Design & Analysis of Algorithms

Bucket Sort & Radix Sort

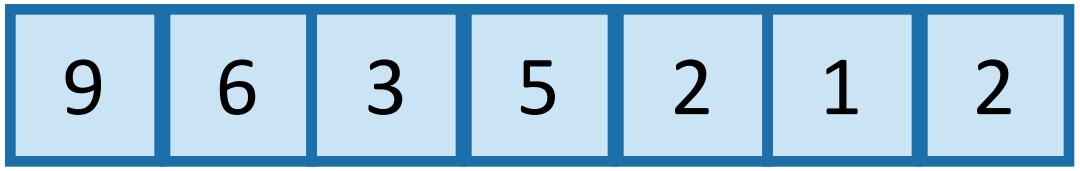
Prof. Wilson Rivera

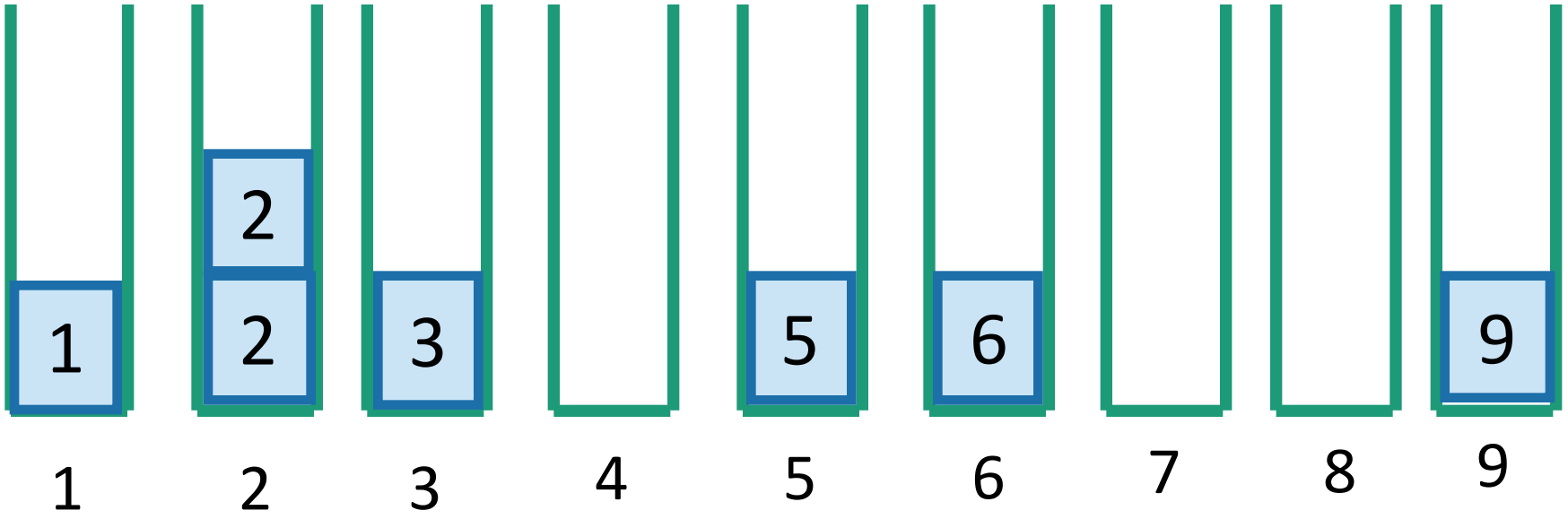
Spring 2018

Why might this help?

Implement the buckets as linked lists. They are first-in, first-out.

BucketSort:





Concatenate the buckets!

