Lecture 12: Non-Comparison Sorting

CSE 332: Data Structures & Parallelism

Yafqa Khan

Summer 2025

Announcements

- EX04: AVL
  - Due Today
- EX05: Hashing
  - Due next Monday
- Exam 1: This Friday!

Today

- Special Sorting Algorithm 1: Bucket Sort
- Special Sorting Algorithm 2: Radix Sort
- Sorting Summary

Special Sorting Algorithm 1: Bucket Sort

- Also called Bin Sort
- Intuition: Small range of integers, get a tally
- Algorithm:
  1. Find the `min` and `max` value
  2. Make an aux array to represent the range between `min` and `max`
  3. Go through the original array and start tallying each number
  4. Copy the aux into the original array

Bucket Sort: Visualization

size = n

4	3	1	2	1	1	2	3	4	2

```
min = 1
max = 4
```

1	2	3	4

size = k

size = n

1	1	1	2	2	2	3	3	4	4

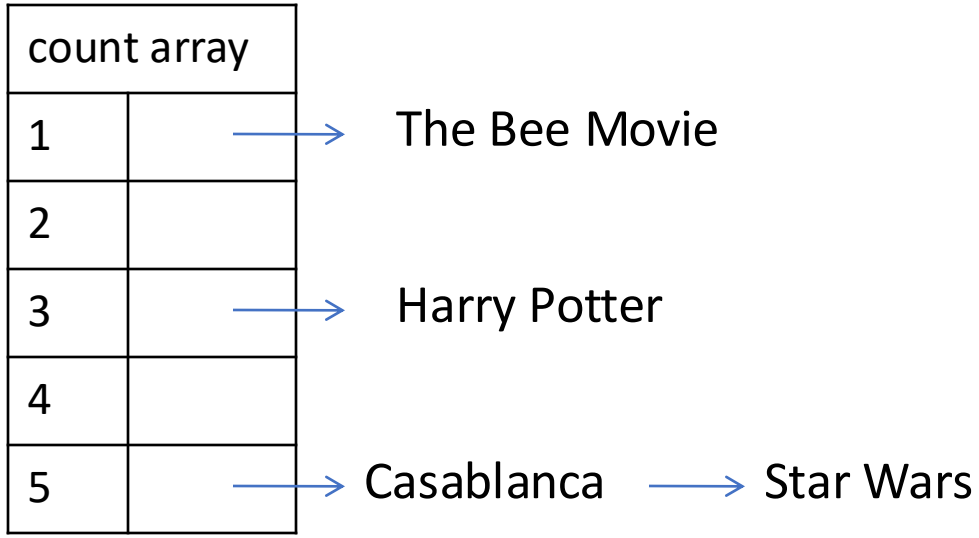
Any Questions?

Bucket Sort: Analysis

1. Stable?
   - Yes!
2. In-Place?
   - No
3. Fast?
   - Yes! (in terms of asymptotics)
     - Let $k =$ range between `min` and `max` value
     - Worst Case: $\mathcal{O}(n + k)$
       - Note: k can be bigger than n
   - **Good constant factors!**

Bucket Sort: Non Integers

- Most real lists aren't just #'s; we have data
- Each bucket is a list (say, linked list)
- To add to a bucket, place at end $\mathcal{O}(1)$ (keep pointer to last element)



- Example: Movie ratings: 1=bad,… 5=excellent
- Input=

  5: Casablanca

  3: Harry Potter movies

  1: The Bee Movie

  5: Star Wars

Result: 1: Rocky V, 3: Harry Potter, 5: Casablanca, 5: Star Wars
This result is stable; Casablanca still before Star Wars

Today

- Special Sorting Algorithm 1: Bucket Sort
- Special Sorting Algorithm 2: Radix Sort
- Sorting Summary

