Searching and Sorting: Analysis

Chris La Pilla
CompSci 6
Summer 2007

Algorithms

• Today, we will have a brief introduction to the analysis of algorithms
• An algorithm is a well-defined procedure for solving a problem (i.e. it produces output given some input)
• This is more abstract than a computer program, which is a sequence of coded instructions in a particular language

Efficiency

• So far, we have been primarily concerned with the correctness of our programs (i.e. do they give the correct results?)
• When we study algorithms, another major concern is efficiency
  – How long does an algorithm take to run?
  – How much memory does it use?
• Today we will explore the running times for some algorithms for sorting and searching

Sorting: Selection Sort

• Problem: sort an array of \( n \) integers
• Algorithm: Selection sort (illustrate on board; see Ch. 19.1)
• Idea:
  – Find minimum element and swap it with element in first position
  – Then look at remaining elements: find second smallest element and swap it with element in second position, etc…
• Let's analyze the running time of this algorithm

Running time

• How do we measure how long it takes for an algorithm to run?
• We could implement the algorithm and time it, but actual running time depends on many variables:
  – Particular machine on which code is being run
  – Programming language used
  – The actual implementation of the algorithm
• Instead, let's measure the running time in an abstract way that doesn't depend on a particular implementation

Running time

• In general, we'll measure the number of "operations" we need to do as a function of the "size" of the problem
• For the selection sort
  – size of the input means the number of elements in the array, \( n \)
  – number of operations means how many comparisons we must make
• How many comparisons must we make for selection sort (on board)?
Selection sort

- When the first $k$ elements are in the right place, we must find the minimum of the $n-k$ remaining elements.
- This involves $n-k-1$ comparisons (why?)
- So total number of comparisons is $\frac{n(n-1)}{2} \approx n^2$.

Scalability

- This result is useful, because it tells us how the algorithm's performance scales with the size of the input.
- In the case of selection sort, the number of comparisons is roughly quadratic ($n^2$) in the length of the array.
  - If we double the size of the input, how much longer will the algorithm take?
  - If we increase the input by a factor of 10?

Merge sort

- Merge sort is an alternative sorting algorithm.
- Relies on a merge procedure:
  - Suppose we have two arrays that are each sorted, and we want to combine them into one array.
  - How do we do this?

Merge sort

- Merging two arrays is easy
  - Look at the first element of each array.
  - Whichever is smaller is the next element added to the new array (illustrate on board).
- Suppose there are $n$ elements total (each initial array has $n/2$ elements).
- In the worst-case, how many comparisons must we make?

Merge sort

- Merge sort uses merge procedure to recursively sort an array:
  - If array has one element, return.
  - If not, split array in half, merge sort each half, and then merge together.
- See algorithm in chapter 19.4.
- Illustrate on board.

Analyzing Merge Sort

- How many comparisons are made? (on board)
Analyzing merge sort

- Number of comparisons needed to sort array of length \( n \) is \( n \log_2 n \)
- Scalability:
  - How much longer will algorithm take if we double the number of elements?
  - If we increase by a factor of 10?
- How does this compare to quadratic growth?

Running time comparison

From Horstmann:

<table>
<thead>
<tr>
<th>( n )</th>
<th>Merge Sort (ms)</th>
<th>Selection Sort (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000</td>
<td>31</td>
<td>772</td>
</tr>
<tr>
<td>20,000</td>
<td>47</td>
<td>3,051</td>
</tr>
<tr>
<td>30,000</td>
<td>62</td>
<td>6,846</td>
</tr>
<tr>
<td>40,000</td>
<td>80</td>
<td>12,188</td>
</tr>
<tr>
<td>50,000</td>
<td>97</td>
<td>19,015</td>
</tr>
<tr>
<td>60,000</td>
<td>113</td>
<td>27,359</td>
</tr>
</tbody>
</table>

Running time comparison

- Moral: the choice of algorithm is very important
- The choice of algorithm can have a far greater impact on the running time of the program than the speed of the computer

Linear Search vs. Binary Search

- Linear search looks for an element by looking at each element in array
- Binary search only works with sorted arrays
  - check element in middle
  - if too big, restrict search to first half of array
  - if too small, restrict search to second half
- What is the running time for each search algorithm?
- How do they compare?