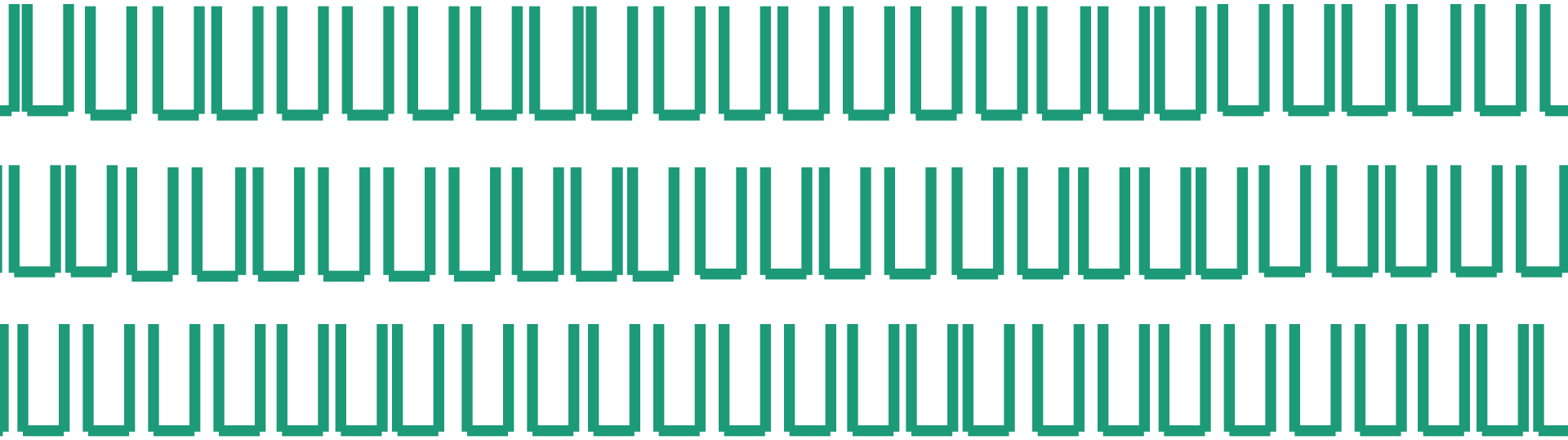
SORTED!

In time O(n).

Issues

- Need to be able to know what bucket to put something in.
  - That's okay for now: it's part of the model.
- Need to know what values might show up ahead of time.

