CSE 373 Lecture 19: Wrap-Up of Sorting

✦ What’s on our platter today?
  ✓ How fast can the fastest sorting algorithm be?
  ✓ Lower bound on comparison-based sorting
  ✓ Tricks to sort faster than the lower bound
  ✓ External versus Internal Sorting
  ✓ Practical comparisons of internal sorting algorithms
  ✓ Summary of sorting

✦ Covered in Chapter 7 of the textbook

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 Pat Swe (pronounced "Sway") from Swetown (formerly Softwareville), USA, who claims to have discovered an $O(N \log \log N)$ sorting algorithm?

The Answer is No! (if using comparisons only)

✦ Recall our basic assumption: we can only compare two elements at a time – how does this limit the run time?

✦ Suppose you are given $N$ elements
  ✓ Assume no duplicates – any sorting algorithm must also work for this case

✦ How many possible orderings can you get?
  ✓ Example: $a, b, c$ ($N = 3$)

R. Rao, CSE 373
A “Decision Tree”

Leaves contain possible orderings of a, b, c

Decision Trees and Sorting

✦ A Decision Tree is a Binary Tree such that:
  - Each node = a set of orderings
  - Each edge = 1 comparison
  - Each leaf = 1 unique ordering
  - How many leaves for N distinct elements?

✦ Only 1 leaf has sorted ordering

✦ Each sorting algorithm corresponds to a decision tree
  - Finds correct leaf by following edges (= comparisons)

✦ Run time \( \geq \) maximum no. of comparisons
  - Depends on: depth of decision tree
  - What is the depth of a decision tree for N distinct elements?

Lower Bound on Comparison-Based Sorting

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

Lower Bound on Comparison-Based Sorting

✦ A binary tree of depth d has at most \( 2^d \) leaves
  - E.g. depth d = 1 \( \rightarrow \) 2 leaves, d = 2 \( \rightarrow \) 4 leaves, etc.
  - Can prove by induction

✦ Number of leaves \( L \leq 2^d \rightarrow d \geq \log L \)

✦ Decision tree has \( L = N! \) leaves \( \rightarrow \) its depth \( d \geq \log(N!) \)
  - What is \( \log(N!) \)? (first, what is \( \log(4!) \)?)
Lower Bound on Comparison-Based Sorting

- Decision tree has $L = N!$ leaves $\Rightarrow$ its depth $d \geq \log(N!)$
  - What is $\log(N!)$? (first, what is $\log(A \cdot B)$?)
  - $\log(N!) = \log N + \log(N-1) + \ldots + \log(N/2) + \ldots + \log 1$
    1. $\log N + \log(N-1) + \ldots + \log(N/2)$ (N/2 terms only)
    2. $(N/2) \cdot \log(N/2) = \Omega(N \log N)$

- Result: Any sorting algorithm based on comparisons between elements requires $\Omega(N \log N)$ comparisons

Hey! (you say)…what about Bucket Sort?

- Recall: Bucket sort $\Rightarrow$ Elements are integers known to always be in the range 0 to B-1
  - Idea: Array Count has B slots ("buckets")
  1. Initialize: Count[i] = 0 for i = 0 to B-1
  2. If input integer = i, Count[i]++
  3. After reading all inputs, scan Count[i] for i = 0 to B-1 and print i if Count[i] ≠ 0

- 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 $\Theta(N)$, then 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

Hey! (you say)…what about Bucket Sort?

- Recall: Bucket sort $\Rightarrow$ Elements are integers known to always be in the range 0 to B-1
  - Idea: Array Count has B slots ("buckets")
  1. Initialize: Count[i] = 0 for i = 0 to B-1
  2. Given input integer i, Count[i]++
  3. After reading all inputs, scan Count[i] for i = 0 to B-1 and print i if Count[i] is non-zero

- What is the running time for sorting N integers?
  - If B is $\Theta(N)$, then running time for Bucket sort = $O(N)$

Radix Sort = Stable Bucket Sort

- Problem: What if number of buckets needed is too large?
- Recall: Stable sort = a sort that does not change order of items with same key
- Radix sort = stable bucket sort on “slices” of key
  - E.g. Divide into integers/strings in digits/characters
  - Stability ensures keys already sorted stay sorted
  - Takes $O(P(B+N))$ time where $P = \text{number of digits}$
Radix Sort Example

<table>
<thead>
<tr>
<th>478</th>
<th>721</th>
<th>03</th>
<th>003</th>
</tr>
</thead>
<tbody>
<tr>
<td>537</td>
<td>532</td>
<td>123</td>
<td>038</td>
</tr>
<tr>
<td>9</td>
<td>62</td>
<td>9721</td>
<td>009</td>
</tr>
<tr>
<td>721</td>
<td>537</td>
<td>123</td>
<td>067</td>
</tr>
<tr>
<td>3</td>
<td>67</td>
<td>57</td>
<td>123</td>
</tr>
<tr>
<td>38</td>
<td>478</td>
<td>38</td>
<td>478</td>
</tr>
<tr>
<td>123</td>
<td>38</td>
<td>67</td>
<td>537</td>
</tr>
<tr>
<td>67</td>
<td>2</td>
<td>428</td>
<td>221</td>
</tr>
</tbody>
</table>

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
  ➤ 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

✦ Wait a minute!! How important is external sorting?

Internal Memory is getting dirt cheap...

![Price of RAM](http://www.macresource.com/mrp/ramwatch/trend.shtml)

External Sorting: A (soon-to-be) Relic of the Past?

✦ Price of internal memory is dropping, memory size is increasing, both at exponential rates (Moore’s law)

✦ Quite likely that in the future, data will probably fit in internal memory for reasonably large input sizes

✦ Tapes seldom used these days – disks are faster and getting cheaper with greater capacity

✦ So, need not worry too much about external sorting

✦ For all practical purposes, internal sorting algorithms such as Quicksort should prove to be efficient
Okay...so let's talk about performance in practice

<table>
<thead>
<tr>
<th>Input Size N</th>
<th>Insertion sort</th>
<th>Heapsort</th>
<th>Shellsort</th>
<th>Quicksort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run time (in seconds)</td>
<td>[Data from textbook Chap. 7]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Summary of Sorting

✦ Sorting choices:
  - \(O(N^2)\) – Bubblesort, Selection Sort, Insertion Sort
  - \(O(N \times x)\) – Shellsort \((x = 3/2, 4/3, 7/6, 2, \text{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 advice: When \(N\) is large, use Quicksort with median-of-three pivot. For small \(N\) (> 20), the \(N \log N\) sorts are slower due to extra overhead (larger constants in \(\text{big-oh notation}\))
  - For \(N < 20\), use Insertion sort
  - E.g. In Quicksort, do insertion sort when sub-array size < 20 (instead of partitioning) and return this sorted sub-array for further processing

Next time: Union-Find and Disjoint Sets

To do:
- Finish reading chapter 7
- Start reading chapter 8
- Have a great weekend!