Linear-time Sorting

Last time: Any comparison-based sorting alg has worst-case complexity $\Omega(n \log n)$

Can we sort in a way not based on comparing elements?
Consider sorting n elements that are all integers \( \{0, 1, \ldots, k-1\} \).

E.g. \( k = 10 \)

Idea: keep 10 lists and put all elements of value \( i \) into list \( i \).

Using an array: Count # of occurrences of each element and then you can put them in order.
Counting Sort

The basic idea we've just described.

We'll need one more property:

**Stable sort** - any two equivalent element are kept in same relative order.

\[d, a, b, c, b \rightarrow a, b, b, c, d\]
Radix Sort (input, output, k) {
  for d = 0 to #digits - 1
  int n = input.length;
  For (i = 0; i < k; i++)
    count[i] = 0;
  for (j = 0; j < n; j++)
    count[Input[j]]++;
  for (i = 1; i < k; i++)
    count[i] += count[i - 1];
  for (j = n - 1; j >= 0; j --)
    Output[---count[Input[j]]] = input[j];
  Radix sort digitizer::getDigit(input[j], d) current digit
$k = 3$

<table>
<thead>
<tr>
<th>Count</th>
<th>after 1st loop</th>
<th>after 2nd loop</th>
<th>after 3rd loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3 2 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3</td>
<td>6 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>8 7</td>
</tr>
</tbody>
</table>
Time complexity

Θ(n + k)

two loops over counters Θ(k)
two loops over elements Θ(n)

When \( k = O(n) \) then this

is a \( \Theta(n) \) sort.

linear time
Suppose I want to sort social security number
9 digits
What would \( k \) be if we wanted to use counting sort? \( 10^9 \)
As another example we might want to alphabetize names. General purpose sorting method
- provide a comparator
- another option is to provide Digitizer
Digitizer Interface

```java
int getBase() { // values b where digit values 0, ..., b-1
    words b = 26
    int numDigits(T x) { // digits in element Cab
        # digits = 3
        int getDigit(T x, int place) { // for x
            x = Cab
            getDigit(x, 0) = 1
            getDigit(x, 1) = 0
        }
    }
}
```
57981
Counting sort treats the element as just one digit

Radix Sort

From least to most significant digit \( d = 0 \) to \( b-1 \)
apply counting sort where we access digit \( d \)
Correctness Highlights.

Prove following holds using induction

After first $p$ phases, numbers
when looking at least significant
$p$ digits are sorted

correctness of base: true after 1 phase by counting sort

inductive step: combines inductive hyp.
correctness of counting sort & stability of counting sort
What's the asymptotic time complexity?

\[ N \quad \text{# elements} \]
\[ d \quad \text{max # of digits of elements} \]
\[ b \quad \text{base of each digit} \ (0, 1, \ldots, b-1) \]
\[ \Theta \left( d \left( n + b \right) \right) \]
Return to the ex.

\[ N = \# \text{ social security } \#s \text{ to sort} \]
\[ d = 9 \] treat each base-10 digit as a digit for radix sort
\[ b = 10 \]

\[ \text{\Hh}(9(n+10)) = \text{\Hh}(n) \]
\( \mathbb{O}(9(n+10)) \)

- XXX XXX XXX
  - base 1000 digit digit number

\( \ldots 0\ldots 999 \)

Time complexity with this digitizer

\( \mathbb{O}(3(n+1000)) \)

Roughly 3 times faster for "large" n
Consider sorting #s that begin in binary (base 2)

$n$ #s

$b$ bits ($b = 32$, 32-bit #)

group $k$ bits into a digit

$\# digits = \frac{b}{k}$  \hspace{1cm} base per digit = $2^k$