Lecture 36 (Sorting 5)

Radix Sorts

CS61B, Spring 2025 @ UC Berkeley

Slides credit: Josh Hug

Sorting Stability

Lecture 36, CS61B, Spring 2025

Sorting Summary (so far)

Listed by mechanism:

- Selection sort: Find the smallest item and put it at the front.
- Insertion sort: Figure out where to insert the current item.
- Merge sort: Merge two sorted halves into one sorted whole.
- Partition (quick) sort: Partition items around a pivot.

Listed by memory and runtime:

	Memory	# Compares	Notes
Heapsort	Θ(1)	Θ(N log N) worst	Bad caching (61C)
Insertion	Θ(1)	Θ(N^2) worst	Θ(N) if almost sorted
Mergesort	Θ(N)	Θ(N log N) worst	
Random Quicksort	Θ(log N) (call stack)	Θ(N log N) expected	Fastest sort

Other Desirable Sorting Properties: Stability

A sort is said to be stable if order of equivalent items is preserved.

sort(studentRecords, BY_NAME);

Bas	3
Fikriyya	4
Jana	3
Jouni	3
Lara	1
Nikolaj	4
Rosella	3
Sigurd	2

sort(studentRecords, BY_SECTION);

Lara	1
Sigurd	2
Bas	3
Jana	3
Jouni	3
Rosella	3
Fikriyya	4
Nikolaj	4

Equivalent items don’t ‘cross over’ when being stably sorted.

Other Desirable Sorting Properties: Stability

A sort is said to be stable if order of equivalent items is preserved.

sort(studentRecords, BY_NAME);

Bas	3
Fikriyya	4
Jana	3
Jouni	3
Lara	1
Nikolaj	4
Rosella	3
Sigurd	2

sort(studentRecords, BY_SECTION);

Lara	1
Sigurd	2
Jouni	3
Rosella	3
Bas	3
Jana	3
Fikriyya	4
Nikolaj	4

Sorting instability can be really annoying! Wanted students listed alphabetically by section.

Sorting Stability

Is insertion sort stable?

```
S O R T E X A M P L E
S O R T E X A M P L E   (0 swaps)
O S R T E X A M P L E   (1 swap )
O R S T E X A M P L E   (1 swap )
O R S T E X A M P L E   (0 swaps)
E O R S T X A M P L E   (4 swaps)
E O R S T X A M P L E   (0 swaps)
A E O R S T X M P L E   (6 swaps)
A E M O R S T X P L E   (5 swaps)
A E M O P R S T X L E   (4 swaps)
A E L M O P R S T X E   (7 swaps)
A E E L M O P R S T X   (8 swaps)
```

Is Quicksort stable?

- Consider -------->

6	8	3	1	2	7	4

Sorting Stability

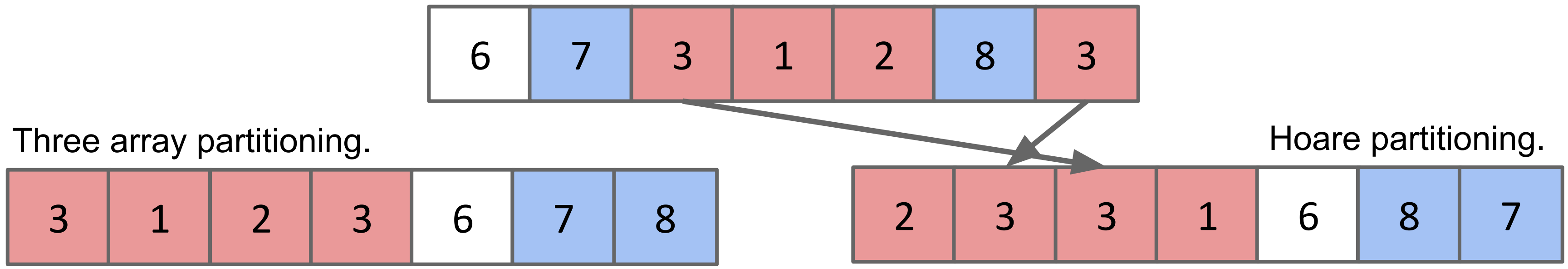
Is insertion sort stable?

- Yes.
- Equivalent items never move past their equivalent brethren.

Is Quicksort stable?

- Depends on your partitioning strategy.

```
S O R T E X A M P L E
S O R T E X A M P L E   (0 swaps)
O S R T E X A M P L E   (1 swap )
O R S T E X A M P L E   (1 swap )
O R S T E X A M P L E   (0 swaps)
E O R S T X A M P L E   (4 swaps)
E O R S T X A M P L E   (0 swaps)
A E O R S T X M P L E   (6 swaps)
A E M O R S T X P L E   (5 swaps)
A E M O P R S T X L E   (4 swaps)
A E L M O P R S T X E   (7 swaps)
A E E L M O P R S T X   (8 swaps)
```



Stability

	Memory	# Compares	Notes	Stable?
Heapsort	Θ(1)	Θ(N log N)	Bad caching (61C)	No
Insertion	Θ(1)	$\Theta(N^2)$	Θ(N) if almost sorted	Yes
Mergesort	Θ(N)	Θ(N log N)		Yes
Quicksort LTHS	Θ(log N)	Θ(N log N) expected	Fastest sort	No

This is due to the cost of tracking recursive calls by the computer, and is also an "expected" amount. The difference between log N and constant memory is trivial.

You can create a stable Quicksort (i.e. the version from the previous lecture). However, unstable partitioning schemes (like Hoare partitioning) tend to be faster. All reasonable partitioning schemes yield Θ(N log N) expected runtime, but with different constants.

Arrays.sort

In Java, Arrays.sort(someArray) uses:

- Mergesort (specifically the TimSort variant) if someArray consists of Objects.
- Quicksort if someArray consists of primitives.

Why? See A level problems.

`static void`	`sort(Object[] a)` Sorts the specified array of objects into ascending order, according to the **natural ordering** of its elements.

`static void`	`sort(int[] a)` Sorts the specified array into ascending numerical order.

Arrays.sort

In Java, Arrays.sort(someArray) uses:

- Mergesort (specifically the TimSort variant) if someArray consists of Objects.
- Quicksort if someArray consists of primitives.

Why?

- Quicksort isn't stable, but there's only one way to order them. Wouldn't have multiple types of orders.
    - Could sort by other things, say sum of the digits.
    - Order by number of digits.
        - My usual answer: 5 is just 5. There's no different possible 5s.

Arrays.sort

In Java, Arrays.sort(someArray) uses:

- Mergesort (specifically the TimSort variant) if someArray consists of Objects.
- Quicksort if someArray consists of primitives.

Why?

- When you are using a primitive value, they are the ‘same’. A 4 is a 4. Unstable sort has no observable effect.
  - There’s really only one natural order for numbers, so why not just assume that’s the case and sort them that way.
- By contrast, objects can have many properties, e.g. section and name, so equivalent items CAN be differentiated.
  - If you know there’s only one way, can you force Java to use Quicksort?

