Announcements

- Homework 6 due before last class: May 4th
- Final Review May 4th
- Exam Wednesday May 13th 1:10-4:00 PM, 633
  - cumulative, closed-book/notes
Review

- \(O(M \log^* N)\) running time for \(M\) unions/finds
- Counted cost of each find by two kinds of “pennies”: American/Canadian
- Basic intuition: Canadian when node is in middle of rank group, American when node is between groups
- Comparison Sort lower bound vs. Radix Sort
Today’s Plan

- Radix Sort specifics
- Comparison sorting algorithm characteristics
- Algorithms: Selection Sort, Insertion Sort, Shellsort, Heapsort, Mergesort, Quicksort
Radix Sort with Least Significant Digit

- CountingSort according to the least significant digit
- Repeat: CountingSort according to the next least significant digit
- Each step must be stable
- Running time: $O(Nk)$ for maximum of $k$ digits
- Space: $O(N+b)$ for base-$b$ number system*
Radix Sort Example

<table>
<thead>
<tr>
<th>815</th>
<th>906</th>
<th>127</th>
<th>913</th>
<th>98</th>
<th>632</th>
<th>278</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th></th>
<th></th>
<th></th>
<th>1</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Radix Sort Example

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>815</td>
<td></td>
</tr>
<tr>
<td>906</td>
<td></td>
</tr>
<tr>
<td>127</td>
<td></td>
</tr>
<tr>
<td>913</td>
<td></td>
</tr>
<tr>
<td>98</td>
<td></td>
</tr>
<tr>
<td>632</td>
<td></td>
</tr>
<tr>
<td>278</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>632</td>
</tr>
<tr>
<td>3</td>
<td>913</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>815</td>
</tr>
<tr>
<td>6</td>
<td>906</td>
</tr>
<tr>
<td>7</td>
<td>127</td>
</tr>
<tr>
<td>8</td>
<td>98, 278</td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>
Radix Sort Example

632
913
815
906
127
98
278

0
1
2
3
4
5
6
7
8
9
Radix Sort Example

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>906</td>
</tr>
<tr>
<td>1</td>
<td>913, 815</td>
</tr>
<tr>
<td>2</td>
<td>127</td>
</tr>
<tr>
<td>3</td>
<td>632</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>278</td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>98</td>
</tr>
</tbody>
</table>
Radix Sort Example

906
913
815
127
632
278
98

0
1
2
3
4
5
6
7
8
9
Radix Sort Example

<table>
<thead>
<tr>
<th>906</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>913</td>
<td></td>
</tr>
<tr>
<td>815</td>
<td></td>
</tr>
<tr>
<td>127</td>
<td></td>
</tr>
<tr>
<td>632</td>
<td></td>
</tr>
<tr>
<td>278</td>
<td></td>
</tr>
<tr>
<td>98</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>98</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>127</td>
</tr>
<tr>
<td>2</td>
<td>278</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>632</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>815</td>
</tr>
<tr>
<td>9</td>
<td>906, 913</td>
</tr>
</tbody>
</table>
Analysis

- For maximum of $k$ digits (in whatever base), we need $k$ passes through the array, $O(Nk)$

- For base-$b$ number system, we need $b$ queues, which will end up containing $N$ elements total, so $O(N+b)$ space

- Stable because if elements are the same, they keep being enqueued and dequeued in the same order
Comparison Sorts

- Of course, Radix Sort only works well for sorting keys representable as digital numbers.
- In general, we must often use comparison sorts.
- We have proven an $\Omega(N \log N)$ lower bound for running time.
- But algorithms also have other desirable characteristics.
Sorting Algorithm Characteristics

- Worst case running time
- Worst case space usage (can it run in place?)
- Stability
- Average running time/space
- (simplicity)
Selection Sort

- Swap least unsorted element with first unsorted element
- Unstable
- Running time $O(N^2)$
- In place O(1) space
- Algorithm Animation
Insertion Sort

- Assume first $p$ elements are sorted. Insert $(p+1)$'th element into appropriate location.
  - Save $A[p+1]$ in temporary variable $t$, shift sorted elements greater than $t$, and insert $t$

- Stable

- Running time $O(N^2)$

- In place $O(1)$ space
Insertion Sort Analysis

- When the sorted segment is $i$ elements, we may need up to $i$ shifts to insert the next element.

$$\sum_{i=2}^{N} i = N(N - 1)/2 - 1 = O(N^2)$$

- Stable because elements are visited in order and equal elements are inserted after its equals.

- Algorithm Animation
Shellsort

- Essentially splits the array into subarrays and runs Insertion Sort on the subarrays.
- Uses an increasing sequence, $h_1, \ldots, h_t$, such that $h_1 = 1$.
- At phase $k$, all elements $h_k$ apart are sorted; the array is called $h_k$-sorted.
- For every $i$, $A[i] \leq A[i + h_k]$. 
Shell Sort Correctness

- Efficiency of algorithm depends on that elements sorted at earlier stages remain sorted in later stages

- Unstable. Example: 2-sort the following: [5 5 1]
Increment Sequences

- Shell suggested the sequence $h_t = \lceil N/2 \rceil$ and $h_k = \lfloor h_{k+1}/2 \rfloor$, which was suboptimal.
- A better sequence is $h_k = 2^k - 1$.
- Shellsort using better sequence is proven $\Theta(N^{3/2})$.
- Often used for its simplicity and sub-quadratic time, even though $O(N \log N)$ algorithms exist.
- Animation
Heapsort

- Build a \textbf{max} heap from the array: $O(N)$
- call deleteMax $N$ times: $O(N \log N)$
- $O(1)$ space
- Simple if we abstract heaps
- Unstable
- Animation
Mergesort

