Outline

• What is an Algorithm?
• What is Sorting?
• What is a Sorting Algorithm?
• Some Sorting Algorithms
• Asymptotic Complexity
• Summary
What is an Algorithm?

• A recipe for solving a problem

• Algorithms can be “implemented”
  – By different programs
  – In different programming languages

• Different algorithms for solving the same problem
  – Are more efficient if they require fewer basic operations
  – May solve different instances of the same problem more or less efficiently
What is Sorting?

- Sorting = Create a sequence of items in a Set S
  - Such that a binary relation (e.g., ≤) holds for pairs of consecutive elements

- We assume CARD1 < CARD2 iff
  - FaceValue(CARD1) < FaceValue(CARD2)
    - 2 < 3 < 4 < ... < J < Q < K < A
  - FaceValue(CARD1) = FaceValue(CARD2) AND
  - SuitValue(CARD1) < SuitValue(CARD2)
    - CLUBS < DIAMONDS < HEARTS < SPADES
What is a Sorting Algorithm?

• An algorithm for sorting a set of elements
• Different algorithms may be better (more efficient) for sorting under different circumstances
Insertion Sort

• Start with a shuffled deck of cards
  – Deck = Card1, Card2, ... Card52

• Choose a spot to store your Result
  – Result = a sequence of 0 or more cards
  – Result is initially empty

• For each CardN from Card1 to Card52
  – Follow the instructions on the next slide
Insertion Sort: For each CardN
• Traverse the Cards in Result from right to left comparing CardM in Result to CardN
• If CardN has a higher value than CardM
  – Put CardN to the right of CardM (and stop)
• If the cards run out before placing CardN
  – Insert CardN at the front of Result
• Otherwise, continue comparing cards
• In the worst case, all 52 cards will be compared to an average of 51/2 cards (the cards in the Result at the time) \((52 \times 51/2 = 1326\) comparisons)
Bubble Sort

- Do Bubble until the cards are sorted, i.e., until the steps do not result in any swapping of cards.
- Bubble: For each CardN between Card1..51
  - If CARDN > CARDN+1, swap the order of the two cards (CARDN becomes CARDN+1)
- A Bubble involves $51/2$ comparisons on average
- 51 bubbles will take you through the entire stack
- A complete sort takes $51 \times 51/2 = 1300.5$ comparisons in the worst case
Merge Sort

• If the deck has only one card
  – There is nothing to do
• Otherwise
  – Divide the deck of cards in half
  – Merge Sort each half
  – Merge each half together
    • Merge is defined on the next slide
Merge (for use by Merge Sort)

- Input consists of 2 Sorted Piles of Cards: 
  - Pile1 and Pile2

- Begin with an empty Result and do the following until one or both piles are empty
  - Compare the top cards of Pile1 and Pile2 and place the lowest card face down in Result

- If one of the two Piles is nonempty, place the nonempty pile at the end of the Result
Merge Sort: Number of Comparisons

- For any set of n things, you can divide those n things in half \( \log_2 n \) times before you end up with just 1 thing in each half (approximately)
- Each split corresponds to merges involving 51 comparisons (see Merge in the previous slide)
- There are approximately \( 51 \times \log_2 52 \approx 291 \) comparisons
Quick Sort

• If the deck has only one card, do nothing
• If the deck has two cards, put them in order
• Otherwise
  – Choose a card and set it as the **PIVOT** and create initially empty stacks **LEFT** and **RIGHT**
  – For each remaining **CARDN** in the deck
    • If **CARDN** precedes **PIVOT**, place **CARDN** in **LEFT**
    • ELSE, place **CARDN** in **RIGHT**
  – Quick Sort **LEFT** and **RIGHT**
  – Form a single stack
    • **LEFT** on top of **PIVOT** on top of **RIGHT**
Quick Sort 2

• Quick Sort involves in the ballpark of 52
  \[ \log_2 52 = 296 \] comparisons on average

• Quick Sort involves \( 52 \times 52 = 2704 \) comparisons in the worst case.