Optimizing Sorts

Additional tricks we can play:

- Switch to insertion sort:
  - When a subproblem reaches size 15 or lower, use insertion sort.
- Make sort ***adaptive***: Exploit existing order in array (Insertion Sort, SmoothSort, TimSort (*the* sort in Python and Java)).
- Exploit restrictions on set of keys. If number of keys is some constant, e.g. [3, 4, 1, 2, 4, 3, …, 2, 2, 2, 1, 4, 3, 2, 3], can sort faster (see 3-way quicksort -- if you're curious, see: http://goo.gl/3sYnv3).
- For Quicksort: Make the algorithm introspective, switching to a different sorting method if recursion goes too deep. Only a problem for deterministic flavors of Quicksort.

Today's Two New Ideas

Today we'll cover two new ideas:

- Digit-by-Digit Sorting
  - A procedure that uses a sort (e.g. Merge Sort, Quicksort, Counting Sort).
  - Using the word "sort" is arguably a misnomer.
    - Digit-by-digit sorting is a process that uses another sort as a subroutine.
- Counting Sort
  - A new type of sort that competes with Merge Sort, Quicksort, Heap Sort, Insertion Sort, Selection Sort, etc.
  - Unlike these other sorts, Counting Sort does not use compareTo.

Warmup: Digit-by-digit Sorting

Lecture 36, CS61B, Spring 2025

Digit-by-digit Sorting

As a warmup to the later part of today's lecture. Suppose we have a list of integers we want to sort.

- Suppose we first sort by **only the rightmost digit.**

22	41
34	41
41	31
53	32
23	22
41	12
32	12
34	42
12	…
31	
12	
42	

Digit-by-digit Sorting

As a warmup to the later part of today's lecture. Suppose we have a list of integers we want to sort.

- Suppose we first sort by **only the rightmost digit.**

22	41
34	41
41	31
53	32
23	22
41	12
32	12
34	42
12	…
31	
12	
42	

What are the 4 integers at the end of the array?

Digit-by-digit Sorting

As a warmup to the later part of today's lecture. Suppose we have a list of integers we want to sort.

- Suppose we first sort by **only the rightmost digit.**

22
34
41
53
23
41
32
34
12
31
12
42

→

41
41
31
32
22
12
12
42
53
23
34
34

Digit-by-digit Sorting

As a warmup to the later part of today's lecture. Suppose we have a list of integers we want to sort.

- Suppose we first sort by **only the rightmost digit.**

22
34
41
53
23
41
32
34
12
31
12
42

41
41
31
32
22
12
12
42
53
23
34
34

I put 53 and 23 in this order. Would they always be in this order?

Digit-by-digit Sorting

As a warmup to the later part of today's lecture. Suppose we have a list of integers we want to sort.

- Suppose we first sort by **only the rightmost digit.**

22	→	41
34		41
41		31
53		32
23		22
41		12
32		12
34		42
12		53
31		23
12		34
42		34

I put 53 and 23 in this order. Would they always be in this order?

- Not necessarily! Depends on if the sort I used is **stable**.
- Stable sort yields 53 then 23.
- Example: If I used Quicksort with shuffle, could have been 23 then 53.

Digit-by-digit Sorting

As a warmup to the later part of today's lecture. Suppose we have a list of integers we want to sort.

- Now suppose we sort by the **left digit** using a **stable sort**.

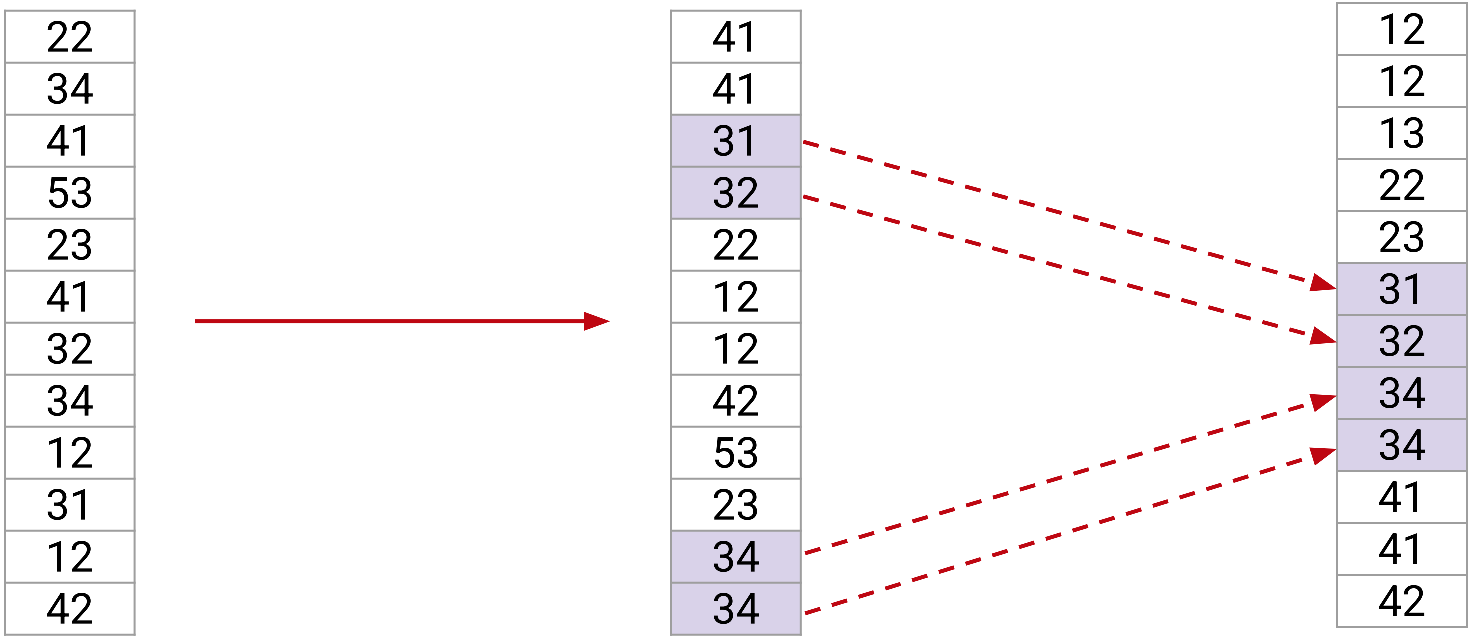
22	41	12
34	41	12
41	31	13
53	32	22
23	22	23
41	12	??
32	12	??
34	42	??
12	53	??
31	23	41
12	34	41
42	34	42

In what order will **31**, **32**, **34**, and **34** appear?