Special Sorting Algorithm 2: Radix Sort

- Intuition: Exploit integer digits
- Algorithm:
  1. For each digit,

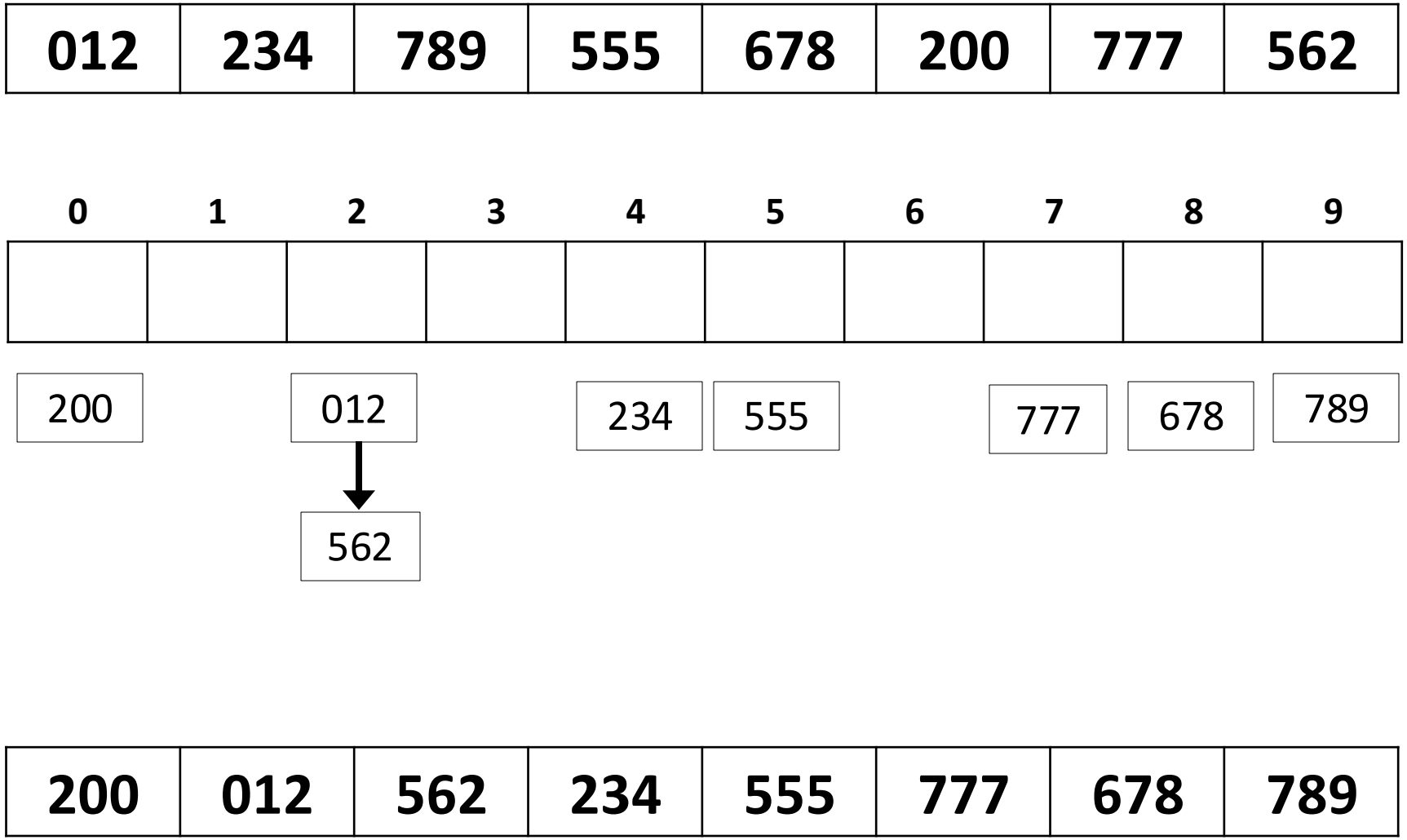
     Start from the **Least Significant Digit** (LSD) to **Most Significant Digit** (MSD)
     - Run bucket sort on just that digit (e.g., just the 1s place)

Radix Sort: Visualization (1s Place)

012	234	789	555	678	200	777	562

0	1	2	3	4	5	6	7	8	9

Radix Sort: Visualization (1s Place)

