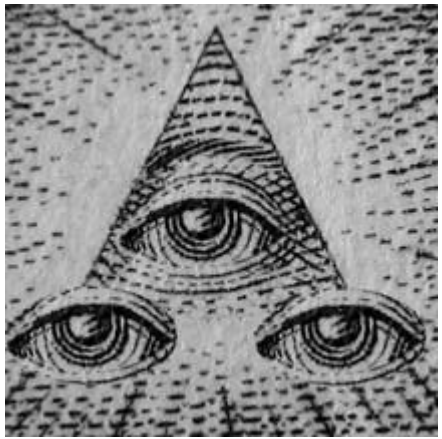


CS61B, Guest Lecturer: Chris Gregg

Lecture 35: Radix Sort

- Counting Sort
- LSD Radix Sort
- LSD Radix Sort vs. Comparison Sorting
- MSD Radix Sort

Comparison Based Sorting

One key idea from lecture last time, sorting requires Ω(N log N) compares:

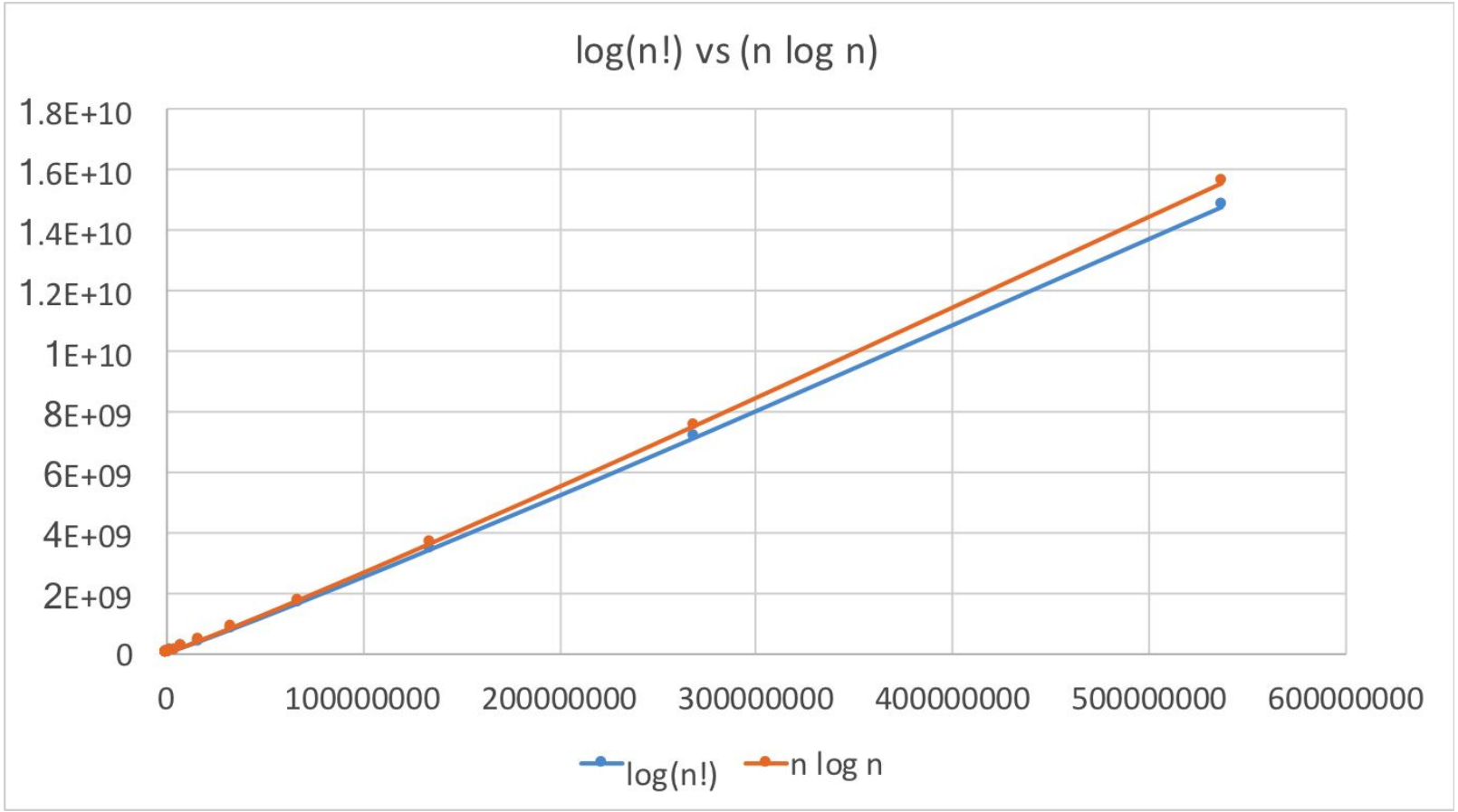
- Sorting provides a solution to the puppy-cat-dog problem.
- Thus, any lower bound on the number of compares for puppy-cat-dog also applies to sorting (assuming it also uses compares for all decisions).
- For N items, there are N! possible answers to the puppy-cat-dog problem.
  - Example: 3! = 6, i.e. 6 different places for puppy, cat, or dog.
- If you are asking yes/no questions, given Q possible answers, you'll need to ask at least lg(Q) questions in the worst case to find the correct answer.
  - Thus, we'll need lg(N!) questions to solve puppy, cat, dog.
  - Since lg(N!) ∈ Θ(N log N), that means the best possible algorithm for solving puppy, cat, dog completes in Ω(N log N) questions.
  - Therefore, any algorithm for sorting that asks yes/no questions (e.g. comparisons) must ask Ω(N log N) questions.