Digit-by-digit Sorting

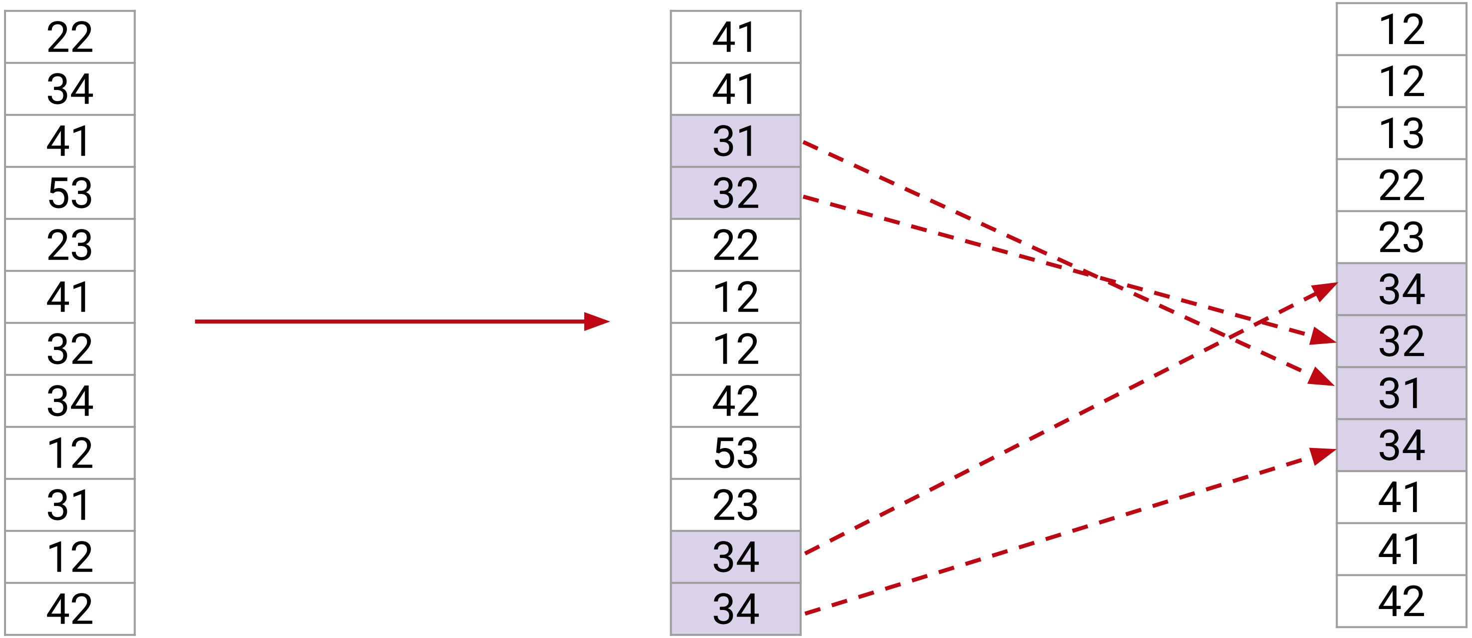
As a warmup to the later part of today's lecture. Suppose we have a list of integers we want to sort.

- Now suppose we sort by the **left digit** using a **stable sort**.



Digit-by-digit Sorting

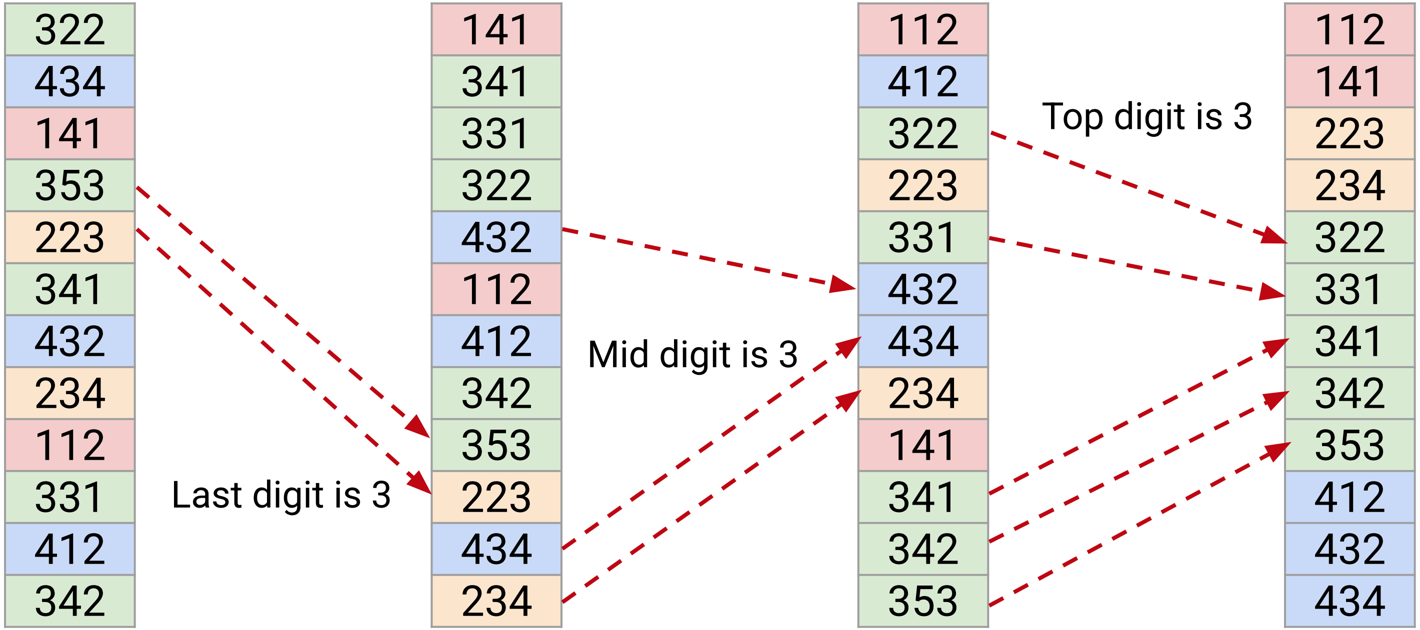
This procedure does not work if the sort subroutine is unstable.



Digit-by-digit Sorting

Example of a digit-by-digit sort:

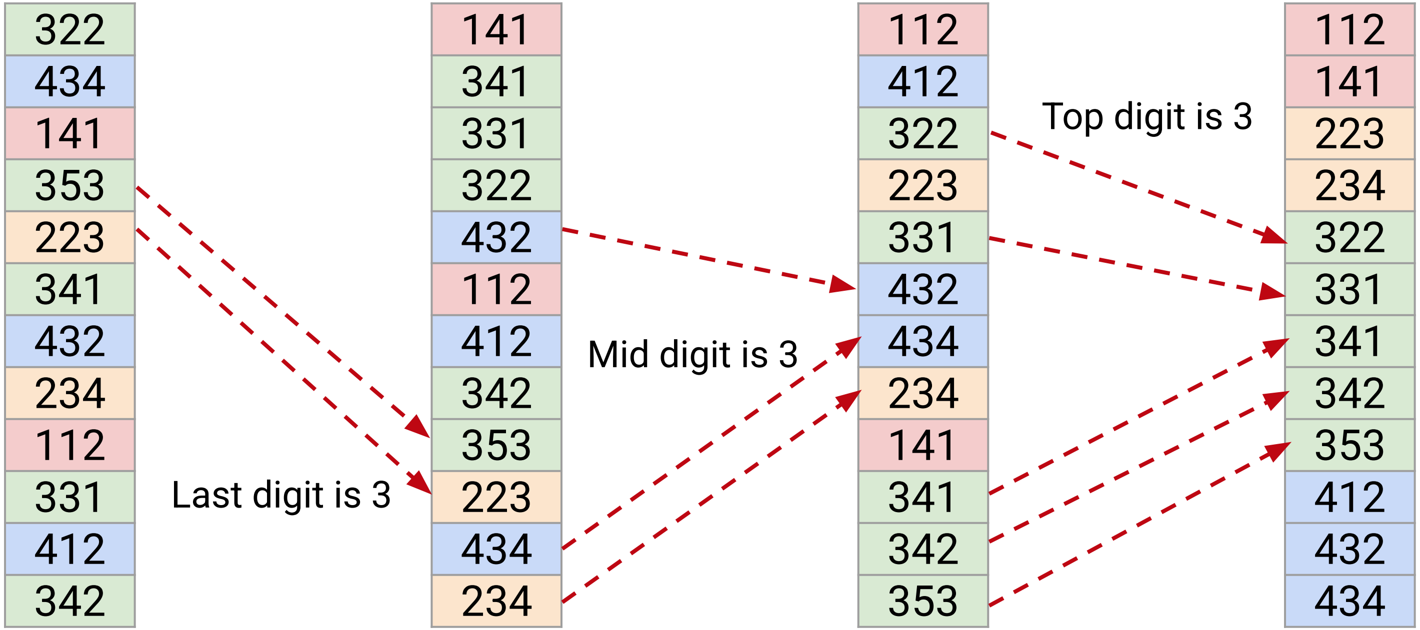
- Use a stable sort on each digit, moving from least to most significant.
- Result is guaranteed to correct!



