Chapter 3: The Efficiency of Algorithms

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 (clarity)
  - Elegance
  - Efficiency

Attributes of Algorithms

- Correctness
  - Does the algorithm solve the problem it is designed for?
  - Does the algorithm solve the problem correctly?
- Ease of understanding (clarity)
  - How easy is it to understand or alter the algorithm?
  - Important for program maintenance
Attributes of Algorithms (continued)

- **Elegance**
  - How clever or sophisticated is the algorithm?
  - Sometimes elegance and ease of understanding work at cross-purposes

- **Efficiency**
  - How much time and/or space does the 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 a 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 \( \Theta \) 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

Figure 3.1
Sequential Search Algorithm

1. Get values for NAME, n, N₁, ..., Nₙ, T₁, ..., Tₙ
2. Set the value of i to 1 and set the value of Found to NO
3. While (Found = NO) and (i ≤ n) do steps 4 through 7
4. If NAME is equal to the ith name on the list, Nᵢ, then
5. Print the telephone number of that person, Tᵢ
6. Set the value of Found to YES
   Else (NAME is not equal to Nᵢ)
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
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
    - $\Theta(1)$
  - Average case
    - Roughly $n/2$ comparisons
    - $\Theta(n)$

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
    - $\Theta(n)$
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 $\Theta(n)$ (for average case)
  - Functions vary as a constant times $n$
Figure 3.25
Comparisons of $\lg n$, $n$, $n^2$, and $2^n$

As $n$ grows large, order of magnitude dominates running time, minimizing effect of coefficients and lower-order terms.
Analysis of Algorithms

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

Figure 3.27
A Comparison of Four Orders of Magnitude
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 \( \text{left} \) to \( n \)
3. Set the value of \( \text{left} \) to 1
4. Set the value of \( \text{right} \) to 2
5. While \( \text{left} \) is less than or equal to \( \text{left} \) do steps 6 through 14
6. If the item at position \( \text{left} \) is not 0 then do steps 7 and 8
7. Increase \( \text{left} \) by 1
8. Increase \( \text{right} \) by 1
9. Else (the item at position \( \text{left} \) is 0) do steps 10 through 14
10. Reduce \( \text{left} \) by 1
11. While \( \text{right} \) is less than or equal to \( \text{n} \) do steps 12 and 13
12. Copy the item at position \( \text{right} \) into position \( \text{right} - 1 \)
13. Increase \( \text{right} \) by 1
14. Set the value of \( \text{right} \) to \( \text{left} + 1 \)
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
    - $\Theta(n)$
  - 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
    - $\Theta(n)$
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
    - $\Theta(n)$
  - Space efficiency
    - $n$ slots for the values, plus a few extra variables
Figure 3.17
Analysis of Three Data Cleanup Algorithms

### Binary Search Algorithm

- **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, N₁, \ldots, Nₙ, and T₁, \ldots, Tₙ
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 Nₘ, the name found at the midpoint between beginning and end, then do steps 7 and 8
7. Print the telephone number of that person, Tₘ
8. Set the value of Found to YES
9. Else if NAME precedes Nₘ alphabetically, then set end = m – 1
10. Else (NAME follows Nₘ 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 Algorithm (continued)

- Efficiency
  - Best case
    - 1 comparison
    - Θ(1)
  - Worst case
    - lg n comparisons
      - lg n: The number of times n can be divided by two before reaching 1
    - Θ(lg n)
Binary Search Algorithm (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 \( \lg n \)
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

---

Figure 3.6
Selection Sort 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

---
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 = $n-1 + (n-2) + \ldots + 2 + 1 = n(n-1)/2$

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
Pattern-Matching Algorithm

- 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

(continued)

- Efficiency
  - Best case
    - Pattern does not match at all
    - $n - m + 1$ comparisons
    - $\Theta(n)$
  - Worst case
    - Pattern almost matches at each point
    - $(m - 1)(n - m + 1)$ comparisons
    - $\Theta(m \times n)$
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 $n^k$
- Intractable algorithms
  - Run in worse than polynomial time
- Examples
  - Hamiltonian circuit
  - Bin-packing
When Things Get Out of Hand (continued)

- Exponential algorithm
  - \( \Theta(2^n) \)
  - 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 \( 2^n \)
Summary of Level 1

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

Figure 3.27
A Comparison of Four Orders of Magnitude

<table>
<thead>
<tr>
<th>Order</th>
<th>$10$</th>
<th>$50$</th>
<th>$100$</th>
<th>$1000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lg n$</td>
<td>0.0003 sec</td>
<td>0.0005 sec</td>
<td>0.0007 sec</td>
<td>0.001 sec</td>
</tr>
<tr>
<td>$n$</td>
<td>0.001 sec</td>
<td>0.005 sec</td>
<td>0.01 sec</td>
<td>0.1 sec</td>
</tr>
<tr>
<td>$n^2$</td>
<td>0.01 sec</td>
<td>0.25 sec</td>
<td>1 sec</td>
<td>1.67 min</td>
</tr>
<tr>
<td>$2^n$</td>
<td>0.1024 sec</td>
<td>3,570 years</td>
<td>$4 \times 10^{16}$ centuries</td>
<td>Too big to compute!!</td>
</tr>
</tbody>
</table>
Summary

- Desirable attributes in algorithms
  - Correctness
  - Ease of understanding (clarity)
  - Elegance
  - Efficiency
- Efficiency—an algorithm’s careful use of resources—is extremely important

Summary (continued)

- 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