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
  - 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 $\Theta$ 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
    - \( \Theta(1) \)

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

---

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

<table>
<thead>
<tr>
<th>$n$</th>
<th>Algorithm A</th>
<th>Algorithm B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>100</td>
<td>100,000</td>
</tr>
<tr>
<td>10,000</td>
<td>10,000</td>
<td>1,000,000</td>
</tr>
<tr>
<td>100,000</td>
<td>1,000,000</td>
<td>10,000,000</td>
</tr>
<tr>
<td>1,000,000</td>
<td>10,000,000</td>
<td>100,000,000</td>
</tr>
<tr>
<td>10,000,000</td>
<td>10,000,000</td>
<td>1,000,000,000</td>
</tr>
</tbody>
</table>

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

Figure 3.11: A Comparison of n and $n^2$

---

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

**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
    - \( \Theta(n^2) \)

**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 right to 2
4. While left is less than or equal to right 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 right in new list
7. Increase left by 1
8. Increase right by 1
9. Else the item at position left is 0 increase left by 1
10. Stop
```

Figure 3.14
The Shuffle-Left Algorithm for Data Cleanup

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

- 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
    - \( \Theta(n) \)

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

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 3.16
The Converging-Pointers Algorithm for Data Cleanup

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

Figure 3.17
Analysis of Three Data Cleanup Algorithms

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Best case</td>
<td>( \Theta(n^2) )</td>
<td>( \Theta(n) )</td>
<td>( \Theta(n) )</td>
</tr>
<tr>
<td>Worst case</td>
<td>( \Theta(n^2) )</td>
<td>( \Theta(n) )</td>
<td>( 2n )</td>
</tr>
<tr>
<td>Average case</td>
<td>( \Theta(n^2) )</td>
<td>( \Theta(n) )</td>
<td>( n/2 &lt; x &lt; 2n ) ( \Theta(n) )</td>
</tr>
</tbody>
</table>
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
    - \( \log n \) comparisons
      - \( \log n \): The number of times \( n \) may be divided by two before reaching 1
      - \( \Theta(\log n) \)

Binary Search (continued)

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

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
    - \( \Theta(n) \)
  - Worst case
    - Pattern almost matches at each point
    - \( (m-1)(n-m+1) \) comparisons
    - \( \Theta(m \times n) \)

When Things Get Out of Hand

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

<table>
<thead>
<tr>
<th>Problem</th>
<th>Best Case</th>
<th>Worst Case</th>
<th>Average Case</th>
<th>Best Case</th>
<th>Worst Case</th>
<th>Average Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Searching</td>
<td>( \Theta(n \log n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n \log n) )</td>
<td>( \Theta(n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n) )</td>
</tr>
<tr>
<td>Sorting</td>
<td>( \Theta(n \log n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n \log n) )</td>
<td>( \Theta(n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n) )</td>
</tr>
<tr>
<td>Data</td>
<td>( \Theta(n \log n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n \log n) )</td>
<td>( \Theta(n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n) )</td>
</tr>
<tr>
<td>cleanup</td>
<td>( \Theta(n \log n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n \log n) )</td>
<td>( \Theta(n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n) )</td>
</tr>
<tr>
<td>Pattern</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem</th>
<th>Best Case</th>
<th>Worst Case</th>
<th>Average Case</th>
<th>Best Case</th>
<th>Worst Case</th>
<th>Average Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Searching</td>
<td>( \Theta(n \log n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n \log n) )</td>
<td>( \Theta(n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n) )</td>
</tr>
<tr>
<td>Sorting</td>
<td>( \Theta(n \log n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n \log n) )</td>
<td>( \Theta(n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n) )</td>
</tr>
<tr>
<td>Data</td>
<td>( \Theta(n \log n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n \log n) )</td>
<td>( \Theta(n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n) )</td>
</tr>
<tr>
<td>cleanup</td>
<td>( \Theta(n \log n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n \log n) )</td>
<td>( \Theta(n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n) )</td>
</tr>
<tr>
<td>Pattern</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n) )</td>
<td>( \Theta(n \log^2 n) )</td>
<td>( \Theta(n) )</td>
</tr>
</tbody>
</table>
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