Digit-by-digit Sorting

Two quick notes:

- No obvious reason why this procedure is useful (can just sort by entire integer)
- Other digit-by-digit sort procedures work.



We'll come back to digit-by-digit sorting later!

The Counting Sort Algorithm

Lecture 36, CS61B, Spring 2025

Sorting Stability

Warmup: Digit-by-digit Sorting

Counting Sort

- **The Counting Sort Algorithm**
- Runtime

Radix Sorts:

- LSD Radix Sort
- MSD Radix Sort

Comparison Based Sorting

The key idea from our previous sorting lecture: Sorting requires $\Omega(N \log N)$ compares in the worst case.

- Thus, the ultimate comparison based sorting algorithm has a worst case runtime of $\Theta(N \log N)$.

From an asymptotic perspective, that means no matter how clever we are, we can never beat Merge Sort’s worst case runtime of $\Theta(N \log N)$.

- ...but what if we don’t compare at all?

Example #1: Sleep Sort (for Sorting Integers) (not actually good)

For each integer x in array A, start a new program that:

- Sleeps for x seconds.
- Prints x.

All start at the same time.

Runtime:

- N + max(A)

The catch: On real machines, scheduling execution of programs must be done by an operating system. In practice requires list of running programs sorted by sleep time.

Genius sorting algorithm: Sleep sort

1 Name: **Anonymous** 2011-01-20 12:22

Man, am I a genius. Check out this sorting algorithm I just invented.

```
#!/bin/bash
function f() {
    sleep "$1"
    echo "$1"
}
while [ -n "$1" ]
do
    f "$1" &
    shift
done
wait
```

example usage:

```
./sleepsort.bash 5 3 6 3 6 3 1 4 7
```

Invented by 4chan (?).

Example #2: Counting Sort: Exploiting Space Instead of Time

#			
5	Sandra	Vanilla	Grimes
0	Lauren	Mint	Jon Talabot
11	Lisa	Vanilla	Blue Peter
9	Dave	Chocolate	Superpope
4	JS	Fish	The Filthy Reds
7	James	Rocky Road	Robots are Supreme
3	Edith	Vanilla	My Bloody Valentine
6	Swimp	Chocolate	Sef
1	Delbert	Strawberry	Ronald Jenkees
2	Glaser	Cardamom	Rx Nightly
8	Lee	Vanilla	La(r)va
10	Bearman	Butter Pecan	Extrobophile

Assuming keys are unique integers 0 to 11.

Idea:

- Create a new array.
- Copy item with key i into ith entry of new array.

Example #2: Counting Sort: Exploiting Space Instead of Time

#			
5	Sandra	Vanilla	Grimes
0	Lauren	Mint	Jon Talabot
11	Lisa	Vanilla	Blue Peter
9	Dave	Chocolate	Superpope
4	JS	Fish	The Filthy Reds
7	James	Rocky Road	Robots are Supreme
3	Edith	Vanilla	My Bloody Valentine
6	Swimp	Chocolate	Sef
1	Delbert	Strawberry	Ronald Jenkees
2	Glaser	Cardamom	Rx Nightly
8	Lee	Vanilla	La(r)va
10	Bearman	Butter Pecan	Extrobophile

#			
5	Sandra	Vanilla	Grimes

Example #2: Counting Sort: Exploiting Space Instead of Time

#			
5	Sandra	Vanilla	Grimes
0	Lauren	Mint	Jon Talabot
11	Lisa	Vanilla	Blue Peter
9	Dave	Chocolate	Superpope
4	JS	Fish	The Filthy Reds
7	James	Rocky Road	Robots are Supreme
3	Edith	Vanilla	My Bloody Valentine
6	Swimp	Chocolate	Sef
1	Delbert	Strawberry	Ronald Jenkees
2	Glaser	Cardamom	Rx Nightly
8	Lee	Vanilla	La(r)va
10	Bearman	Butter Pecan	Extrobophile

#			
0	Lauren	Mint	Jon Talabot
5	Sandra	Vanilla	Grimes

Example #2: Counting Sort: Exploiting Space Instead of Time

#			
5	Sandra	Vanilla	Grimes
0	Lauren	Mint	Jon Talabot
11	Lisa	Vanilla	Blue Peter
9	Dave	Chocolate	Superpope
4	JS	Fish	The Filthy Reds
7	James	Rocky Road	Robots are Supreme
3	Edith	Vanilla	My Bloody Valentine
6	Swimp	Chocolate	Sef
1	Delbert	Strawberry	Ronald Jenkees
2	Glaser	Cardamom	Rx Nightly
8	Lee	Vanilla	La(r)va
10	Bearman	Butter Pecan	Extrobophile

#			
0	Lauren	Mint	Jon Talabot
5	Sandra	Vanilla	Grimes
11	Lisa	Vanilla	Blue Peter

Example #2: Counting Sort: Exploiting Space Instead of Time

#			
5	Sandra	Vanilla	Grimes
0	Lauren	Mint	Jon Talabot
11	Lisa	Vanilla	Blue Peter
9	Dave	Chocolate	Superpope
4	JS	Fish	The Filthy Reds
7	James	Rocky Road	Robots are Supreme
3	Edith	Vanilla	My Bloody Valentine
6	Swimp	Chocolate	Sef
1	Delbert	Strawberry	Ronald Jenkees
2	Glaser	Cardamom	Rx Nightly
8	Lee	Vanilla	La(r)va
10	Bearman	Butter Pecan	Extrobophile

#			
0	Lauren	Mint	Jon Talabot
1	Delbert	Strawberry	Ronald Jenkees
2	Glaser	Cardamom	Rx Nightly
3	Edith	Vanilla	My Bloody Valentine
4	JS	Fish	The Filthy Reds
5	Sandra	Vanilla	Grimes
6	Swimp	Chocolate	Sef
7	James	Rocky Road	Robots are Supreme
8	Lee	Vanilla	La(r)va
9	Dave	Chocolate	Superpope
10	Bearman	Butter Pecan	Extrobophile
11	Lisa	Vanilla	Blue Peter

Generalizing Counting Sort

We just sorted N items in Θ(N) worst case time.

