Chapter 3: The Efficiency of Algorithms

Invitation to Computer Science,
C++ Version, Third Edition
Additions by Shannon Steinfadt SP'05

Objectives

In this chapter, you will learn about:

- Attributes of algorithms
- Measuring efficiency
- Analysis of algorithms
- When things get out of hand

Introduction

- Desirable characteristics in an algorithm
    - Correctness
    - Ease of understanding
    - Elegance
    - Efficiency

Attributes of Algorithms

- Correctness
    - Does the algorithm solve the problem it is designed for?
    - Does the algorithm solve the problem correctly?
        - *First, make it correct!*
- Ease of understanding
    - How easy is it to understand or alter an algorithm?
    - Important for program maintenance

Attributes of Algorithms (continued)

- Elegance
    - How clever or sophisticated is an algorithm?
    - Sometimes elegance and ease of understanding work at cross-purposes
- Efficiency
    - How much time and/or space does an algorithm require when executed?
    - Perhaps the most important desirable attribute

Measuring Efficiency

- Analysis of algorithms
    - Study of the efficiency of various algorithms
- Efficiency measured as function relating size of input to time or space used
- For one input size, best case, worst case, and average case behavior must be considered
- The Θ notation captures the order of magnitude of the efficiency function

Sequential Search

- Search for *NAME* among a list of n names
- Start at the beginning and compare *NAME* to each entry until a match is found

```
1. Get values for $NAME, n, N_1, ..., N_n$ and $T_1, ..., T_n$
2. Set the value of $i$ to 1 and set the value of Found to NO
3. While (Found = NO) and ($i \le n$) do steps 4 through 7
4.    If NAME is equal to the $i$th name on the list, $N_i$, then
5.       Print the telephone number of that person, $T_i$
6.       Set the value of Found to YES
      Else (NAME is not equal to $N_i$)
7.       Add 1 to the value of $i$
8. If (Found = NO) then
9.    Print the message 'Sorry, this name is not in the directory'
10. Stop
```

Figure 3.1
Sequential Search Algorithm

Sequential Search (continued)

- Comparison of the *NAME* being searched for against a name in the list
  - Central unit of work
  - Used for efficiency analysis
- For lists with n entries:
  - Best case
    - *NAME* is the first name in the list
    - 1 comparison
    - Θ(1)

Sequential Search (continued)

- For lists with n entries:
  - Worst case
    - *NAME* is the last name in the list
    - *NAME* is not in the list
    - n comparisons
    - Θ(n)
  - Average case
    - Roughly n/2 comparisons
    - Θ(n)

Sequential Search (continued)

- Space efficiency
  - Uses essentially no more memory storage than original input requires
  - Very space-efficient

Order of Magnitude: Order n

- As n grows large, order of magnitude dominates running time, minimizing effect of coefficients and lower-order terms
- All functions that have a linear shape are considered equivalent
- Order of magnitude n
  - Written Θ(n)
  - Functions vary as a constant times n

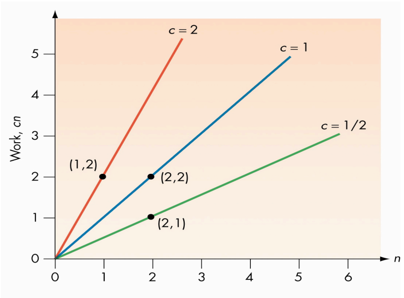
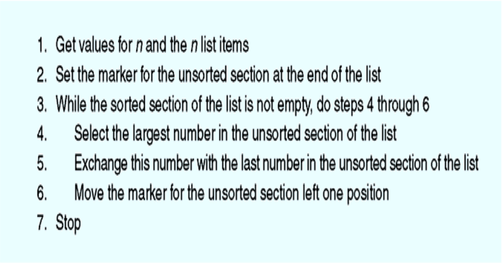


