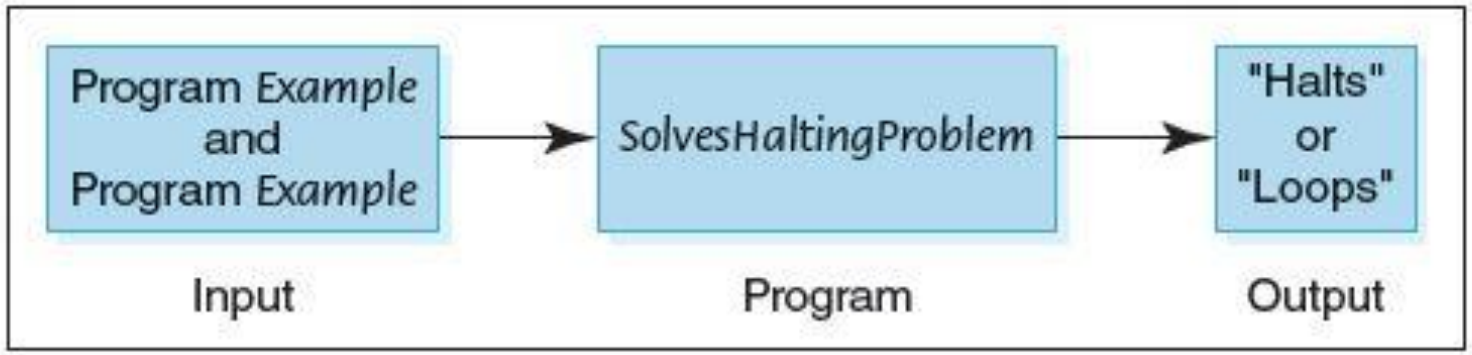


FIGURE 18.6 Proposed program for solving the halting problem

Halting Problem

Now let's construct a new program, **NewProgram,** that takes program **Example** as both program and data

And **NewProgram** uses the algorithm from **SolvesHaltingProblem** to write "Halts" if **Example** halts and "Loops" if it does not halt

Halting Problem

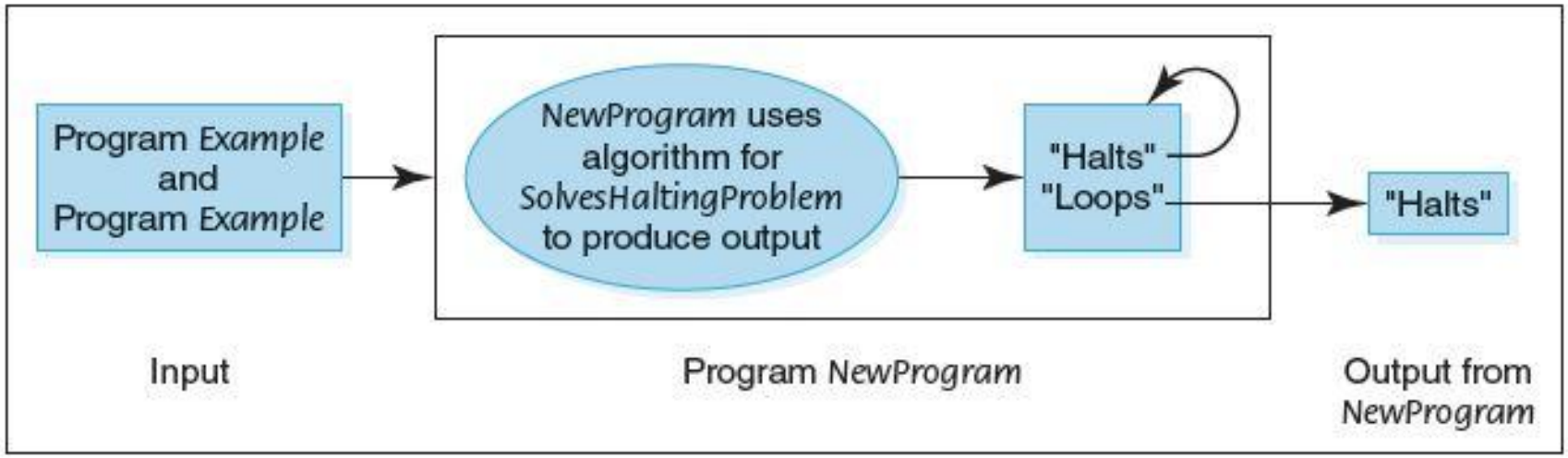


FIGURE 18.7 Construction of NewProgram

Halting Problems

Let's now apply program **SolvesHaltingProblem** to **NewProgram,** using **NewProgram** as data

- If **SolvesHaltingProblem** prints "Halts", program **NewProgram** goes into an infinite loop
- If **SolvesHaltingProblem** prints "Loops", program **NewProgram** prints "Halts" and stops
- In either case, **SolvesHaltingProblem** gives the wrong answer
- Because **SolvesHaltingProblem** gives the wrong answer in at least one case, it doesn't work on all cases

Classification of Algorithms

Polynomial-time algorithms

Algorithms whose order of magnitude can be expressed as a polynomial in the size of the problem

Class P

Problems that can be solved with one processor in polynomial time

NP-complete problems

Problems that can be solved in polynomial time with as many processors as desired