Comparison Based Sorting

- ...Since $\lg(N!) \in \Theta(N \log N)$ …
- Is this true?
- We could prove this mathematically (or at least convince ourselves if we don't have a rigorous proof), but let's look at an empirical method that would at least give us a good idea that this is the case.
- Off to Excel!
  - We have to be a bit sneaky to calculate lg(N!) -- N! grows really fast!
    - Let's use a loop

```
public static final double log2 = Math.log(2.0);
public static double logNFactorial(int n) {
  double result = 0;
  for (int i=n; i >= 1; i--) {
    result += Math.log(i) / log2;
  }
  return result;
}
```

Comparison Based Sorting



Avoiding The Need to Compare

We showed that sorting requires Ω(N log N) compares, and thus the Ultimate Comparison Based Sorting Algorithm has a worst case runtime of Θ(N log N).

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

Important note before we begin our quest:

- Better asymptotic performance does not imply better real world performance, i.e. even if we discover a Θ(N) sort, it might not be faster than Mergesort/Quicksort when used on real inputs.
- Why not?
  - We don't always compute for infinitely large N.
  - For practical N, log N is going to be pretty small.

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

- I need some volunteers...

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 (but we had limitations!)

- 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

Questic the first ♥?

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

Counting Sort

Alphabet case: Keys belong to a finite ordered alphabet.

- Example: {♣, ♠, ♥, ♦}

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

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

Questic 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.
- Controllable demo (from Josh): Demo
- Alternate animated demo (by Paul Hilfinger): http://inst.eecs.berkeley.edu/~cs61b/fa15/demos/radix-sort-demo.html

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

Counting Sort vs. Quicksort:

For sorting an int[] array of modest length (say N < 1000), what sort do you think has a better expected worst case runtime in seconds?

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

Counting Sort vs. Quicksort

For sorting an int[] array of modest length (say N < 1000), what sort do you think has a better expected worst case runtime in seconds?

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

Example of an array where Quicksort wins easily: [5 1,000 2 9,001 1,000,000]

What is our alphabet size?

- 1,000,000, therefore our algorithm consists of:
  - Create an array of length 1,000,000 (initializes all values to zero).
  - Put at 1 in 5 of the locations.
  - Create a “starting positions” array of length 1,000,000.
  - Copy things over to a new array of size 5.

Counting Sort Runtime Analysis

Total runtime on N keys with alphabet of size R: $\Theta(N+R)$

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

For ordered array.

For counts and starting points.

Memory usage: $\Theta(N+R)$

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

Bottom line: If N is $\geq$ R, then we expect reasonable performance.

Counting Sort vs. Quicksort:

For sorting really really really big collections of items from some alphabet, which algorithm will be fastest?

A. Counting Sort
B. Quicksort

Counting Sort vs. Quicksort

For sorting really really really big collections of items from some alphabet, which algorithm will be fastest?

A. **Counting Sort: Θ(N+R)**
B. Quicksort: Θ(N log N)

For sufficiently large collections, counting sort will simply be faster.

Sort Summary

	Memory	Runtime	Notes	Stable?
Heapsort	Θ(1)	Θ(N log N)	Bad caching (61C)	No
Insertion	Θ(1)	$\Theta(N^2)$	Small N, almost sorted	Yes
Mergesort	Θ(N)	Θ(N log N)	Fastest stable	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

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

Counting sort is nice, but alphabetic restriction limits usefulness.