Figure 3.4
Work = cn for Various Values of c

Selection Sort

- Sorting
    - Take a sequence of n values and rearrange them into order
- Selection sort algorithm
    - Repeatedly searches for the largest value in a section of the data
        - Moves that value into its correct position in a sorted section of the list
    - Uses the Find Largest algorithm



```
1. Get values for n and the n list items
2. Set the marker for the unsorted section at the end of the list
3. While the sorted section of the list is not empty, do steps 4 through 6
4.     Select the largest number in the unsorted section of the list
5.     Exchange this number with the last number in the unsorted section of the list
6.     Move the marker for the unsorted section left one position
7. Stop
```

Figure 3.6
Selection Sort Algorithm

Selection Sort (continued)

- Count comparisons of *largest so far* against other values
- Find Largest, given m values, does m-1 comparisons
- Selection sort calls Find Largest n times,
    - Each time with a smaller list of values
    - Cost = n-1 + (n-2) + … + 2 + 1 = n(n-1)/2

Selection Sort (continued)

- Time efficiency
    - Comparisons: n(n-1)/2
    - Exchanges: n (swapping largest into place)
    - Overall: $\Theta(n^2)$, best and worst cases
- Space efficiency
    - Space for the input sequence, plus a constant number of local variables

---

Order of Magnitude – Order n^2

- All functions with highest-order term cn^2 have similar shape
- An algorithm that does cn^2 work for any constant c is order of magnitude n^2, or $\Theta(n^2)$

Order of Magnitude – Order n^2 (continued)

- Anything that is $\Theta(n^2)$ will eventually have larger values than anything that is $\Theta(n)$, no matter what the constants are
- An algorithm that runs in time $\Theta((n)$ will outperform one that runs in $\Theta(n^2)$

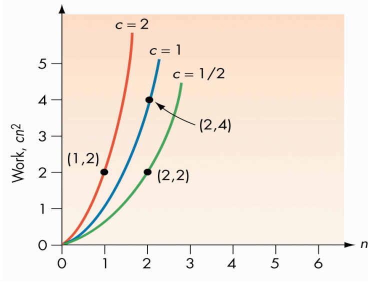


Figure 3.10
Work = cn^2 for Various Values of c

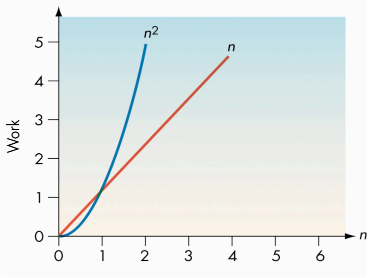


Figure 3.11
A Comparison of n and n^2

Comparision of two extreme $O(n^2)$ and $O(n)$ algorithms

	Number of Work Units Required	
	Algorithm A	Algorithm B
n	$0.0001n^2$	100n
1,000	100	100,000
10,000	10,000	1,000,000
100,000	1,000,000	10,000,000
1,000,000	100,000,000	100,000,000
10,000,000	10,000,000,000	1,000,000,000

Figure 3.13

Analysis of Algorithms

- Multiple algorithms for one task may be compared for efficiency and other desirable attributes
- Data cleanup problem
- Search problem
- Pattern matching

Data Cleanup Algorithms

- Given a collection of numbers, find and remove all zeros
- Possible algorithms
  - Shuffle-left
  - Copy-over
  - Converging-pointers

The Shuffle-Left Algorithm

- Scan list from left to right
  - When a zero is found, shift all values to its right one slot to the left

---

```
1.  Get values for n and the n data items
2.  Set the value of legit to n
3.  Set the value of left to 1
4.  Set the value of right to 2
5.  While left is less than or equal to legit do steps 6 through 14
6.    If the item at position left is not 0 then do steps 7 and 8
7.      Increase left by 1
8.      Increase right by 1
9.    Else (the item at position left is 0) do steps 10 through 14
10.     Reduce legit by 1
11.     While right is less than or equal to n do steps 12 and 13
12.       Copy the item at position right into position (right − 1)
13.       Increase right by 1
14.     Set the value of right to (left + 1)
15. Stop
```

Figure 3.14
The Shuffle-Left Algorithm for Data Cleanup

The Shuffle-Left Algorithm (continued)

- Time efficiency
  - Count examinations of list values and shifts
  - Best case
    - No shifts, n examinations
    - $\Theta(n)$
  - Worst case
    - Shift at each pass, n passes
    - n^2 shifts plus n examinations
    - $\Theta(n^2)$

