6.006 Lecture 11: Sorting IV

- Radix Sort
- A Broader Perspective: Bucket Sort
  - Quick Sort
  - Order Statistics (median)
- CLRS
  - 8.3
  - 8.4
  - Ch 7
  - Ch 9
Radix Sort

- Key idea: View input integers as multidigit numbers in some convenient base (10 for exposition, larger in practice, e.g. base 1024, or base 1,000,000, etc).
- Use counting sort on each digit.
- Intuitively, we might sort on most significant digit first; this is not how Radix sort works; too many invocation of counting sort.

```
<table>
<thead>
<tr>
<th>ex:</th>
<th>first round</th>
<th>second round</th>
<th>sorted</th>
</tr>
</thead>
<tbody>
<tr>
<td>329</td>
<td>329</td>
<td>329</td>
<td>329</td>
</tr>
<tr>
<td>457</td>
<td>355</td>
<td>355</td>
<td>355</td>
</tr>
<tr>
<td>657</td>
<td>457</td>
<td>436</td>
<td>436</td>
</tr>
<tr>
<td>839</td>
<td>436</td>
<td>657</td>
<td>657</td>
</tr>
<tr>
<td>436</td>
<td>657</td>
<td>457</td>
<td>457</td>
</tr>
<tr>
<td>720</td>
<td>720</td>
<td>720</td>
<td>720</td>
</tr>
<tr>
<td>355</td>
<td>839</td>
<td>839</td>
<td>839</td>
</tr>
</tbody>
</table>
```

- Radix sort does it the other way around.

```
<table>
<thead>
<tr>
<th>ex:</th>
<th>first round</th>
<th>second round</th>
<th>sorted</th>
</tr>
</thead>
<tbody>
<tr>
<td>329</td>
<td>720</td>
<td>720</td>
<td>329</td>
</tr>
<tr>
<td>457</td>
<td>355</td>
<td>329</td>
<td>355</td>
</tr>
<tr>
<td>657</td>
<td>436</td>
<td>436</td>
<td>436</td>
</tr>
<tr>
<td>839</td>
<td>457</td>
<td>839</td>
<td>457</td>
</tr>
<tr>
<td>436</td>
<td>657</td>
<td>657</td>
<td>657</td>
</tr>
<tr>
<td>720</td>
<td>329</td>
<td>720</td>
<td>720</td>
</tr>
<tr>
<td>355</td>
<td>839</td>
<td>839</td>
<td>839</td>
</tr>
</tbody>
</table>
```

- Stability of counting sort is key; would not work otherwise.
Correctness of Radix Sort

- Consider two numbers in the input:
  
  6543210
  2345678
  2335978

- The numbers would be put into the correct order when we sort on digit 4; we may have changed their order earlier, but it does not matter.

- When we sort on digits 5 & 6, the numbers retain their (correct) relative positions, because counting sort is stable.

Running time of Radix Sort

- For $d$-digit decimal numbers: $\Theta(d(n+10))$
- For length $l$ ASCII strings: $\Theta(l(n+128))$

- In general, we can choose the radix: assume inputs are $b$-bit binary numbers

  1011010001

  - radix-2, range of “digits” is 0-1
  - radix-4, range is 0-3
  - radix-32, range is 0-31

- Split inputs into $\frac{b}{r}$ r-bit “digits”: $\Theta(\frac{b}{r}(n+2^r))$

- The counting sort factor $(n+2^r)$ is asymptotically $O(n)$ for $b \leq \log n$
(but grows quickly afterwards; do not choose $r^* = 2\log n$, for example - this gives $2^r = n^2$)

*This is asymptotically optimal.*

$\Rightarrow$ inputs $\leq n^2$ : one pass $\frac{1}{2} = 1$ (just counting sort)

inputs $\leq n^2$ : two passes

inputs $\leq n^3$ : three passes

*Another perspective*: for $n = 1,000,000$, two passes sort inputs up to $10^{12}$, three up to $10^{18}$. Pretty good!

*Much more useful than counting sort alone.*
Another way to sort in linear time: **Bucket Sort**

Submission times of PS2A in nanoseconds before deadline:

We can split the range of input keys into **buckets**, place keys in buckets, sort each bucket (using mergesort or even insertion sort), and concatenate.

If we know the statistical distribution of the input keys and can split them into n buckets such that $E[\text{key in bucket}] = \frac{1}{n}$

$Pr[\text{key i falls in bucket j}] = \frac{1}{n}$

then we can sort using bucket sort in $\Theta(n)$
A Glimpse of Quicksort

- Another divide & conquer algorithm
  - Divide phase \(\rightarrow\) left half - right half
  - Sort halves recursively
  - Conquer \(\rightarrow\) merge (hard)

- Splitting can be done in-place (called parboil)
  - So concatenate phase does nothing

- The trick is to choose \(k\) so that \(\approx \frac{1}{2}\) elts are \(\leq A[k]\)
- Best techniques are randomized. Eg., choose 3 elts at random and take the middle of them
- Expected running time is \(\Theta(n \log n)\)
  - (can do deterministically in \(\Theta(n \log n)\), but more complicated and larger constants)
Order Statistics

- Once we sort a list, we can find max element in $O(n)$, also min, median, 14th, etc.
- We can find the min, max in $\Theta(n)$ time without sorting; can we do the same for the median? (View it as another way to beat the $\Omega(n \log n)$ comparison-based lower bound, this time by requesting less).
- Yes
- Easy with a randomized algorithm (same structure as quicksort, but at each step continue just with part that contains median

<table>
<thead>
<tr>
<th>$\leq A[k]$</th>
<th>$A[k]$</th>
<th>$\geq A[k]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>299 elts, ignore</td>
<td>Find 200th elt recursively</td>
<td>600 elements</td>
</tr>
</tbody>
</table>

- Can do without randomization, but harder.