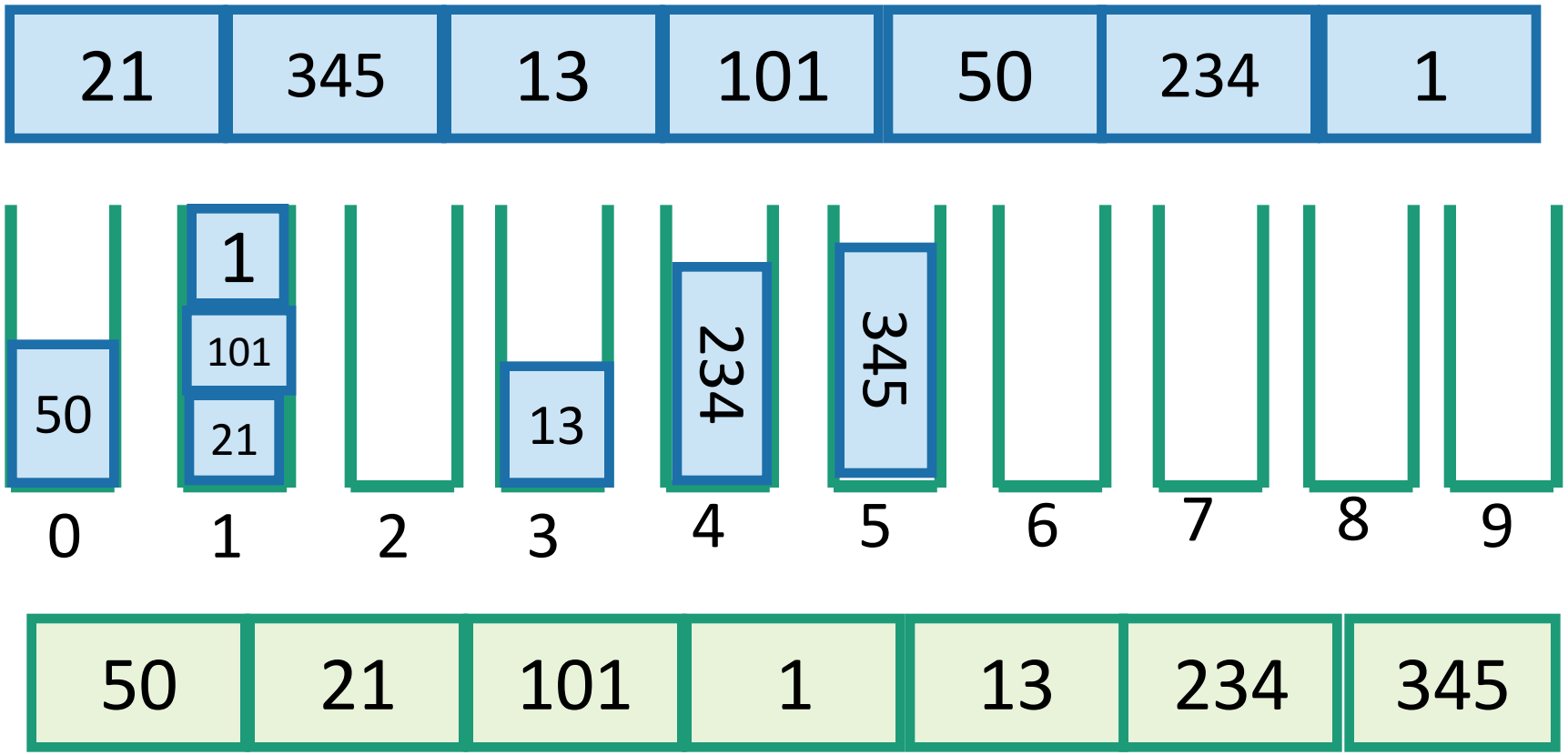
2	12345	13	2^{1000}	50	100000000	1



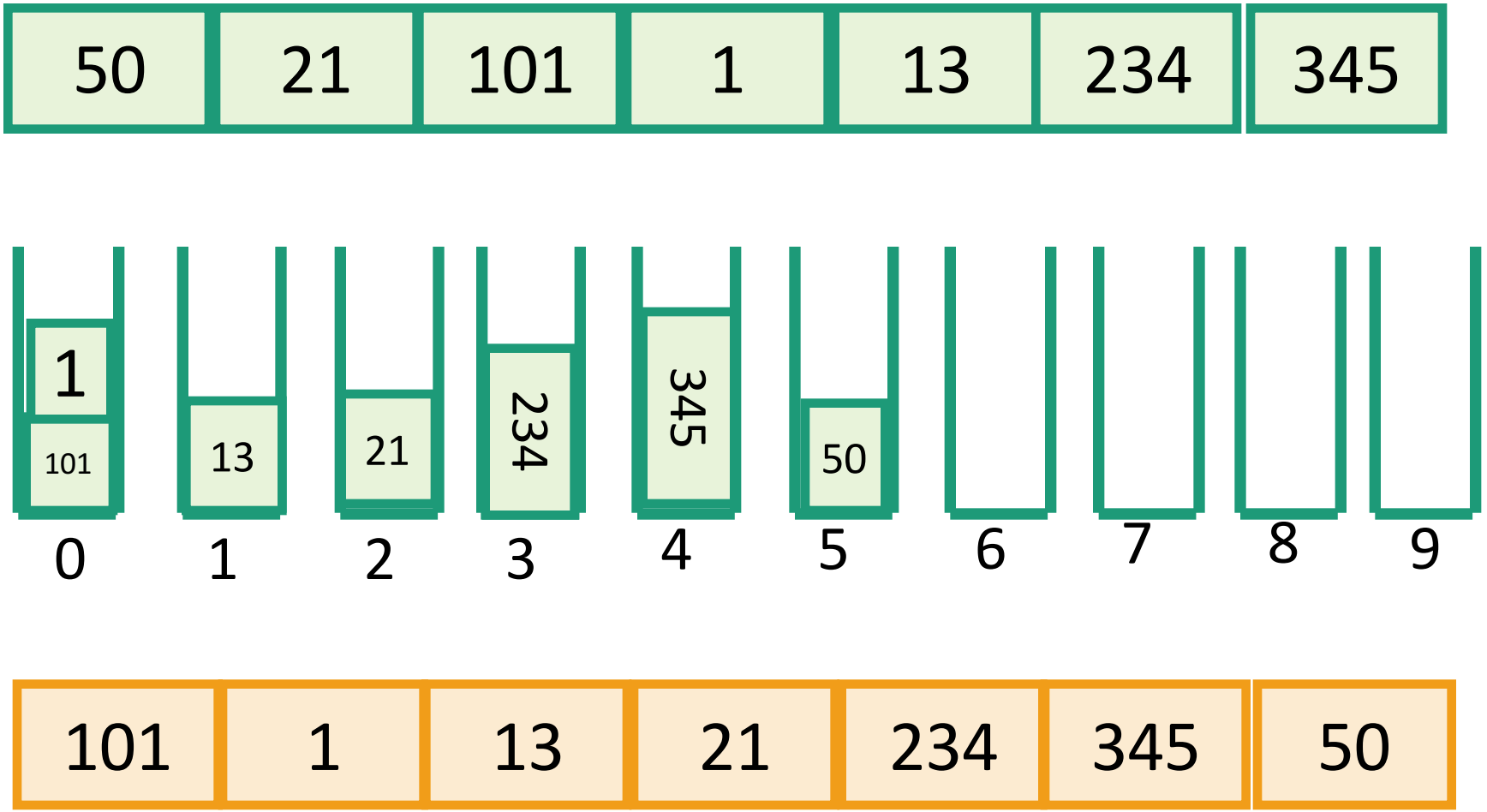
One solution: RadixSort

