How fast can we sort?

- Heapsort, Mergesort, and Quicksort all run in $O(N \log N)$ best case running time
- Can we do any better?
- Can we believe LaMoC, Inc, which claims to have discovered an $O(N \log(\log N))$ general purpose sorting algorithm?
  - The US patent office probably believes it, do you?

No! (if using comparisons only)

- Recall our basic assumption: we can only compare two elements at a time
  - we can only reduce the possible solution space by half each time we make a comparison
- Suppose you are given $N$ elements
  - Assume no duplicates
- How many possible orderings can you get?
  - Example: a, b, c ($N = 3$)
Permutations

- How many possible orderings can you get?
  - Example: a, b, c (N = 3)
  - (a b c), (a c b), (b a c), (b c a), (c a b), (c b a)
  - 6 orderings = 3\cdot2\cdot1 = 3!  (ie, “3 factorial”)
  - All the possible permutations of a set of 3 elements
- For N elements
  - N choices for the first position, (N-1) choices for the second position, …, (2) choices, 1 choice
  - \(N(N-1)(N-2)\ldots(2)(1) = N!\) possible orderings

Decision Tree

- A Decision Tree is a Binary Tree such that:
  - Each node = a set of orderings
    - ie, the remaining solution space
  - Each edge = 1 comparison
  - Each leaf = 1 unique ordering
  - How many leaves for N distinct elements?
    - \(N!\), ie, a leaf for each possible ordering
  - Only 1 leaf has the ordering that is the desired correctly sorted arrangement

Decision Trees and Sorting

- Every sorting algorithm corresponds to a decision tree
  - Finds correct leaf by choosing edges to follow
    - ie, by making comparisons
  - Each decision reduces the possible solution space by one half
- Run time is \(\geq\) maximum no. of comparisons
  - maximum number of comparisons is the length of the longest path in the decision tree
    - the length of the longest path is the depth of the tree
**Decision Tree Depth Example**

- Possible orders:
  - \(a < b < c\)
  - \(b < c < a\)
  - \(c < a < b\)
  - \(a < c < b\)
  - \(b < a < c\)
  - \(c < b < a\)

- Actual order:
  - \(a < b < c\)
  - \(a < c < b\)
  - \(c < a < b\)

**How many leaves on a tree?**

- Suppose you have a binary tree of depth \(d\).
- How many leaves can the tree have?
  - \(d = 1 \rightarrow\) at most 2 leaves,
  - \(d = 2 \rightarrow\) at most 4 leaves, etc.

**How deep is it, Jim?**

- A binary tree of depth \(d\) has at most \(2^d\) leaves
  - \(d = 1 \rightarrow 2\) leaves, \(d = 2 \rightarrow 4\) leaves, etc.
  - Can prove by induction
- The decision tree has \(L = N!\) leaves
- Depth \(d\) must be deep enough such that \(2^d \geq L\)
  - \(2^d \geq L \rightarrow d \geq \log L\)
- So the decision tree depth is \(d \geq \log(N!)\)

**log(N!) is \(\Omega(N \log N)\)**

- \(\log(N!) = \log(N \cdot (N - 1) \cdot (N - 2) \cdots (2) \cdot (1))\)
  - \(= \log N + \log(N - 1) + \log(N - 2) + \cdots + \log 2 + \log 1\)
  - \(\geq \log N + \log(N - 1) + \log(N - 2) + \cdots + \log \left(\frac{N}{2}\right)\)
  - \(\geq \frac{N}{2} \log \frac{N}{2}\)
  - \(\geq \frac{N}{2} \left(\log N - \log 2\right)\)
  - \(= \Omega(N \log N)\)
Ω(N log N)

- Run time of any comparison-based sorting algorithm is Ω(N log N)
  - Any sorting algorithm based on comparisons between elements requires Ω(N log N) comparisons
  - Can never find an O(N log log N) general purpose sorting algorithm
    - sorry, LaMoC, Inc!
    - get a clue, patent office

What about bucket sort?

- You may be saying to yourself
  “But on slide 27 of the List lecture on April 5th, he showed that the bucket sort only takes O(N+B) operations, what's up with that?”
- And I say to you: Advance knowledge of the data lets you do all sorts of magic
  - perfect hash
  - bucket sort, radix sort

Bucket Sort: Sorting integers

- Bucket sort: N integers in the range 0 to B-1
  - Array Count has B elements ("buckets"), initialized to 0
  - Given input integer i, Count[i]++
  - After reading all N numbers go through the B buckets and read out the resulting sorted list
  - N operations to read and record the numbers plus B operations to recover the sorted numbers

Bucket Sort Run Time?

