Goals

Know the meaning of a **comparison sort**.

Be able to execute **LSD radix sort** on paper.

Be prepared to implement LSD radix sort using **bucket sort** in the inner loop.
Comparison Sorts
(or "comparison-based sorting algorithms")

A comparison sort sorts values by comparing pairs of elements.

For example: all the sorts we've covered so far!

Fact:

• $O(n \log n)$ is the best possible worst-case runtime for a comparison-based sorting algorithm.
• It's mathematically impossible to do better!

...but is there any other way to do it?
*if your values have a constant \( O(1) \) number of digits
LSD Radix Sort
(Least-Significant-Digit)

/** least significant digit radix sort A */

LSDRadixSort(A):
  max_digits = max # digits in any element of A
  for d in 0..max_digits:
    stably sort A on the dth least significant digit

  // A is now sorted(!)

  ones place, then
tens place, then
hundreds place, and so on
Does this work?

/** least significant digit radix sort A */
LSDRadixSort(A):
    max_digits = max # digits in any element of A
    for d in 0..max_digits:
        stably sort A on the dth least significant digit

// A is now sorted(!)

[45, 26, 42, 32]
Sorted on ones: [42, 32, 45, 26]
Sorted on tens: [26, 32, 42, 45]
Why does this work?

```c
/** least significant digit radix sort A */
LSDRadixSort(A):
    max_digits = max # digits in any element of A
    for d in 0..max_digits:
        stably sort A on the dth least significant digit

// A is now sorted(!)
```

Intuition: if we're sorting 3-digit numbers,
• sort on 100's place **last**
• 100's-place ties yield to the already-sorted 10's place
• Works because **stability** preserves orderings from (already sorted) less significant digits in case of ties.
That's well and good, but...

/** least significant digit radix sort A */
LSDRadixSort(A):
    max_digits = max # digits in any element of A
    for d in 0..max_digits:
        stably sort A on the dth least significant digit
        ...how do we do this part?
    // A is now sorted(!)

Comparison sorts are O(n log n) at best.
To sort in O(n), we need something better...
How do you sort things without comparing them?

Suppose I asked you to sort 10 sticky notes with the digits 0 through 9.

What algorithm would you use?

4 3 9 1 5 0 7 2 8 6
How do you sort things without comparing them?

Suppose I asked you to sort 10 sticky notes with the digits 0 through 9.

What algorithm would you use?

What algorithm would minimize the number of times you look at each sticky note?
How do you sort things without comparing them?

Suppose I asked you to sort 10 sticky notes with the digits 0 through 9.

What algorithm would you use?

What algorithm would minimize the number of times you look at each sticky note?

What if there are duplicates?
Example: Radix sort this

[7, 19, 61, 11, 14, 54, 1, 08]

Buckets on 1’s place:

Sorted on 1’s place:

Buckets on 10’s place:

Sorted on 10’s place:
Example: Radix sort this

[07, 19, 61, 11, 14, 54, 01, 08]

Buckets on 1’s place:

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

Sorted on 1’s place:

Buckets on 10’s place:

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

Sorted on 10’s place:
## Example: Radix sort this

```
[7, 19, 61, 11, 14, 54, 1, 8]
```

### Buckets on 1’s place:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td>54</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>61</td>
<td></td>
<td>14</td>
<td></td>
<td></td>
<td>07</td>
<td></td>
<td>08</td>
<td>19</td>
</tr>
</tbody>
</table>

### Sorted on 1’s place:

```
61 11 01 14 54 07 08 19
```

### Buckets on 10’s place:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>08</td>
<td></td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>07</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td>54</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Sorted on 10’s place:

```
01 07 08 11 14 19 54 61
```
Try it out yourself: https://visualgo.net/en/sorting
Radix sort using bucket queues

Pseudocode adapted from visualgo.net:

LSDRadixSort(A):
    create a bucket (queue) for each digit (0 to 9)
    for each digit (least- to most-significant):
        for each element in A:
            enqueue element into its bucket based on digit
    for each bucket, starting from smallest digit
        while bucket is non-empty
            dequeue element into list
Counting Sort

Bucket sort is not in-place: requires $O(n)$ storage.

Counting sort is an in-place alternative: requires only $O(d)$ extra storage.

Intuition:

http://www.cs.miami.edu/home/burt/learning/Csc517.091/workbook/countingsort.html

Pseudocode in CLRS (reproduced on the next slide).
Counting Sort - from CLRS

Notes:
- $k$ is the base or radix (10 in our examples)
- $B$ is filled with the sorted values from $A$.
- $C$ maintains counts for each bucket.
- The final loop must go back-to-front to guarantee stability.

**COUNTING-SORT**($A$, $B$, $k$)

1. let $C[0..k]$ be a new array
2. for $i = 0$ to $k$
3. \hspace{1em} $C[i] = 0$
4. for $j = 1$ to $A.length$
5. \hspace{1em} $C[A[j]] = C[A[j]] + 1$
6. // $C[i]$ now contains the number of elements equal to $i$.
7. for $i = 1$ to $k$
8. \hspace{1em} $C[i] = C[i] + C[i - 1]$
9. // $C[i]$ now contains the number of elements less than or equal to $i$.
10. for $j = A.length$ downto 1
11. \hspace{1em} $B[C[A[j]]] = A[j]$
12. \hspace{1em} $C[A[j]] = C[A[j]] - 1$