- Avoiding yes/no questions lets us dodge our lower bound based on puppy, cat, dog!

Simplest case:

- Keys are unique integers from 0 to N-1.

More complex cases:

- Non-unique keys.
- Non-consecutive keys.
- Non-numerical keys.

Counting Sort

Alphabet case: Keys belong to a finite ordered alphabet.

- Example: {♣, ♠, ♥, ♦} (in that order)

♠	Lauren
♥	Delbert
♦	Glaser
♣	Edith
♠	JS
♦	Sandra
♥	Swimp
♥	James
♣	Lee
♥	Dave
♣	Bearman
♦	Lisa

0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		

Sorted

Question: What will be the index of the first ♥?

Counting Sort

Alphabet case: Keys belong to a finite ordered alphabet.

- Example: {♣, ♠, ♥, ♦} (in that order)

♠	Lauren
♥	Delbert
♦	Glaser
♣	Edith
♠	JS
♦	Sandra
♥	Swimp
♥	James
♣	Lee
♥	Dave
♣	Bearman
♦	Lisa

0	♣	
1	♣	
2	♣	
3	♠	
4	♠	
5		
6		
7		
8		
9		
10		
11		

Sorted

Question: What will be the index of the first ♥?

Implementing Counting Sort with Counting Arrays

Counting sort:

- Count number of occurrences of each item.
- Iterate through list, using count array to decide where to put everything.

Bottom line, we can use counting sort to sort N objects in $\Theta(N)$ time.

Counting Sort

Example:

- Alphabet: {♣, ♠, ♥, ♦}

♠	Lauren
♥	Delbert
♦	Glaser
♣	Edith
♠	JS
♦	Sandra
♥	Swimp
♥	James
♣	Lee
♥	Dave
♣	Bearman
♦	Lisa

Counting Sort

Example:

- Alphabet: {♣, ♠, ♥, ♦}

♠	Lauren
♥	Delbert
♦	Glaser
♣	Edith
♠	JS
♦	Sandra
♥	Swimp
♥	James
♣	Lee
♥	Dave
♣	Bearman
♦	Lisa

0	♣	3
1	♠	2
2	♥	4
3	♦	3

Counts

Counting Sort

Example:

- Alphabet: {♣, ♠, ♥, ♦}

♠	Lauren
♥	Delbert
♦	Glaser
♣	Edith
♠	JS
♦	Sandra
♥	Swimp
♥	James
♣	Lee
♥	Dave
♣	Bearman
♦	Lisa

0	♣	3
1	♠	2
2	♥	4
3	♦	3

Counts

0	♣	0
1	♠	3
2	♥	5
3	♦	9

Starting Points

Counting Sort

Example:

- Alphabet: {♣, ♠, ♥, ♦}

♠	Lauren
♥	Delbert
♦	Glaser
♣	Edith
♠	JS
♦	Sandra
♥	Swimp
♥	James
♣	Lee
♥	Dave
♣	Bearman
♦	Lisa

0	♣	3
1	♠	2
2	♥	4
3	♦	3

Counts

0	♣	0
1	♠	3
2	♥	5
3	♦	9

Starting Points

0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		

Sorted

Counting Sort

Example:

- Alphabet: {♣, ♠, ♥, ♦}

♠	Lauren
♥	Delbert
♦	Glaser
♣	Edith
♠	JS
♦	Sandra
♥	Swimp
♥	James
♣	Lee
♥	Dave
♣	Bearman
♦	Lisa

→

0	♣	3
1	♠	2
2	♥	4
3	♦	3

Counts

→

0	♣	0
1	♠	4
2	♥	5
3	♦	9

Starting Points

→

0		
1		
2		
3	♠	Lauren
4		
5		
6		
7		
8		
9		
10		
11		

Sorted

Counting Sort

Example:

- Alphabet: {♣, ♠, ♥, ♦}

♠	Lauren
♥	Delbert
♦	Glaser
♣	Edith
♠	JS
♦	Sandra
♥	Swimp
♥	James
♣	Lee
♥	Dave
♣	Bearman
♦	Lisa

→

0	♣	3
1	♠	2
2	♥	4
3	♦	3

Counts

→

0	♣	0
1	♠	4
2	♥	6
3	♦	9

Starting Points

→

0		
1		
2		
3	♠	Lauren
4		
5	♥	Delbert
6		
7		
8		
9		
10		
11		

Sorted

Counting Sort

Example:

- Alphabet: {♣, ♠, ♥, ♦}

♠	Lauren
♥	Delbert
♦	Glaser
♣	Edith
♠	JS
♦	Sandra
♥	Swimp
♥	James
♣	Lee
♥	Dave
♣	Bearman
♦	Lisa

→

0	♣	3
1	♠	2
2	♥	4
3	♦	3

Counts

→

0	♣	0
1	♠	4
2	♥	6
3	♦	10

Starting Points

→

0		
1		
2		
3	♠	Lauren
4		
5	♥	Delbert
6		
7		
8		
9	♦	Glaser
10		
11		

Sorted

Counting Sort

Example:

- Alphabet: {♣, ♠, ♥, ♦}

♠	Lauren
♥	Delbert
♦	Glaser
♣	Edith
♠	JS
♦	Sandra
♥	Swimp
♥	James
♣	Lee
♥	Dave
♣	Bearman
♦	Lisa

→

0	♣	3
1	♠	2
2	♥	4
3	♦	3

Counts

→

0	♣	1
1	♠	4
2	♥	6
3	♦	10

Starting Points

→

0	♣	Edith
1		
2		
3	♠	Lauren
4		
5	♥	Delbert
6		
7		
8		
9	♦	Glaser
10		
11		

Sorted

Counting Sort

Example:

- Alphabet: {♣, ♠, ♥, ♦}

♠	Lauren
♥	Delbert
♦	Glaser
♣	Edith
♠	JS
♦	Sandra
♥	Swimp
♥	James
♣	Lee
♥	Dave
♣	Bearman
♦	Lisa

→

0	♣	3
1	♠	2
2	♥	4
3	♦	3

Counts

→

0	♣	1
1	♠	5
2	♥	6
3	♦	10

Starting Points

→

0	♣	Edith
1		
2		
3	♠	Lauren
4	♠	JS
5	♥	Delbert
6		
7		
8		
9	♦	Glaser
10		
11		

Sorted

Counting Sort

Example:

- Alphabet: {♣, ♠, ♥, ♦}

♠	Lauren
♥	Delbert
♦	Glaser
♣	Edith
♠	JS
♦	Sandra
♥	Swimp
♥	James
♣	Lee
♥	Dave
♣	Bearman
♦	Lisa

→

0	♣	3
1	♠	2
2	♥	4
3	♦	3

Counts

→

0	♣	1
1	♠	5
2	♥	6
3	♦	11

Starting Points

→

0	♣	Edith
1		
2		
3	♠	Lauren
4	♠	JS
5	♥	Delbert
6		
7		
8		
9	♦	Glaser
10	♦	Sandra
11		