- Quintessential divide-and-conquer example
- Mergesort each half of the array, merge the results
- Merge by iterating through both halves, compare the current elements, copy lesser of the two into output array
- Animation
Mergesort Recurrence

Merge operation is costs $O(N)$

$T(N) = 2\ T(N/2) + N$

We solved this recurrence for the recursive solutions to the homework 1 theory problem

$$= \sum_{i=0}^{\log N} 2^i c \frac{N}{2^i}$$

$$= \sum_{i=0}^{\log N} cN = cN \log N$$
Quicksort

- Choose an element as the **pivot**
- Partition the array into elements greater than pivot and elements less than pivot
- Quicksort each partition

**Animation**
Choosing a Pivot

- The worst case for Quicksort is when the partitions are of size zero and $N-1$.
- Ideally, the pivot is the median, so each partition is about half.
- If your input is random, you can choose the first element, but this is very bad for presorted input!
- Choosing randomly works, but a better method is...
Median-of-Three

• Choose three entries, use the median as pivot

• If we choose randomly, $\frac{2}{N}$ probability of worst case pivots

• Median-of-three gives 0 probability of worst case, tiny probability of 2nd-worst case. (Approx. $\frac{2}{N^3}$)

• Randomness less important, so choosing (first, middle, last) works reasonably well
Partitioning the Array

- Once pivot is chosen, swap pivot to end of array. Start counters \( i = 1 \) and \( j = N-1 \)
- Intuition: \( i \) will look at less-than partition, \( j \) will look at greater-than partition
- Increment \( i \) and decrement \( j \) until we find elements that don't belong (\( A[i] > \text{pivot} \) or \( A[j] < \text{pivot} \))
- Swap \( (A[i], A[j]) \), continue increment/decrements
- When \( i \) and \( j \) touch, swap pivot with \( A[j] \)
Quicksort Worst Case

- Running time recurrence includes the cost of partitioning, then the cost of 2 quicksorts
- We don't know the size of the partitions, so let $i$ be the size of the first partition
- $T(N) = T(i) + T(N-i-1) + N$
- Worst case is $T(N) = T(N-1) + N$
Quicksort Average Case

We'll average over all partition sizes:

\[ T(N) = \frac{2}{N-1} \sum_{i=0}^{N-1} T(i) + N \]

\[ NT(N) = 2 \sum_{i=0}^{N-1} T(i) + N^2 \]

\[ (N - 1)T(N - 1) = 2 \sum_{i=0}^{N-2} T(i) + (N - 1)^2 \]
Quicksort Average Case

\[ NT(N) = 2 \sum_{i=0}^{N-1} T(i) + N^2 \]

\[ (N - 1)T(N - 1) = 2 \sum_{i=0}^{N-2} T(i) + (N - 1)^2 \]

\[
NT(N) - (N - 1)T(N - 1) = 2 \left[ \sum_{i=0}^{N-1} T(i) - \sum_{i=0}^{N-2} T(i) \right] + N^2 - (N - 1)^2
\]
QuickSort Average Case

\[ NT(N) - (N - 1)T(N - 1) = 2 \left[ \sum_{i=0}^{N-1} T(i) - \sum_{i=0}^{N-2} T(i) \right] + N^2 - (N - 1)^2 \]

\[ NT(N) - (N - 1)T(N - 1) = 2T(N - 1) + 2N - 1 \]

\[ NT(N) = (N + 1)T(N - 1) + 2N \]

\[ \frac{T(N)}{N + 1} = \frac{T(N - 1)}{N} + \frac{2}{N + 1} \]
Quicksort Average Case

\[
\begin{align*}
\frac{T(N)}{N + 1} &= \frac{T(N - 1)}{N} + \frac{2}{N + 1} \\
\frac{T(N - 2)}{N - 1} &= \frac{T(N - 3)}{N - 2} + \frac{2}{N - 1} \\
\frac{T(2)}{3} &= \frac{T(1)}{2} + \frac{2}{3} \\
\frac{T(N)}{N + 1} &= \frac{T(1)}{2} + 2 \sum_{i=3}^{N+1} \frac{1}{i} \\
\frac{T(N)}{N + 1} &= O(\log N) \quad \text{and} \quad T(N) = O(N \log N)
\end{align*}
\]
Quicksort Properties

- Unstable
- Average time $O(N \log N)$
- Worst case time $O(N^2)$
- Space $O(\log N)/O(N^2)$ because we need to store the pivots
### Summary

<table>
<thead>
<tr>
<th></th>
<th>Worst Case Time</th>
<th>Average Time</th>
<th>Space</th>
<th>Stable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>$O(N^2)$</td>
<td>$O(N^2)$</td>
<td>$O(1)$</td>
<td>No</td>
</tr>
<tr>
<td>Insertion</td>
<td>$O(N^2)$</td>
<td>$O(N^2)$</td>
<td>$O(1)$</td>
<td>Yes</td>
</tr>
<tr>
<td>Shell</td>
<td>$O(N^{3/2})$</td>
<td>?</td>
<td>$O(1)$</td>
<td>No</td>
</tr>
<tr>
<td>Heap</td>
<td>$O(N \log N)$</td>
<td>$O(N \log N)$</td>
<td>$O(1)$</td>
<td>No</td>
</tr>
<tr>
<td>Merge</td>
<td>$O(N \log N)$</td>
<td>$O(N \log N)$</td>
<td>$O(N)/O(1)$</td>
<td>Yes/No</td>
</tr>
<tr>
<td>Quick</td>
<td>$O(N^2)$</td>
<td>$O(N \log N)$</td>
<td>$O(\log N)$</td>
<td>No</td>
</tr>
</tbody>
</table>
Reading

* http://www.sorting-algorithms.com/
* Weiss Chapter 7