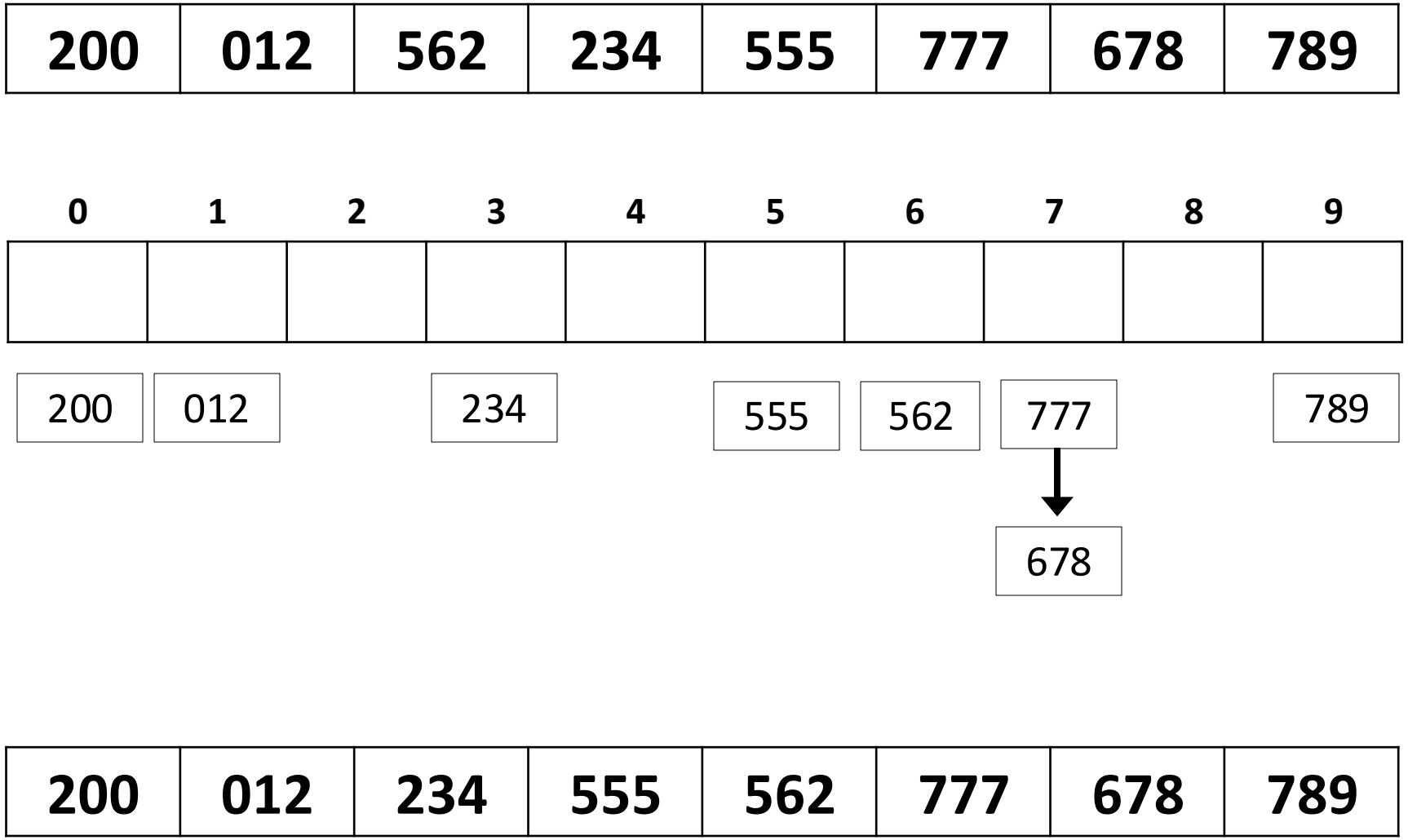


Radix Sort: Visualization (10s Place)

200	012	562	234	555	777	678	789

0	1	2	3	4	5	6	7	8	9

Radix Sort: Visualization (10s Place)