• Typically it tends to have fewer comparisons than Merge Sort (and other algorithms) and is thus assumed to be \textit{quicker}
Radix Sort

• Similar to Bucket Sort and Counting Sort

• Mark 13 positions (buckets) in front of you

• Go through all 52 cards and
  – Put the clubs in the first bucket
  – Put the diamonds in the second bucket
  – Put the hearts in the third bucket
  – Put the spades in the fourth bucket

• Combine the stacks:
  – first on top of second on top of third on top of fourth

• Next Slide
Radix Sort 2

• Go through all 52 cards and
  – Put the 2s in the first bucket
  – Put the 3s in the second bucket
  – ...
  – Put the Aces in the 13th bucket

• Combine the stacks:
  – first on top of second on top of third, ..., 12th on top of 13th

• I.e., sort the least significant digit or equivalent first
Complexity

• Algorithms are judged by
  – Time (how fast)
  – Space (how much memory)

• Asymptotic complexity
  – As data size increases, how fast does time/space use increase

• Upper Bound complexity (Big O)
  – Worse case

• Other cases:
  – Lower Bound
  – Average
Asymptotic Time Complexity

• Count number of basic operations
  – Comparisons of two cards
  – Insertions of cards

• Evaluating sorting algorithms
  – As the size of the deck of cards increases,
    • How many more comparisons and insertions are there?
Complexity is Approximate

- When you weigh an elephant, do you consider the weight of a fly on its back?
- Complexity
  - Expressed as a polynomial
  - Big O notation takes highest term, ignoring constants
  - $4X^3 + X^2 - 3$ becomes $O(X^3)$
Figuring out Complexity

• In Merge Sort (suppose n=52)
  – you split the deck $\log_2 n$ times
    • For a 52 card deck, you have to do about 6 splits before you have just 1 card in a pile
    – Then for each split of the whole deck, you must do $n-1$ comparisons (let's round off to $n$)
      • i.e. there are 51 comparisons for a 52 card deck
    – $n$ comparisons $\times \log_2 n$ splits means that ...
  – Merge Sort has an $n \log_2 n$ time complexity
    • Approximately 300 comparisons for 52 cards
<table>
<thead>
<tr>
<th>n</th>
<th>logn</th>
<th>2n</th>
<th>nlogn</th>
<th>N²</th>
<th>2ⁿ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1.6</td>
<td>6</td>
<td>4.8</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>2.3</td>
<td>10</td>
<td>11.6</td>
<td>25</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>2.6</td>
<td>12</td>
<td>15.5</td>
<td>36</td>
<td>64</td>
</tr>
</tbody>
</table>
Graphing Complexities

MUSICOMPUTATION
Sorting Algorithms
June, 2008
Asymptotic Complexity is an Estimate

<table>
<thead>
<tr>
<th>Actual Complexity</th>
<th>Asymptotic 1</th>
<th>Asymptotic 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$3^n - 7000$</td>
<td>$3^n$</td>
<td>exponential</td>
</tr>
<tr>
<td>$n^3 + n^2 - 2$</td>
<td>$n^3$</td>
<td>cubic</td>
</tr>
<tr>
<td>$5n^2 + 3n + 7$</td>
<td>$n^2$</td>
<td>quadratic</td>
</tr>
<tr>
<td>$n^2 + (n \log n)$</td>
<td>$n^2$</td>
<td>quadratic</td>
</tr>
<tr>
<td>$5n + \log n - 3$</td>
<td>$5n$</td>
<td>linear</td>
</tr>
<tr>
<td>$3 \log n - 50$</td>
<td>$3(\log n)$</td>
<td>logarithmic</td>
</tr>
</tbody>
</table>
Complexities

• Logarithmic Time: Constant * log n
• Linear Time: Constant * n
• nlogn Time
• Polynomial Time: \( n^m \) where m is a constant
  – \( n^2 \) or \( n^3 \) or \( n^4 \) ...
  – quadratic, cubic, etc.
• Exponential Time: \( m^n \) where m is a constant
  – \( 2^n \) or \( 3^n \) or \( 4^n \) ..
Sorting Algorithm Complexities

• Insertion Sort, Bubble Sort: $O(n^2)$

• Merge Sort: $O(n \log n)$

• Quick Sort: $O(n \log n)$
  – if implemented badly $O(n^2)$
    • Choosing bad pivot
  – Typically faster than Merge Sort

• Radix Sort: $O(kn)$, where $k =$ number of bins
  – linear time algorithm
  – uses lots of space
  – requires detailed knowledge of input
Useful Links

• Descriptions
  – en.wikipedia.org/wiki/Sort_algorithm
  – www.personal.kent.edu/~rmuhamma/Algorithms/algorithm.htm

• Animated Depictions and Applets
  – vision.bc.edu/~dmartin/teaching/sorting/anim-html/all.html
  – homepage.mac.com/mihailod/atic/sorting.html
  – coderaptors.com/?All_sorting_algorithms