Chapter 3: The Efficiency of Algorithms

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?
- 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 $\Theta$ 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

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
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

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

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 $\Theta(n^2)$

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 $n^2$ (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 $\Theta(n^2)$.

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.
The Shuffle-Left Algorithm

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

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 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
  - \( \theta(n) \)

Space efficiency
- \( 2n \) slots for \( n \) values, plus a few extraneous variables

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
    - \( \Theta(n) \)

Swap zero values from left with values from right until pointers converge in the middle

Time efficiency (continued)

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

Figure 3.16
The Converging-Pointers Algorithm for Data Cleanup

Figure 3.17
Analysis of Three Data Cleanup Algorithms

The Converging-Pointers Algorithm (continued)

- Space efficiency
  - \( n \) slots for the values, plus a few extra variables

Time efficiency (continued)

- Worst case
  - \( n \) examinations, \( n \) swaps
  - \( \Theta(n) \)

Space efficiency

- \( n \) slots for the values, plus a few extra variables

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
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
    - $\Theta(\lg n)$

Pattern Matching (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
    - $\Theta(m \times n)$
When Things Get Out of Hand (continued)

- **Exponential algorithm**
  - $O(2^n)$
  - 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 three 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 of Magnitude</th>
<th>Time Efficiency Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorting</td>
<td>Compartions</td>
</tr>
<tr>
<td></td>
<td>Selection sort</td>
</tr>
<tr>
<td></td>
<td>Shuffle-iff</td>
</tr>
<tr>
<td></td>
<td>Converging-pointers</td>
</tr>
<tr>
<td>Pattern matching</td>
<td>Character comparisons</td>
</tr>
<tr>
<td></td>
<td>Forward match</td>
</tr>
</tbody>
</table>

Table 3.22 Order-of-Magnitude Time Efficiency Summary

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Time of Work</th>
<th>Access Cost</th>
<th>Best Case</th>
<th>Worst Case</th>
<th>Amount Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Searching</td>
<td>Compartions</td>
<td>Sequential</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Sorting</td>
<td>Compartions</td>
<td>Selection</td>
<td>$O(n)$</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td></td>
<td>and exchanges</td>
<td>sort</td>
<td>$O(n)$</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Data cleanup</td>
<td>Examinations</td>
<td>Shuffle-iff</td>
<td>$O(n)$</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td></td>
<td>copies</td>
<td>Converging</td>
<td>$O(n)$</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>point</td>
<td>$O(n)$</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Pattern</td>
<td></td>
<td>character</td>
<td>$O(n)$</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td></td>
<td>matching</td>
<td>comparison</td>
<td>$O(n)$</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>forward</td>
<td>$O(n)$</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>


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

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