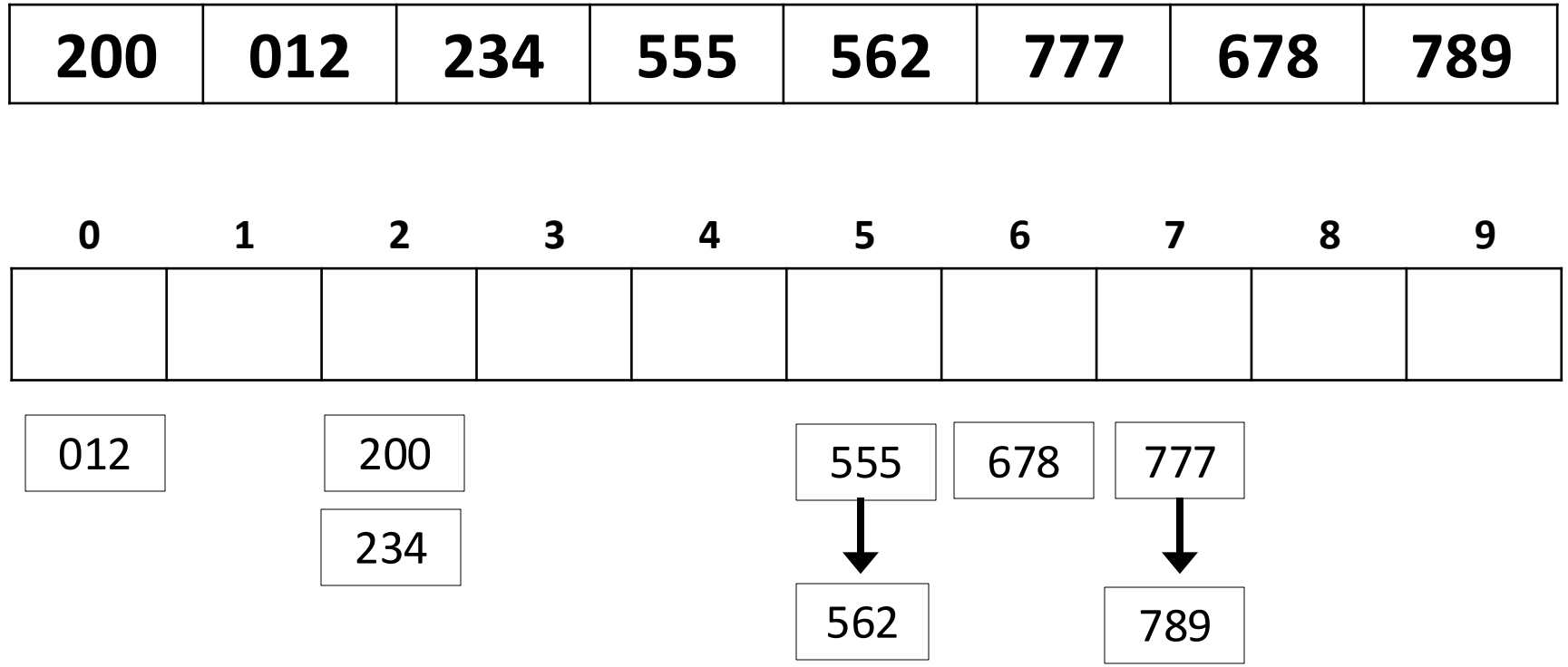


Radix Sort: Visualization (100s Place)

200	012	234	555	562	777	678	789

0	1	2	3	4	5	6	7	8	9

Radix Sort: Visualization (100s Place)



Any Questions?

Radix Sort: Analysis

1. Stable?
   - Yes!
2. In-Place?
   - No
3. Fast?
   - Yes! (in terms of asymptotics)
     - Let $n =$ input size
     - $d =$ number of digits (e.g., previous example is 3 digits because only until 100s place)
     - $k =$ range of values for each digit (e.g., 10 for base-10 integers)
     - Worst Case: $\mathcal{O}\big(d \cdot (n + k)\big)$
       - Note: k can be bigger than n
   - **Good constant factors!**

Radix Sort: Runtime

- Let $n =$ input size
- $d =$ number of digits (e.g., previous example is 3 digits because only until 100s place)
- $k =$ range of values for each digit (e.g., 10 for base-10 integers)
- Worst Case: $\mathcal{O}\big(d \cdot (n + k)\big)$
  - How many passes? d number of passes
  - Work per pass = 1 Bucket Sort = $\mathcal{O}(n + k)$

Today

- Special Sorting Algorithm 1: Bucket Sort
- Special Sorting Algorithm 2: Radix Sort
- Sorting Summary

Sorting: Comparisons

	Run-time	Stable?	In-Place?
Insertion Sort	Best Case: $\mathcal{O}(n)$ Worst Case: $\mathcal{O}(n^2)$ Average Case: $\mathcal{O}(n^2)$	Stable	In-place
Selection Sort	$\mathcal{O}(n^2)$	Not Stable	In-place
Heap Sort	Best Case (all equal): $\mathcal{O}(n)$ Best Case (all distinct): $\mathcal{O}(n \log n)$ Worst Case: $\mathcal{O}(n \log n)$	Not Stable	In-place
Merge Sort	$\mathcal{O}(n \log n)$	Stable	Not In-place
Quick Sort ("Hoare" Partition)	Best Case: $\mathcal{O}(n \log n)$ Worst Case: $\mathcal{O}(n^2)$ Average Case: $\mathcal{O}(n \log n)$	Not Stable	In-place
Bucket Sort	$\mathcal{O}(n + k)$	Stable	Not In-place
Radix Sort	$\mathcal{O}\big(d \cdot (n + k)\big)$	Stable	Not In-place

Sorting: Summary

- Simple $\mathcal{O}(n^2)$ Sorting
  - Selection Sort, Insertion Sort, etc.
  - Good for "below a cut-off" (e.g., for small input size n) to help divide-and-conquer sorts
- $\mathcal{O}(n \log n)$ Sorting
  - Heap Sort, Not stable but in-place, not parallelizable (later)
  - Merge Sort, Stable but not in-place and works as external sort
  - Quick Sort, Not stable but in-place and $\mathcal{O}(n^2)$ in worst-case
    - often fastest, but depends on costs of comparisons/copies (Java uses this)
- $\Omega(n \log n)$ lower-bound for comparison sorting
- Non-Comparison Sorting
  - Bucket Sort for small number of $\texttt{k}-\texttt{v}$
  - Radix Sort for digits
- Best way to sort? lol depends

Any Questions?