Today’s topics

- Algorithms
- Complexity
- Upcoming
  - AI
- Reading
  - Brookshear 5.6

New machines vs. new algorithms

- New machine.
  - Costs $$$ or more.
  - Makes "everything" finish sooner.
  - Incremental quantitative improvements (Moore’s Law).
  - May not help much with some problems.

- New algorithm.
  - Costs $ or less.
  - Dramatic qualitative improvements possible! (million times faster)
  - May make the difference, allowing specific problem to be solved.
  - May not help much with some problems.

- Algorithmic Successes
  - N-body Simulation, Discrete Fourier transform, Quantum mechanical simulations, Pixar movies...

Algorithms

- What is an algorithm?

- So far we have been expressing our algorithms in Java code

- Pseudocode is a more informal notational system
  - Can’t be too pseudo. Should still be able to derive real code.
  - Worry about the problem solving and not compilation errors, file permission, or browser settings

- Coming up with solution is just the first problem

- For many problems, there may be several competing algorithms

- Computational complexity
  - Rigorous and useful framework for comparing algorithms and predicting performance

Linear Growth

- Grade school addition
  - Work is proportional to number of digits N
  - Linear growth: kN for some constant k

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

- How many reads? How many writes? How many operations?
Quadratic Growth

- Grade school multiplication
  - Work is proportional to square of number of digits \( N \)
  - Quadratic growth: \( kN^2 \) for some constant \( k \)

<table>
<thead>
<tr>
<th>7 &amp; 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>- 4 &amp; 2</td>
</tr>
<tr>
<td>1 &amp; 5</td>
</tr>
<tr>
<td>- 3 &amp; 1</td>
</tr>
<tr>
<td>3 &amp; 2</td>
</tr>
</tbody>
</table>

\[ \begin{array}{c}
7 & 8 \\
- & 4 & 2 \\
1 & 5 \\
3 & 1 \\
3 & 2 \\
\end{array} \]

\[ \begin{array}{c}
8 & 5 & 5 & 6 \\
1 & 2 & 0 \\
3 & 2 & 4 & 0 & 0 \\
2 & 5 & 6 & 8 & 0 & 0 & 0 \\
2 & 9 & 2 & 0 & 0 & 7 & 6 \\
\end{array} \]

N = 2

- How many reads? How many writes? How many operations?

Searching

- Determine the location or existence of an element in a collection of elements of the same type
- Easier to search large collections when the elements are already sorted
  - finding a phone number in the phone book
  - looking up a word in the dictionary
- What if the elements are not sorted?

Sequential search

- Given a collection of \( n \) unsorted elements, compare each element in sequence
- Worst-case: Unsuccessful search
  - search element is not in input
  - make \( n \) comparisons
  - search time is linear
- Average-case:
  - expect to search \( \frac{1}{2} \) the elements
  - make \( n/2 \) comparisons
  - search time is linear

Searching sorted input

- If the input is already sorted, we can search more efficiently than linear time
- Example: “Higher-Lower”
  - think of a number between 1 and 1000
  - have someone try to guess the number
  - if they are wrong, you tell them if the number is higher than their guess or lower
- Strategy?
- How many guesses should we expect to make?
Logarithms Revisited

- Power to which any other number \( a \) must be raised to produce \( n \)
  - \( a \) is called the base of the logarithm
- Frequently used logarithms have special symbols
  - \( \lg n = \log_2 n \)  
    logarithm base 2
  - \( \ln n = \log_e n \)  
    natural logarithm (base e)
  - \( \log n = \log_{10} n \)  
    common logarithm (base 10)
- If we assume \( n \) is a power of 2, then the number of times we can recursively divide \( n \) numbers in half is \( \lg n \)

Best Strategy

- Always pick the number in the middle of the range
- Why?
  - you eliminate half of the possibilities with each guess
- We should expect to make at most
  - \( \lg 1000 \approx 10 \) guesses
- Binary search
  - search \( n \) sorted inputs in logarithmic time

Sequential vs. binary search

- Average-case running time of sequential search is linear
- Average-case running time of binary search is logarithmic
- Number of comparisons:

<table>
<thead>
<tr>
<th>( n )</th>
<th>sequential search</th>
<th>binary search</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>256</td>
<td>128</td>
<td>8</td>
</tr>
<tr>
<td>4096</td>
<td>2048</td>
<td>12</td>
</tr>
<tr>
<td>65536</td>
<td>32768</td>
<td>16</td>
</tr>
</tbody>
</table>

Sorting

- Given \( n \) items, rearrange them so that they are in increasing order
- A key recurring problem
- Many different methods, how do we choose?
- Given a set of cards, describe how you would sort them:
  - Given a set of words, describe how you would sort them in alphabetical order?
Comparisons in insertion sort

- **Worst case**
  - element $k$ requires $(k-1)$ comparisons
  - total number of comparisons:
    
    $$0+1+2+ \ldots + (n-1) = \frac{1}{2} (n)(n-1)$$
    
    $$= \frac{1}{2} (n^2-n)$$

- **Best case**
  - elements 2 through $n$ each require one comparison
  - total number of comparisons:
    
    $$1+1+1+ \ldots + 1 = n-1$$

(n-1) times

## Running time of insertion sort

- **Best case running time is linear**
- **Worst case running time is quadratic**
- **Average case running time is also quadratic**
  - on average element $k$ requires $(k-1)/2$ comparisons
  - total number of comparisons:
    
    $$\frac{1}{2} (0+1+2+ \ldots + n-1) = \frac{1}{4} (n)(n-1)$$
    
    $$= \frac{1}{4} (n^2-n)$$

Comparisons in merging

- **Merging two sorted lists of size $m$ requires at least $m$ and at most $2m-1$ comparisons**
  - $m$ comparisons if all elements in one list are smaller than all elements in the second list
  - $2m-1$ comparisons if the smallest element alternates between lists

Comparisons at each merge

<table>
<thead>
<tr>
<th>#lists</th>
<th>#elements in each list</th>
<th>#merges</th>
<th>#comparisons per merge</th>
<th>#comparisons total</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>1</td>
<td>$n/2$</td>
<td>1</td>
<td>$n/2$</td>
</tr>
<tr>
<td>$n/2$</td>
<td>2</td>
<td>$n/4$</td>
<td>3</td>
<td>$3n/4$</td>
</tr>
<tr>
<td>$n/4$</td>
<td>4</td>
<td>$n/8$</td>
<td>7</td>
<td>$7n/8$</td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>2</td>
<td>$n/2$</td>
<td>1</td>
<td>$n-1$</td>
<td>$n-1$</td>
</tr>
</tbody>
</table>
**Comparisons in mergesort**
- Total number of comparisons is the sum of the number of comparisons made at each merge
  - at most $n$ comparisons at each merge
  - the number of times we can recursively divide $n$ numbers in half is $\log_2 n$, so there are $\log_2 n$ merges
  - there are at most $n \log_2 n$ comparisons total

**Comparison of sorting algorithms**
- Best, worst and average-case running time of mergesort is $\Theta(n \log n)$
- Compare to average case behavior of insertion sort:

<table>
<thead>
<tr>
<th>n</th>
<th>Insertion sort</th>
<th>Mergesort</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>25</td>
<td>33</td>
</tr>
<tr>
<td>100</td>
<td>2500</td>
<td>664</td>
</tr>
<tr>
<td>1000</td>
<td>250000</td>
<td>9965</td>
</tr>
<tr>
<td>10000</td>
<td>25000000</td>
<td>132877</td>
</tr>
<tr>
<td>100000</td>
<td>2500000000</td>
<td>1660960</td>
</tr>
</tbody>
</table>

**Quicksort**
- Most commonly used sorting algorithm
- One of the fastest sorts in practice
- Best and average-case running time is $O(n \log n)$
- Worst-case running time is quadratic
- Runs very fast on most computers when implemented correctly

**Algorithmic successes**
- N-body Simulation
- Discrete Fourier transform
- Quantum mechanical simulations
- Pixar movies...