- No obvious way to sort hard-to-count things like Strings.

LSD Radix Sort


LSD Radix Sort

LSD Radix Sort

LSD = *L*east *S*ignificant *D*igit

Example: 54826

Most significant **Least significant**

Radix Sort

Not all keys belong to finite alphabets, e.g. Strings.

- However, Strings consist of characters from a finite alphabet.

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
♥♦	James
♣♠	Lee
♥♣	Dave
♣♠	Bearman
♦♠	Lisa

42387	Lauren
34163	Delbert
123	Glaser
43415	Edith
9918	JS
767	Sandra
3	Swimp
634	James
724	Lee
2346	Dave
457	Bearman
312	Lisa

LSD (Least Significant Digit) Radix Sort -- Using Counting Sort

Sort each digit independently from rightmost digit towards left.

- Example: Over {♣, ♠, ♥, ♦}

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

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

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

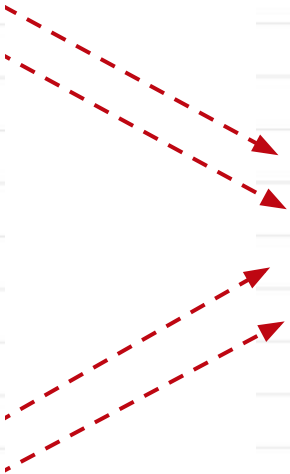
LSD (Least Significant Digit) Radix Sort -- Using Counting Sort

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

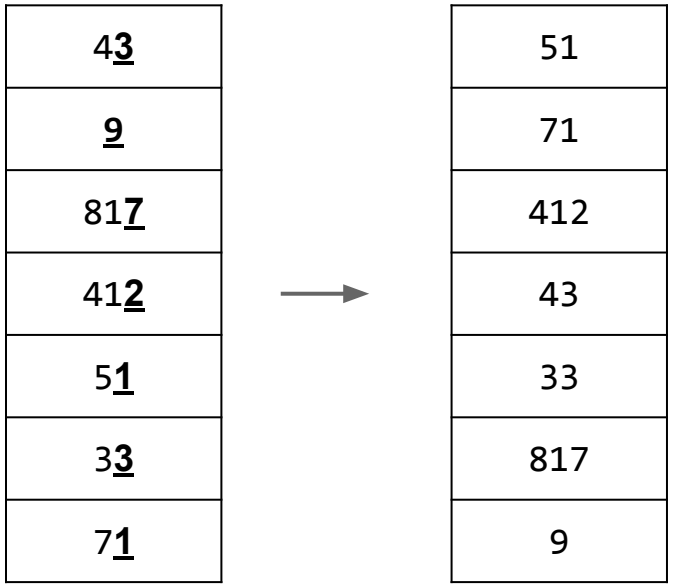
41	Glaser
41	Sandra
31	Dave
32	Swimp
22	Lauren
12	Lee
12	Bearman
42	Lisa
23	JS
13	Edith
34	James
34	Delbert