Sorted

Counting Sort

Example:

- Alphabet: {♣, ♠, ♥, ♦}

♠	Lauren
♥	Delbert
♦	Glaser
♣	Edith
♠	JS
♦	Sandra
♥	Swimp
♥	James
♣	Lee
♥	Dave
♣	Bearman
♦	Lisa

→

0	♣	3
1	♠	2
2	♥	4
3	♦	3

Counts

→

0	♣	1
1	♠	5
2	♥	7
3	♦	11

Starting Points

→

0	♣	Edith
1		
2		
3	♠	Lauren
4	♠	JS
5	♥	Delbert
6	♥	Swimp
7		
8		
9	♦	Glaser
10	♦	Sandra
11		

Sorted

Counting Sort

Example:

- Alphabet: {♣, ♠, ♥, ♦}

♠	Lauren
♥	Delbert
♦	Glaser
♣	Edith
♠	JS
♦	Sandra
♥	Swimp
♥	James
♣	Lee
♥	Dave
♣	Bearman
♦	Lisa

→

0	♣	3
1	♠	2
2	♥	4
3	♦	3

Counts

→

0	♣	1
1	♠	5
2	♥	8
3	♦	11

Starting Points

→

0	♣	Edith
1		
2		
3	♠	Lauren
4	♠	JS
5	♥	Delbert
6	♥	Swimp
7	♥	James
8		
9	♦	Glaser
10	♦	Sandra
11		

Sorted

Counting Sort

Example:

- Alphabet: {♣, ♠, ♥, ♦}

♠	Lauren
♥	Delbert
♦	Glaser
♣	Edith
♠	JS
♦	Sandra
♥	Swimp
♥	James
♣	Lee
♥	Dave
♣	Bearman
♦	Lisa

→

0	♣	3
1	♠	2
2	♥	4
3	♦	3

Counts

→

0	♣	2
1	♠	5
2	♥	8
3	♦	11

Starting Points

→

0	♣	Edith
1	♣	Lee
2		
3	♠	Lauren
4	♠	JS
5	♥	Delbert
6	♥	Swimp
7	♥	James
8		
9	♦	Glaser
10	♦	Sandra
11		

Sorted

Counting Sort

Example:

- Alphabet: {♣, ♠, ♥, ♦}

♠	Lauren
♥	Delbert
♦	Glaser
♣	Edith
♠	JS
♦	Sandra
♥	Swimp
♥	James
♣	Lee
♥	Dave
♣	Bearman
♦	Lisa

→

0	♣	3
1	♠	2
2	♥	4
3	♦	3

Counts

→

0	♣	2
1	♠	5
2	♥	9
3	♦	11

Starting Points

→

0	♣	Edith
1	♣	Lee
2		
3	♠	Lauren
4	♠	JS
5	♥	Delbert
6	♥	Swimp
7	♥	James
8	♥	Dave
9	♦	Glaser
10	♦	Sandra
11		

Sorted

Counting Sort

Example:

- Alphabet: {♣, ♠, ♥, ♦}

♠	Lauren
♥	Delbert
♦	Glaser
♣	Edith
♠	JS
♦	Sandra
♥	Swimp
♥	James
♣	Lee
♥	Dave
♣	Bearman
♦	Lisa

→

0	♣	3
1	♠	2
2	♥	4
3	♦	3

Counts

→

0	♣	3
1	♠	5
2	♥	9
3	♦	11

Starting Points

→

0	♣	Edith
1	♣	Lee
2	♣	Bearman
3	♠	Lauren
4	♠	JS
5	♥	Delbert
6	♥	Swimp
7	♥	James
8	♥	Dave
9	♦	Glaser
10	♦	Sandra
11		

Sorted

Counting Sort

Example:

- Alphabet: {♣, ♠, ♥, ♦}

♠	Lauren
♥	Delbert
♦	Glaser
♣	Edith
♠	JS
♦	Sandra
♥	Swimp
♥	James
♣	Lee
♥	Dave
♣	Bearman
♦	Lisa

→

0	♣	3
1	♠	2
2	♥	4
3	♦	3

Counts

→

0	♣	3
1	♠	5
2	♥	9
3	♦	12

Starting Points

→

0	♣	Edith
1	♣	Lee
2	♣	Bearman
3	♠	Lauren
4	♠	JS
5	♥	Delbert
6	♥	Swimp
7	♥	James
8	♥	Dave
9	♦	Glaser
10	♦	Sandra
11	♦	Lisa

Sorted

Counting Sort Runtime

Lecture 36, CS61B, Spring 2025

Counting Sort vs. Quicksort

For sorting an array of the 100 largest cities by population, which sort do you think has a better expected worst case runtime in seconds?

A. Counting Sort (as described in our demo)
B. Quicksort

First question to ask yourself: What is the alphabet for counting sort here?

Counting Sort vs. Quicksort

For sorting an array of the 100 largest cities by population, which sort do you think has a better expected worst case runtime in seconds?

A. Counting Sort (as described in our demo)
B. Quicksort

Counting sort requires building an array of size 37,832,892 (population of Tokyo).

6352254	Ahmedabad
4778000	Alexandria
5346518	Ankara
6555956	Atlanta
8495928	Bandung
12517749	Bangalore
...	...

→

...	...
4777999	0
4778000	1
4778001	0
4778002	0
...	...
37832892	1

Counts

→ ...

Counting Sort Runtime Analysis

What is the runtime for counting sort on N keys with alphabet of size R?

- Treat R as a variable, not a constant.

Counting Sort Runtime Analysis

Total runtime on N keys with alphabet of size R: Θ(N+R)

- Creating and filling our count-related arrays: Θ(R)
  - Example: R = 4 for four card suits.
- Counting each item and copying into new array: Θ(N)

For ordered array.

For counts and starting points.

Memory usage: Θ(N+R)

Empirical experiments needed to compare vs. Quicksort on practical inputs.

Bottom line: If N is ≥ R, then we expect reasonable performance.

See hidden slide after this for a more verbose explanation.

Counting Sort Runtime Analysis (More Verbose)

Total runtime on N keys with alphabet of size R: Θ(N+R)

- Create an array of size R to store counts: Θ(R)
- Counting number of each item: Θ(N)
- Calculating target positions of each item: Θ(R)
- Creating an array of size N to store ordered data: Θ(N)
- Copying items from original array to ordered array: Do N times:
  - Check target position: Θ(1)
  - Update target position: Θ(1)
- Copying items from ordered array back to original array: Θ(N)

For ordered array. For counts and starting points.

Memory usage: Θ(N+R)

Empirical experiments needed to compare vs. Quicksort on practical inputs.

Bottom line: If N is ≥ R, then we expect reasonable performance.

Counting Sort vs. Quicksort

Give an example of a specific situation where Counting Sort will be clearly faster than Quicksort.

A. Counting Sort: $\Theta(N+R)$
B. Quicksort: $\Theta(N \log N)$

Previous example was sorting N = 100 cities by population (R = 37,832,892).