- Idea: BucketSort on the least-significant digit first, then the next least-significant, and so on.

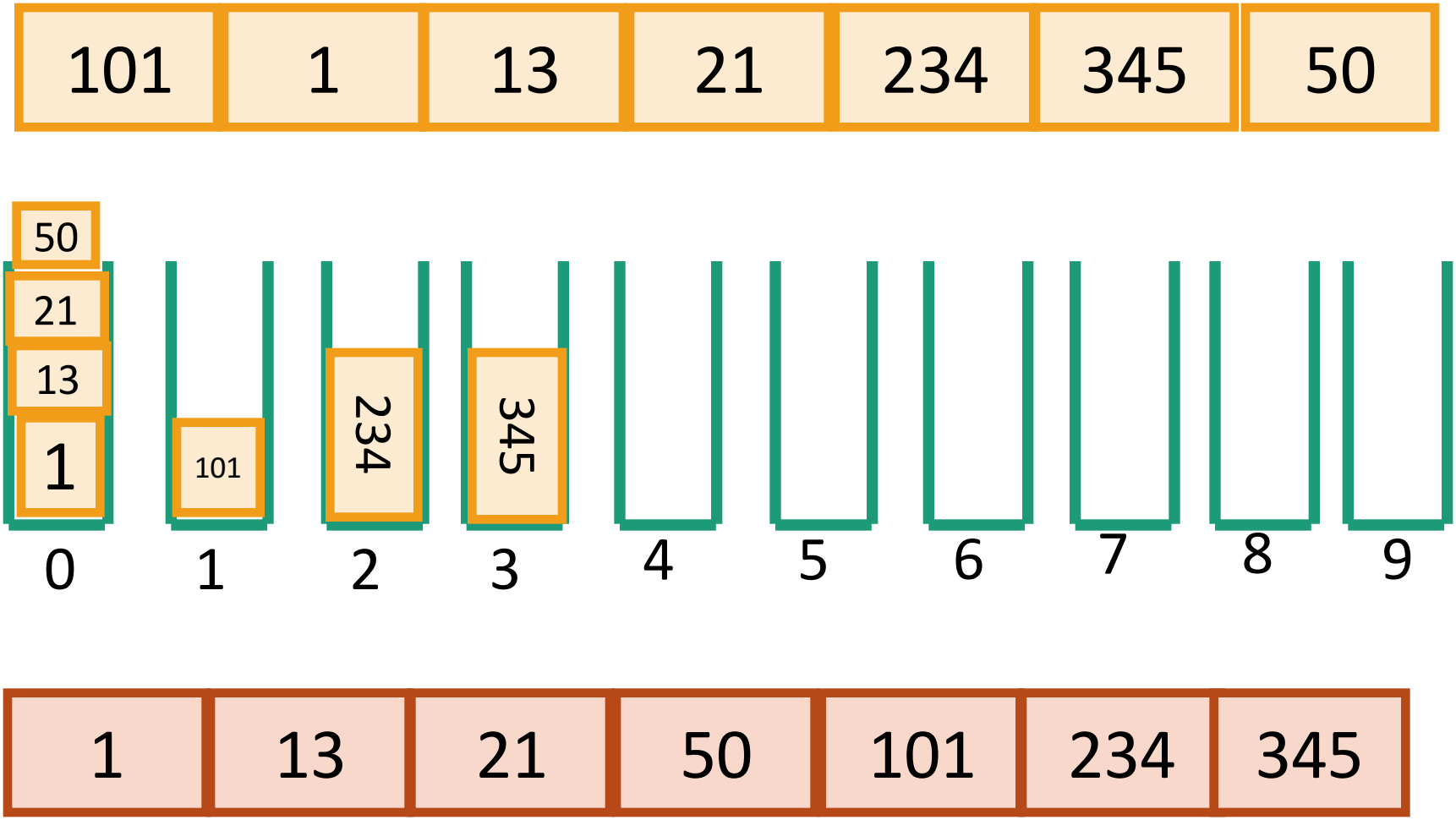
Step 1: BucketSort on LSB:



Step 2: BucketSort on the 2nd digit



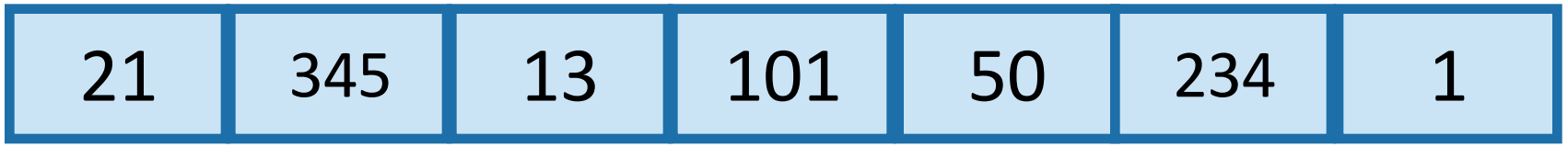
Step 3: BucketSort on the 3rd digit



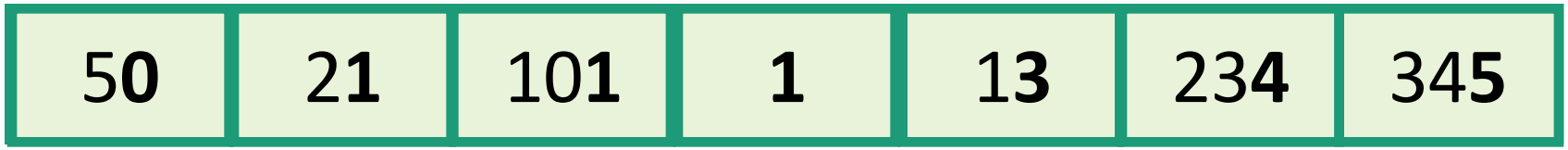
It worked!!

Why does this work?

Original array:



Next array is sorted by the first digit.



Next array is sorted by the first two digits.

Next array is sorted by all three digits.

Sorted array

Correctness

- Argument via loop invariant (aka induction).
- Loop Invariant:
  - After the k'th iteration, the array is sorted by the first k least-significant digits.
- Base case:
  - "Sorted by 0 least-significant digits" means not sorted.
- Inductive step:
  - (You fill in…)
- Termination:
  - After the d'th iteration, the array is sorted by the d least-significant digits. Aka, it's sorted.

This needs to use: (1) bucket sort works, and (2) we treat each bucket as a FIFO queue.

Running time

- Depends on how many digits the biggest number has.
  - Say d-digit numbers.
- There are d iterations
- Each iteration takes time $O(n + 10)$
  - We can change the 10 into an "r:" this is the "radix"
  - Example: if r = 2, we write everything in binary and only have two buckets.
  - Example: If r = 10000000, we write everything base-10000000 and have 10000000 buckets.
  - Example: if r = n, we write everything in base-n and have n buckets.
- Time is $O(d(n+r))$.
- If $d = O(1)$ and $r = O(n)$, running time $O(n)$.

So this is a O(n)-time sorting algorithm!

Why would we ever use a comparison-based sorting algorithm?

- d might not be "constant." (aka, it might be big)

π	$\frac{123456}{987654}$	e	140!	2.1234123	2^n	42

  - We can compare these pretty quickly (just look at the most-significant digit):
    - π = 3.14….
    - e = 2.78….
  - But to do RadixSort we'd have to look at every digit.
  - This is especially problematic since both of these have infinitely many digits…

- RadixSort needs extra memory for the buckets.
  - Not in-place
- I want to sort emoji by talking to a genie.
  - RadixSort makes more assumptions on the input.

Summary

- How difficult a problem is depends on the model of computation.
- How reasonable a model of computation is is up for debate.

- RadixSort can sort smallish integers in O(n) time.
- If we want to sort emoji (or arbitrary-precision numbers), we require Ω(nlog(n)) time (like MergeSort).