The Shuffle-Left Algorithm (continued)

- Space efficiency
  - n slots for n values, plus a few local variables
  - $\Theta(n)$

The Copy-Over Algorithm

- Use a second list
  - Copy over each nonzero element in turn
- Time efficiency
  - Count examinations and copies
  - Best case
    - All zeros
    - n examinations and 0 copies
    - $\Theta(n)$

```
1.  Get values for n and the n data items
2.  Set the value of left to 1
3.  Set the value of newposition to 1
4.  While left is less than or equal to n do steps 5 through 9
5.    If the item at position left is not 0 then do steps 6 through 8
6.      Copy the item at position left into position newposition in new list
7.      Increase left by 1
8.      Increase newposition by 1
9.    Else (the item at position left is 0) increase left by 1
10. Stop
```

Figure 3.15
The Copy-Over Algorithm for Data Cleanup

The Copy-Over Algorithm (continued)

- Time efficiency (continued)
    - Worst case
        - No zeros
        - n examinations and n copies
        - Θ(n)
- Space efficiency
    - 2n slots for n values, plus a few extraneous variables

The Copy-Over Algorithm (continued)

- Time/space tradeoff
    - Algorithms that solve the same problem offer a tradeoff:
        - One algorithm uses more time and less memory
        - Its alternative uses less time and more memory

The Converging-Pointers Algorithm

- Swap zero values from left with values from right until pointers converge in the middle
- Time efficiency
    - Count examinations and swaps
    - Best case
        - n examinations, no swaps
        - Θ(n)

```
1.  Get values for n and the n data items
2.  Set the value of legit to n
3.  Set the value of left to 1
4.  Set the value of right to n
5.  While left is less than right do steps 6 through 10
6.      If the item at position left is not 0 then increase left by 1
7.      Else (the item at position left is 0) do steps 8 through 10
8.          Reduce legit by 1
9.          Copy the item at position right into position left
10.         Reduce right by 1
11. If the item at position left is 0, then reduce legit by 1
12. Stop
```

Figure 3.16
The Converging-Pointers Algorithm for Data Cleanup

The Converging-Pointers Algorithm (continued)

- Time efficiency (continued)
    - Worst case
        - n examinations, n swaps
        - Θ(n)
- Space efficiency
    - n slots for the values, plus a few extra variables

---

	1. SHUFFLE-LEFT		2. COPY-OVER		3. CONVERGING-POINTERS	
	Time	Space	Time	Space	Time	Space
Best case	$\Theta(n)$	n	$\Theta(n)$	n	$\Theta(n)$	n
Worst case	$\Theta(n^2)$	n	$\Theta(n)$	$2n$	$\Theta(n)$	n
Average case	$\Theta(n^2)$	n	$\Theta(n)$	$n \le x \le 2n$	$\Theta(n)$	n

Figure 3.17
Analysis of Three Data Cleanup Algorithms

Binary Search

- Given ordered data,
  - Search for *NAME* by comparing to middle element
  - If not a match, restrict search to either lower or upper half only
  - Each pass eliminates half the data

```
1.  Get values for $NAME, n, N_1, \ldots, N_n$ and $T_1, \ldots, T_n$
2.  Set the value of beginning to 1 and set the value of Found to NO
3.  Set the value of end to $n$
4.  While Found = NO and end is less than beginning do steps 5 through 10
5.    Set the value of $m$ to the middle value between beginning and end
6.    If NAME is equal to $N_m$, the name found at the midpoint between beginning
      and end, then do steps 7 and 8
7.      Print the telephone number of that person, $T_m$
8.      Set the value of Found to YES
9.    Else if NAME precedes $N_m$ alphabetically, then set $end = m - 1$
10.   Else (NAME follows $N_m$ alphabetically) set $beginning = m + 1$
11. If (Found = NO) then print the message 'I am sorry but that name is not in the
    directory'
12. Stop
```

Figure 3.18
Binary Search Algorithm (list must be sorted)

Binary Search (continued)