Sort Summary

	Memory	Runtime	Notes	Stable?
Heapsort	$\Theta(1)$	$\Theta(N \log N)$	Bad caching (61C)	No
Insertion	$\Theta(1)$	$\Theta(N^2)$	Small N, almost sorted	Yes
Mergesort	$\Theta(N)$	$\Theta(N \log N)$	Fastest stable	Yes
Random Quicksort	$\Theta(\log N)$	$\Theta(N \log N)$ expected	Fastest compare sort	No
Counting Sort	$\Theta(N+R)$	$\Theta(N+R)$	Alphabet keys only	Yes

N: Number of keys. R: Size of alphabet.

Counting sort is nice, but alphabetic restriction limits usefulness.

- Idea: Let's try digit-by-digit sorting.
- The set of possible digits will be a relatively small alphabet.

LSD Radix Sort

Lecture 36, CS61B, Spring 2025

Sorting Stability

Warmup: Digit-by-digit Sorting

Counting Sort

- Procedure
- Runtime

Radix Sorts

- **LSD Radix Sort**
- MSD Radix Sort

Counting sort is slow when the alphabet is large.

- By decomposing input into a string of characters from a finite alphabet, we can force R to be small.

horse	Lauren
elf	Delbert
cat	Glaser
crab	Edith
monkey	JS
rhino	Sandra
raccoon	Swimp
cat	James
fish	Lee
tree	Dave
virus	Bearman
human	Lisa

♠♠	Lauren
♥♦	Delbert
♦♣	Glaser
♣♥	Edith
♠♥	JS
♦♣	Sandra
♥♠	Swimp

4238	Lauren
34163	Delbert
123	Glaser
43415	Edith
9918	JS
767	Sandra
3	Swimp
634	James
724	Lee
2346	Dave
457	Bearman
312	Lisa

Digit-by-digit Counting Sort

As we've seen, we can sort each digit independently from rightmost digit towards left.

- Example: Over {♣, ♠, ♥, ♦}

♠♠	Lauren
♥♦	Delbert
♦♣	Glaser
♣♥	Edith
♠♥	JS
♦♣	Sandra
♥	Swimp

→

♦♣	Glaser
♦♣	Sandra
♥♣	Dave
♠♠	Lauren
♥♠	Swimp
♣♠	Lee
♣	Bearman

♣♠	Lee
♣♠	Bearman
♣♥	Edith
♠♠	Lauren
♠♥	JS
♥♣	Dave
♥	Swimp

Digit-by-digit Counting Sort

Sort each digit independently from rightmost digit towards left.

- Example: Over {1, 2, 3, 4}

22	Lauren
34	Delbert
41	Glaser
13	Edith
23	JS
41	Sandra
32	Swimp
34	James
12	Lee
31	Dave
12	Bearman
42	Lisa

41	Glaser
41	Sandra
31	Dave
22	Lauren
32	Swimp
12	Lee
12	Bearman
42	Lisa
13	Edith
23	JS
34	Delbert
34	James

12	Lee
12	Bearman
13	Edith
22	Lauren
23	JS
31	Dave
32	Swimp
34	Delbert
34	James
41	Glaser
41	Sandra
42	Lisa

LSD Radix Sort

Non-comparison based sorting algorithms that proceed digit-by-digit are called "Radix Sorts".

Via wikipedia: "In a positional numeral system, the **radix** or base is the number of unique digits, including the digit zero, used to represent numbers."

The sort we've just discussed is called "LSD Radix Sort".

- LSD: Least Significant Digit.

LSD Runtime

What is the runtime of LSD sort?

- Pick appropriate letters to represent non-constant terms.

22	Lauren
34	Delbert
41	Glaser
13	Edith
23	JS
41	Sandra
32	Swimp
34	James
12	Lee
31	Dave
12	Bearman
42	Lisa

41	Glaser
41	Sandra
31	Dave
22	Lauren
32	Swimp
12	Lee
12	Bearman
42	Lisa
13	Edith
23	JS
34	Delbert
34	James

12	Lee
12	Bearman
13	Edith
22	Lauren
23	JS
31	Dave
32	Swimp
34	Delbert
34	James
41	Glaser
41	Sandra
42	Lisa

LSD Runtime

What is the runtime of LSD sort?

- Θ(WN+WR)
- N: Number of items, R: size of alphabet, W: Width of each item in # digits

22	Lauren
34	Delbert
41	Glaser
13	Edith
23	JS
41	Sandra
32	Swimp
34	James
12	Lee
31	Dave
12	Bearman
42	Lisa

41	Glaser
41	Sandra
31	Dave
22	Lauren
32	Swimp
12	Lee
12	Bearman
42	Lisa
13	Edith
23	JS
34	Delbert
34	James

12	Lee
12	Bearman
13	Edith
22	Lauren
23	JS
31	Dave
32	Swimp
34	Delbert
34	James
41	Glaser
41	Sandra
42	Lisa

Non-equal Key Lengths

After processing least significant digit, we have array shown below. Now what?

4**3**	51
9	71
81**7**	412
41**2**	43
5**1**	33
3**3**	817
7**1**	9

Non-equal Key Lengths

When keys are of different lengths, can treat empty spaces as less than all other characters.

·4**3**	·**5**1	**·**·9	··9
··**9**	·**7**1	**4**12	·33
81**7**	4**1**2	**8**17	·43
41**2**	·**4**3	**·**33	·51
·5**1**	·**3**3	**·**43	·71
·3**3**	8**1**7	**·**51	412
·7**1**	·**·**9	**·**71	817

Sorting Summary

W passes of counting sort: Θ(WN+WR) runtime.

- Annoying feature: Runtime depends on length of longest key.

	Memory	Runtime	Notes	Stable?
Heapsort	Θ(1)	Θ(N log N)*	Bad caching (61C)	No
Insertion	Θ(1)	Θ(N^2)*	Small N, almost sorted	Yes
Mergesort	Θ(N)	Θ(N log N)*	Fastest stable sort	Yes
Random Quicksort	Θ(log N)	Θ(N log N)* expected	Fastest compare sort	No
Counting Sort	Θ(N+R)	Θ(N+R)	Alphabet keys only	Yes
LSD Sort	Θ(N+R)	Θ(WN+WR)	Strings of alphabetical keys only	Yes

N: Number of keys. R: Size of alphabet. W: Width of longest key.

*: Assumes constant compareTo time.

MSD Radix Sort

Lecture 36, CS61B, Spring 2025

Sorting Stability

Warmup: Digit-by-digit Sorting

Counting Sort

- Procedure
- Runtime

Radix Sorts

- LSD Radix Sort
- **MSD Radix Sort**

MSD (Most Significant Digit) Radix Sort

Basic idea: Just like LSD, but sort from leftmost digit towards the right.

Pseudopseudohypoparathyroidism
Floccinaucinihilipilification
Antidisestablishmentarianism
Honorificabilitudinitatibus
Pneumonoultramicroscopicsilicovolcanoconiosis

MSD Sort Question

Suppose we sort by topmost digit, then middle digit, then rightmost digit. Will we arrive at the correct result?   A. Yes,   B. No

a	d	d
c	a	b
f	a	d
f	e	e
b	a	d
b	e	e
f	e	d
b	e	d
a	c	e

a	d	d
a	c	e
b	a	d
b	e	e
b	e	d
c	a	b
f	a	d
f	e	e
f	e	d

MSD Sort Question

Suppose we sort by topmost digit, then middle digit, then rightmost digit. Will we arrive at the correct result?   A. Yes,   **B. No.** How do we fix?