- What is the running time for sorting N integers?
  - Running Time: O(B+N)
    - B to zero/scan the array and N to read the input
  - If B is Θ(N), running time for Bucket sort = O(N)
- Doesn’t this violate the O(N log N) lower bound result??
- No – When we do Count[i]++, we are comparing one element with all B elements, not just two elements
Radix Sort: Sorting integers

- Radix sort = multi-pass bucket sort of integers in the range 0 to $B^P - 1$
  - Bucket-sort from least significant to most significant “digit” (base $B$)
  - Use linked list to store numbers that are in same bucket
  - Requires $P*(B+N)$ operations where $P$ is the number of passes (the number of base $B$ digits in the largest possible input number)
  - Do $P$ passes instead of using $B^P$ space

Radix Sort Example

<table>
<thead>
<tr>
<th>data</th>
<th>Bucket sort by 1’s digit</th>
<th>Bucket sort by 10’s digit</th>
<th>Bucket sort by 100’s digit</th>
</tr>
</thead>
<tbody>
<tr>
<td>478</td>
<td>721</td>
<td>37</td>
<td>3</td>
</tr>
<tr>
<td>537</td>
<td>121</td>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>72</td>
<td>38</td>
<td>9</td>
</tr>
<tr>
<td>721</td>
<td>123</td>
<td>3</td>
<td>123</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>53</td>
<td>3</td>
</tr>
<tr>
<td>38</td>
<td>123</td>
<td>72</td>
<td>3</td>
</tr>
<tr>
<td>123</td>
<td>67</td>
<td>87</td>
<td>7</td>
</tr>
<tr>
<td>67</td>
<td>9</td>
<td>478</td>
<td>9</td>
</tr>
</tbody>
</table>

This example uses $B=10$ and base 10 digits for simplicity of demonstration. Larger bucket counts should be used in an actual implementation.

Internal versus External Sorting

- So far assumed that accessing $A[i]$ is fast – Array $A$ is stored in internal memory (RAM)
  - Algorithms so far are good for internal sorting
- What if $A$ is so large that it doesn’t fit in internal memory?
  - Data on disk or tape
  - Delay in accessing $A[i]$ – e.g. need to spin disk and move head

Internal versus External Sorting

- Need sorting algorithms that minimize disk/tape access time
  - External sorting – Basic Idea:
    - Load chunk of data into RAM, sort, store this “run” on disk/tape
    - Use the Merge routine from Mergesort to merge runs
    - Repeat until you have only one run (one sorted chunk)
    - Text gives some examples
- But … how important is external sorting?
Internal Memory is getting cheap…

From: http://www.macresource.com/mrp/ramwatch/trend.shtml

Price (in US$) for 1 MB of RAM

External Sorting

- For most data sets, internal sorting in a large memory space is possible and intricate external sorts are not required
  - Tapes seldom used these days – random access disks are faster and getting cheaper with greater capacity
  - Operating systems provide very, very large virtual memory address spaces so it looks like an internal sort, even though you are using the disk
    - be careful though, you can end up doing a lot of disk I/O if you’re not careful

Okay…so let’s talk about performance in practice

Input Size N

Run time (in seconds)

[Data from textbook Chap. 7]
Summary of Sorting

- Sorting choices:
  - O(N^2) – Bubblesort, Selection Sort, Insertion Sort
  - O(N^3) – Shellsort (x = 3/2, 4/3, 7/6, 2, etc. depending on increment sequence)
  - O(N log N) average case running time:
    - **Heapsort**: uses 2 comparisons to move data (between children and between child and parent) – may not be fast in practice (see graph)
    - **Mergesort**: easy to code but uses O(N) extra space
    - **Quicksort**: fastest in practice but trickier to code, O(N^2) worst case

Practical Sorting

- When N is large, use Quicksort with median3 pivot
- For small N (< 20), the N log N sorts are slower due to extra overhead (larger constants in big-Oh notation)
  - For N < 20, use Insertion sort
  - In Quicksort, do insertion sort when sub-array size < 20 (instead of partitioning)
- When you need a sorter
  - remember the various candidate algorithms
  - think about the type and quantity of your data
  - look up an appropriate reference implementation and adapt it to your requirements (ElementType, comparator, etc)