- Efficiency
  - Best case
    - 1 comparison
    - $\Theta(1)$
  - Worst case
    - lg n comparisons
      - lg n: The number of times n may be divided by two before reaching 1
    - $\Theta(\lg n)$

Binary Search (continued)

- Tradeoff
  - Sequential search
    - Slower, but works on unordered data
  - Binary search
    - Faster (much faster), but data must be sorted first

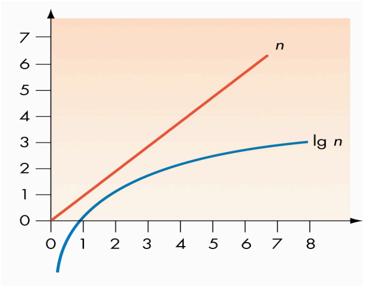


Figure 3.21
A Comparison of n and lg n

Pattern Matching

- Analysis involves two measures of input size
  - m: length of pattern string
  - n: length of text string
- Unit of work
  - Comparison of a pattern character with a text character

Pattern Matching (continued)

- Efficiency
  - Best case
    - Pattern does not match at all
    - n - m + 1 comparisons
    - Θ(n)
  - Worst case
    - Pattern almost matches at each point
    - (m -1)(n - m + 1) comparisons
    - Θ(m x n)

PROBLEM	UNIT OF WORK	ALGORITHM	BEST CASE	WORST CASE	AVERAGE CASE
Searching	Comparisons	Sequential search	1	Θ(*n*)	Θ(*n*)
		Binary search	1	Θ(lg *n*)	Θ(lg *n*)
Sorting	Comparisons and exchanges	Selection sort	Θ(n^2)	Θ(n^2)	Θ(n^2)
Data cleanup	Examinations and copies	Shuffle-left	Θ(*n*)	Θ(n^2)	Θ(n^2)
		Copy-over	Θ(*n*)	Θ(*n*)	Θ(*n*)
		Converging-pointers	Θ(*n*)	Θ(*n*)	Θ(*n*)
Pattern matching	Character comparisons	Forward march	Θ(*n*)	Θ(*m* × *n*)	

Figure 3.22
Order-of-Magnitude Time Efficiency Summary

When Things Get Out of Hand

- Polynomially bound algorithms
  - Work done is no worse than a constant multiple of n^2
- Intractable algorithms
  - Run in worse than polynomial time
  - Examples
    - Hamiltonian circuit
    - Bin-packing

When Things Get Out of Hand (continued)

- Exponential algorithm
  - Θ(2^n)
  - More work than any polynomial in n
- Approximation algorithms
  - Run in polynomial time but do not give optimal solutions

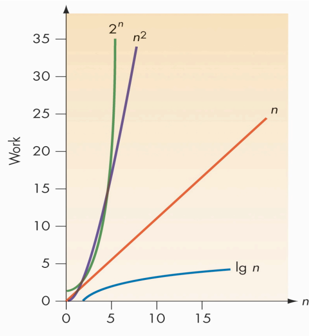


Figure 3.25
Comparisons of lg n, n, n^2 , and 2^n

ORDER	*n* 10	50	100	1,000
lg *n*	0.0003 sec	0.0006 sec	0.0007 sec	0.001 sec
n	0.001 sec	0.005 sec	0.01 sec	0.1 sec
n^2	0.01 sec	0.25 sec	1 sec	1.67 min
2^n	0.1024 sec	3,570 years	4 x 10^{16} centuries	*Too big to compute!!*

Figure 3.27
A Comparison of Four Orders of Magnitude

Summary of Level 1

- Level 1 (Chapters 2 and 3) explored algorithms
  - Chapter 2
    - Pseudocode
    - Sequential, conditional, and iterative operations
    - Algorithmic solutions to three practical problems
  - Chapter 3
    - Desirable properties for algorithms
    - Time and space efficiencies of a number of algorithms

Summary

- Desirable attributes in algorithms:
  - Correctness
  - Ease of understanding
  - Elegance
  - Efficiency
- Efficiency – an algorithm’s careful use of resources – is extremely important

Summary

- To compare the efficiency of two algorithms that do the same task
  - Consider the number of steps each algorithm requires
- Efficiency focuses on order of magnitude