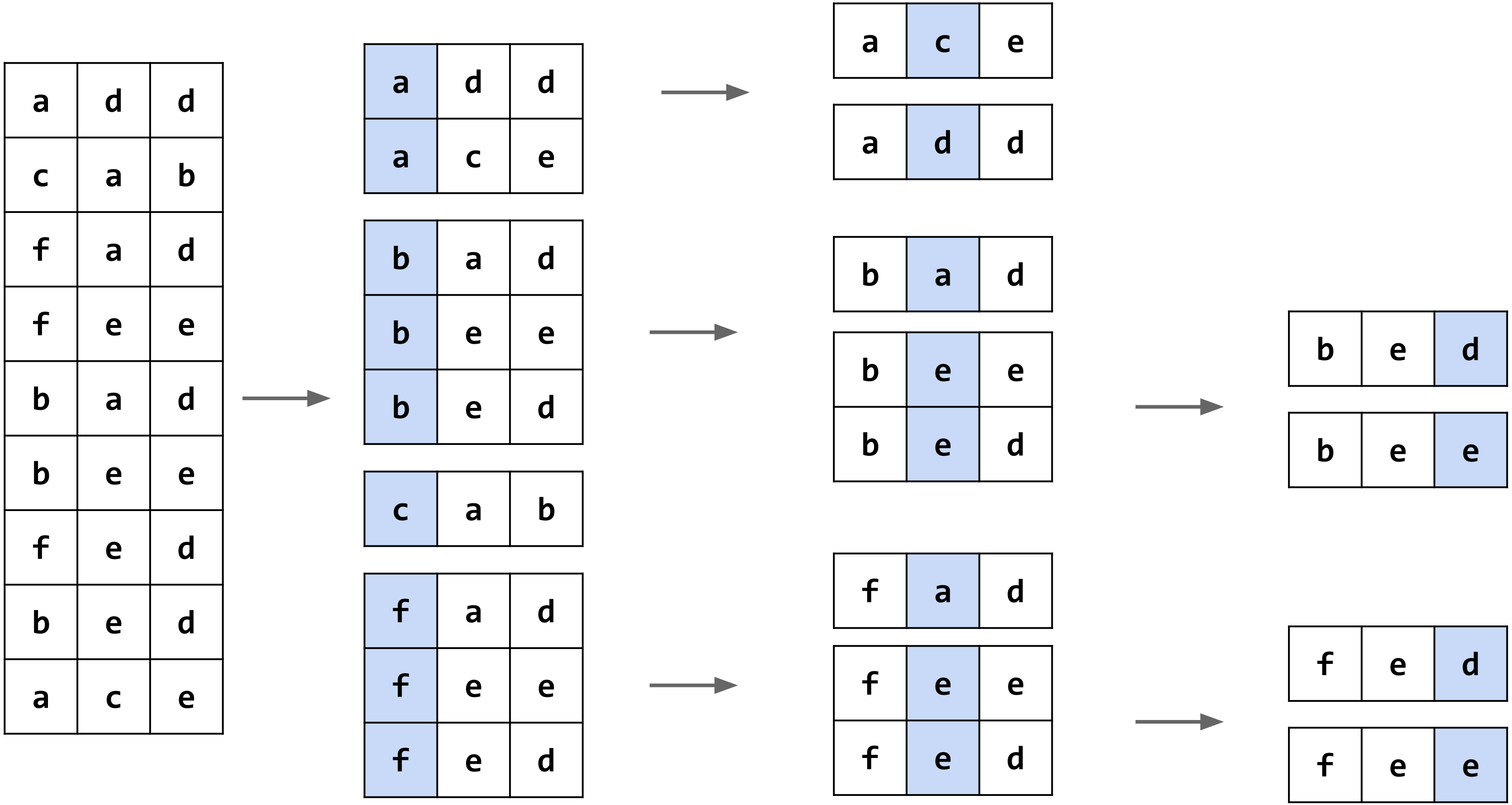
a	d	d
c	a	b
f	a	d
f	e	e
b	a	d
b	e	e
f	e	d
b	e	d
a	c	e

a	d	d
a	c	e
b	a	d
b	e	e
b	e	d
c	a	b
f	a	d
f	e	e
f	e	d

b	a	d
a	d	d

MSD Radix Sort (correct edition)

Key idea: Sort each subproblem separately.



Runtime of MSD

What is the Best Case of MSD sort (in terms of N, W, R)?

What is the Worst Case of MSD sort (in terms of N, W, R)?

Runtime of MSD

Best Case.

- We finish in one counting sort pass, looking only at the top digit: $\Theta(N + R)$

Worst Case.

- We have to look at every character, degenerating to LSD sort: $\Theta(WN + WR)$

Sorting Runtime Analysis

	Memory	Runtime (worst)	Notes	Stable?
Heapsort	$\Theta(1)$	$\Theta(N \log N)$*	Bad caching (61C)	No
Insertion	$\Theta(1)$	$\Theta(N^2)$*	Fastest for small N, almost sorted data	Yes
Mergesort	$\Theta(N)$	$\Theta(N \log N)$*	Fastest stable sort	Yes
Random Quicksort	$\Theta(\log N)$	$\Theta(N \log N)$* expected	Fastest compare sort	No
Counting Sort	$\Theta(N+R)$	$\Theta(N+R)$	Alphabet keys only	Yes
LSD Sort	$\Theta(N+R)$	$\Theta(WN+WR)$	Strings of alphabetical keys only	Yes
MSD Sort	$\Theta(N+WR)$	$\Theta(N+R)$ (best) $\Theta(WN+WR)$ (worst)	Bad caching (61C)	Yes

N: Number of keys. R: Size of alphabet. W: Width of longest key.

*: Assumes constant compareTo time.

Sounds of Sorting Algorithms

Starts with selection sort: https://www.youtube.com/watch?v=kPRA0W1kECg

Insertion sort: https://www.youtube.com/watch?v=kPRA0W1kECg&t=0m9s

Quicksort: https://www.youtube.com/watch?v=kPRA0W1kECg&t=0m38s

Mergesort: https://www.youtube.com/watch?v=kPRA0W1kECg&t=1m05s

Heapsort: https://www.youtube.com/watch?v=kPRA0W1kECg&t=1m28s

LSD sort: https://www.youtube.com/watch?v=kPRA0W1kECg&t=1m54s

MSD sort: https://www.youtube.com/watch?v=kPRA0W1kECg&t=2m10s

Shell's sort: https://www.youtube.com/watch?v=kPRA0W1kECg&t=3m37s

Questions to ponder (later… after class):

- How many items are sorted in the video for selection sort?
- Why does insertion sort take longer / more compares than selection sort?
- At what time stamp does the first partition complete for Quicksort?
- Could the size of the input used by mergesort in the video be a power of 2?
- What do the colors mean for heapsort?
- How many characters are in the alphabet used for the LSD sort problem?
- How many digits are in the keys used for the LSD sort problem?