Classification of Algorithms

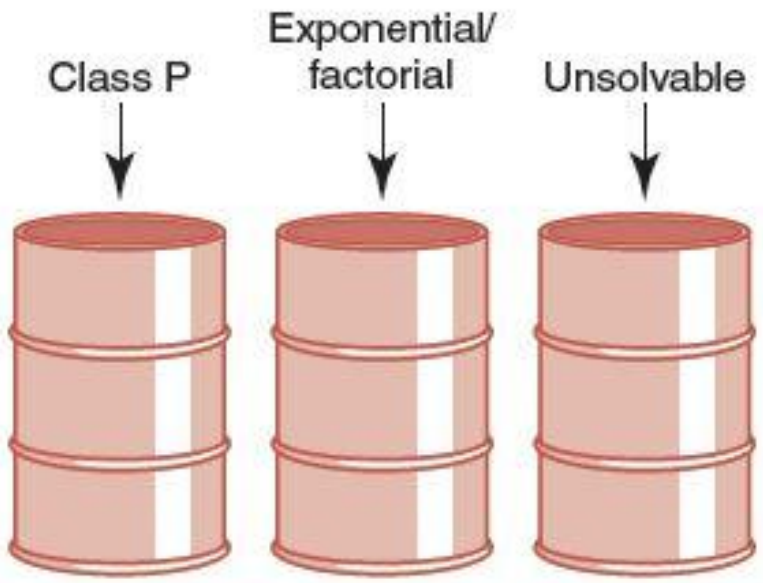


FIGURE 18.8 A reorganization of algorithm classifications

Let's reorganize our bins, combining all polynomial algorithms in a bin labeled **Class P**

Classification of Algorithms

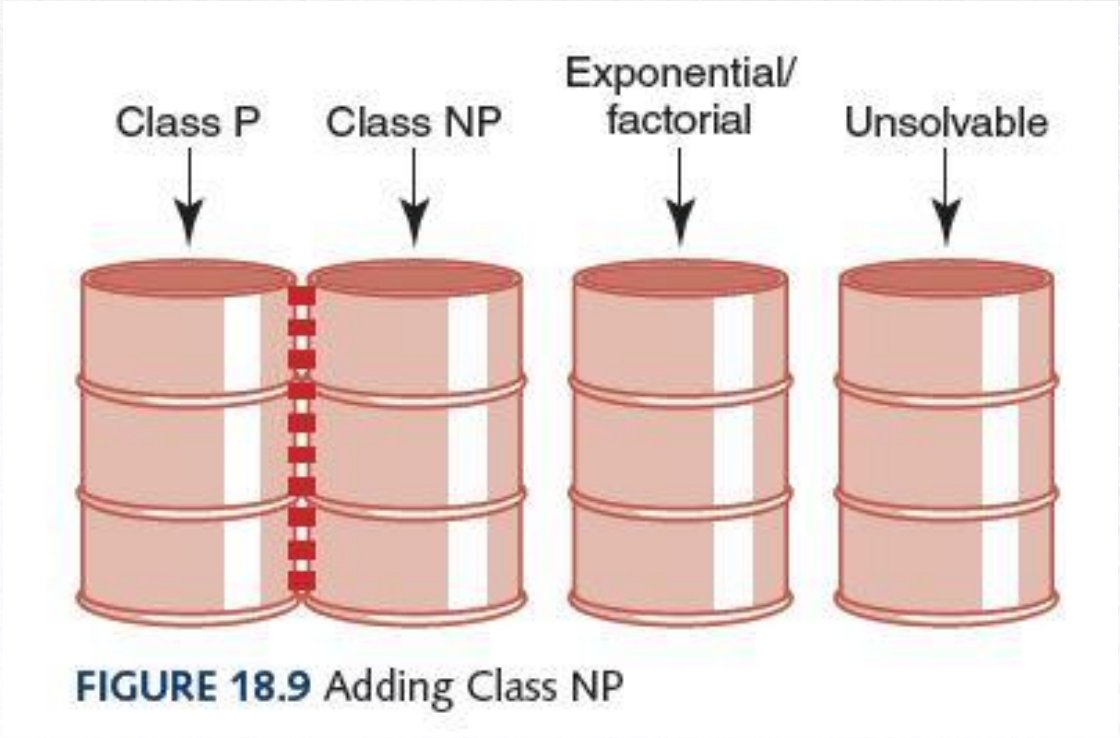


FIGURE 18.9 Adding Class NP

Are class NP problems also in class P ?

Ethical Issues

Therac-25

What were the Therac machines?

What happened to six patients?

What were the causes of the problems?

What lesson can be learned from this disaster?

Who am I?

© Pictorial Press Ltd/Alamy

What machine and what award are named after me?

Why am I on Time's list of the 100 most influential persons in the 20th century?

Why did I die before my time?

Do you know?

Why did Julia Angwin ask if privacy has become a luxury in her New York times opinion piece?

Why is it unwise to tweet about your vacation?

Watson is this, in addition to being a Jeopardy champion.

Which university had students hack into their graduate applications?

Why did Dijkstra criticize the use of the term "bugs?"

What is a micro-taggant? How is it used?

What is The Bletchley Circle?

Would you have responded to Capt. Roy's email?

What is the Traveling Salesman Problem?