Chapter 10

Algorithm Efficiency and Sorting

Measuring the Efficiency of Algorithms

- Analysis of algorithms
  - contrasts the efficiency of different methods of solution
- A comparison of algorithms
  - Should focus of significant differences in efficiency
  - Should not consider reductions in computing costs due to clever coding tricks

The Execution Time of Algorithms

- Counting an algorithm's operations is a way to access its efficiency
  - An algorithm’s execution time is related to the number of operations it requires

Algorithm Growth Rates

- An algorithm’s time requirements can be measured as a function of the input size
- Algorithm efficiency is typically a concern for large data sets only

Algorithm Growth Rates

- Definition of the order of an algorithm
  - Algorithm A is order f(n) – denoted O(f(n)) – if constants k and n₀ exist such that A requires no more than k * f(n) time units to solve a problem of size n ≥ n₀
- Big O notation
  - A notation that uses the capital letter O to specify an algorithm’s order
  - Example: O(f(n))

Order-of-Magnitude Analysis and Big O Notation

Figure 10-1

Time requirements as a function of the problem size n

Algorithm A requires n²/5 seconds
Algorithm B requires 5*n seconds
Order-of-Magnitude Analysis and Big O Notation

A comparison of growth-rate functions: b) in graphical form.

Order of growth of some common functions:
O(1) < O(log₂n) < O(n) < O(nlog₂n) < O(n²) < O(n³) < O(2ⁿ)

Properties of growth-rate functions:
- You can ignore low-order terms
- You can ignore a multiplicative constant in the high-order term
- O(f(n)) + O(g(n)) = O(f(n) + g(n))

Worst-case analyses:
- An algorithm can require different times to solve different problems of the same size

Worst-case analysis:
- A determination of the maximum amount of time that an algorithm requires to solve problems of size n

The Efficiency of Searching Algorithms

Sequential search:
- Strategy
  - Look at each item in the data collection in turn, beginning with the first one
  - Stop when
    - You find the desired item
    - You reach the end of the data collection

Worst case: O(n)
Average case: O(n)
Best case: O(1)

Binary search:
- Strategy
  - To search a sorted array for a particular item
    - Repeatedly divide the array in half
    - Determine which half the item must be in, if it is indeed present, and discard the other half
- Efficiency
  - Worst case: O(log₂n)
Sorting Algorithms and Their Efficiency

- **Sorting**
  - A process that organizes a collection of data into either ascending or descending order

- **Categories of sorting algorithms**
  - An internal sort
    - Requires that the collection of data fit entirely in the computer’s main memory
  - An external sort
    - The collection of data will not fit in the computer’s main memory all at once but must reside in secondary storage

Data items to be sorted can be
- Integers
- Character strings
- Objects

- **Sort key**
  - The part of a record that determines the sorted order of the entire record within a collection of records

Selection Sort

- **Selection sort**
  - **Strategy**
    - Select the largest item and put it in its correct place
    - Select the next largest item and put it in its correct place, etc.

Comparison and exchange of adjacent elements

- Comparing the first two elements, the second and third elements, and so on, will move the largest (or smallest) elements to the end of the array
- Repeating this process will eventually sort the array into ascending (or descending) order

Bubble Sort

- **Bubble sort**
  - **Strategy**
    - Compare adjacent elements and exchange them if they are out of order

Analysis
- Selection sort is $O(n^2)$

Advantage of selection sort
- It does not depend on the initial arrangement of the data

Disadvantage of selection sort
- It is only appropriate for small $n$
Bubble Sort

- Analysis
  - Worst case: $O(n^2)$
  - Best case: $O(n)$

Insertion Sort

- Insertion sort
  - Strategy
    - Partition the array into two regions: sorted and unsorted
    - Take each item from the unsorted region and insert it into its correct order in the sorted region

Insertion Sort

- Analysis
  - Worst case: $O(n^2)$
  - For small arrays
    - Insertion sort is appropriate due to its simplicity
  - For large arrays
    - Insertion sort is prohibitively inefficient

Mergesort

- Important divide-and-conquer sorting algorithms
  - Mergesort
  - Quicksort
- Mergesort
  - A recursive sorting algorithm
  - Gives the same performance, regardless of the initial order of the array items
  - Strategy
    - Divide an array into halves
    - Sort each half
    - Merge the sorted halves into one sorted array
**Mergesort**

- **Analysis**
  - Worst case: $O(n \log_2 n)$
  - Average case: $O(n \log_2 n)$
  - Advantage
    - It is an extremely efficient algorithm with respect to time
  - Drawback
    - It requires a second array as large as the original array

**Quicksort**

- **Quickstart**
  - A divide-and-conquer algorithm
  - **Strategy**
    - Partition an array into items that are less than the pivot and those that are greater than or equal to the pivot
    - Sort the left section
    - Sort the right section

**Analysis**

- **Worst case**
  - quicksort is $O(n^2)$ when the array is already sorted and the smallest item is chosen as the pivot

**Quicksort**

- **Analysis**
  - **Average case**
    - quicksort is $O(n \log_2 n)$ when $S_1$ and $S_2$ contain the same—or nearly the same—number of items arranged at random
Quicksort

• Analysis
  – quicksort is usually extremely fast in practice
  – Even if the worst case occurs, quicksort’s performance is acceptable for moderately large arrays

Radix Sort

• Radix sort
  – Treats each data element as a character string
  – Strategy
    • Repeatedly organize the data into groups according to the i-th character in each element
  – Analysis
    – Radix sort is O(n)

Radix Sort

0123, 2154, 0222, 0004, 0283, 1560, 1061, 2150
(1560, 2150) (1061) (0222) (0123, 0283) (2154, 0004)
1560, 2150, 1061, 0222, 0123, 0283, 2154, 0004
(0004) (0222, 0123) (2150, 2154) (1560, 1061) (0283)
0004, 0222, 0123, 2150, 2154, 1560, 1061, 0203
(0004, 1061) (0123, 2150, 2154) (0222, 0283) (1560)
0004, 0123, 0222, 0283, 1061, 1560, 2150, 2154
0004, 0123, 0222, 0283, 1061, 1560, 2150, 2154

Figure 10-21
A radix sort of eight integers

A Comparison of Sorting Algorithms

<table>
<thead>
<tr>
<th>Worst case</th>
<th>Average case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection sort</td>
<td>n²</td>
</tr>
<tr>
<td>Bubble sort</td>
<td>n²</td>
</tr>
<tr>
<td>Insertion sort</td>
<td>n²</td>
</tr>
<tr>
<td>Mergesort</td>
<td>n * log n</td>
</tr>
<tr>
<td>Quicksort</td>
<td>n²</td>
</tr>
<tr>
<td>Radix sort</td>
<td>n</td>
</tr>
<tr>
<td>Tore sort</td>
<td>n²</td>
</tr>
<tr>
<td>Heapsort</td>
<td>n * log n</td>
</tr>
</tbody>
</table>

Figure 10-22
Approximate growth rates of time required for eight sorting algorithms

Summary

• Worst-case and average-case analyses
  – Worst-case analysis considers the maximum amount of work an algorithm requires on a problem of a given size
  – Average-case analysis considers the expected amount of work an algorithm requires on a problem of a given size
• Order-of-magnitude analysis can be used to choose an implementation for an abstract data type
• Selection sort, bubble sort, and insertion sort are all O(n²) algorithms
• Quicksort and mergesort are two very efficient sorting algorithms