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

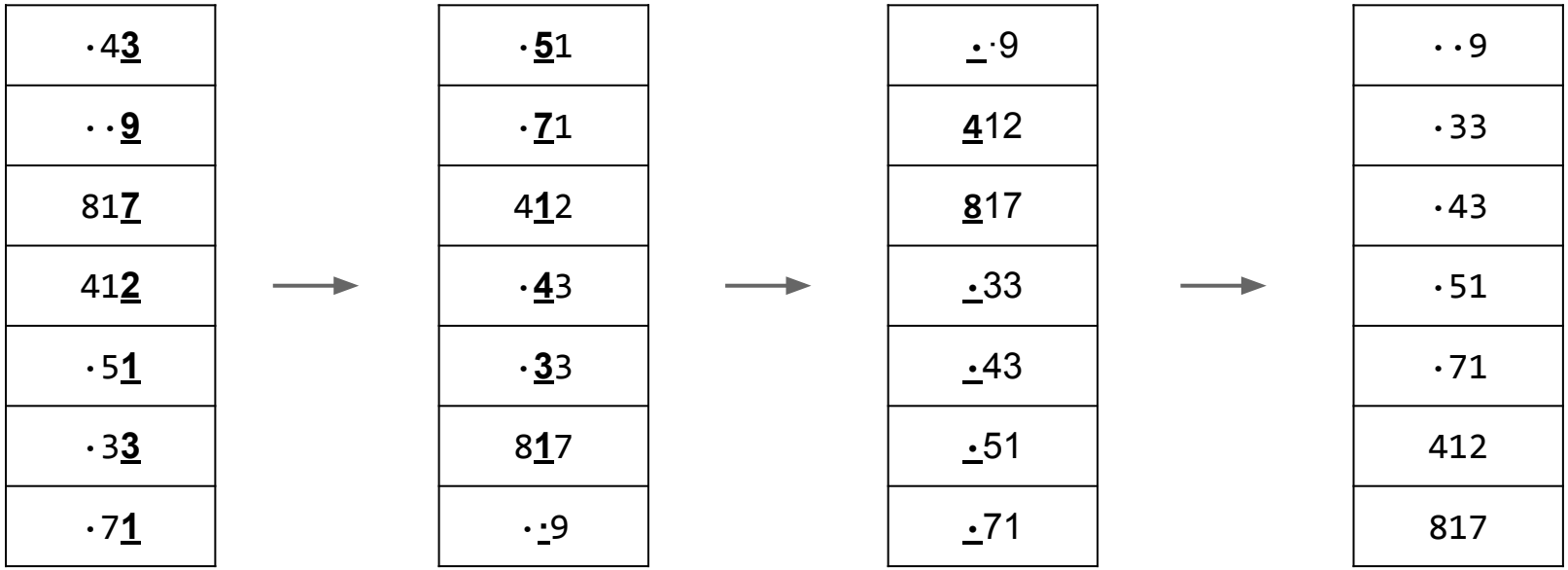
Non-equal Key Lengths

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



Non-equal Key Lengths

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



LSD Runtime Analysis

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)	$\Theta(N^2)$*	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.

LSD Radix Sort
vs.
Comparison Sorting

LSD Runtime Analysis

The facts:

- LSD sort completes with a runtime of $\Theta(WN+WR)$.
- Merge Sort requires $\Theta(N \log N)$ compares.

Which is better?

- Depends on the input!

Example 1: Long Dissimilar Strings:

The facts:

- LSD sort completes with a runtime of Θ(WN+WR).
- Merge Sort requires Θ(N log N) compares.

For sorting a large number of very long strings that are highly dissimilar, which algorithm will be faster?

A. LSD is faster.
B. Merge Sort is faster.

IUYQWLKJASHDIUHQWLEIUHAD...
LIUHLIUHRGLIUEHLQIUHREWEF...
OZIUHIOHLHLZIUHIUHFWEIUHF...
...

Example 1: Long Dissimilar Strings

The facts:

- LSD sort completes with a runtime of Θ(WN+WR).
- Merge Sort requires Θ(N log N) compares.

Sorting a large number of very long strings that are highly dissimilar.

A. LSD is faster.

B. Merge Sort is faster.

- Most characters never even looked at.
- W dominates log N.

IUYQWLKJASHDIUHQWLEIUHAD...
LIUHLIUHRGLIUEHLQIUHREWEF...
OZIUHIOHLHLZIUHIUHFWEIUHF...
...

Example 2: Long Similar Strings:

The facts:

- LSD sort completes with a runtime of Θ(WN+WR).
- Merge Sort requires Θ(N log N) compares.

For sorting a large number of very long strings that are highly similar, particularly in their top characters, which algorithm will be faster?

A. LSD is faster.
B. Merge Sort is faster.

AAAAAA…AAAAAAAAAAAAAAAAAB
AAAAAA…AAAAAAAAAAAAAAAAAD
AAAAAA…AAAAAAAAAAAAAAAAAF
…

Example 2: Long Similar Strings

The facts:

- LSD sort completes with a runtime of Θ(WN+WR).
- Merge Sort requires Θ(N log N) compares.

Sorting a large number of very long strings that are highly similar, particularly in their top characters.

A. LSD is faster.
- Comparison takes W time!
- Overall Mergesort time: Θ(WN log N)

A. Merge Sort is faster.

AAAAAA…AAAAAAAAAAAAAAAAAB
AAAAAA…AAAAAAAAAAAAAAAAAD
AAAAAA…AAAAAAAAAAAAAAAAAF
…

Overall

Extreme cases:

- Highly dissimilar strings: Merge Sort easily wins.
- Highly similar strings: LSD easily wins.
- Real data: Merge Sort will usually win because we only need to examine upper characters of most strings.
  - Runtime for `compareTo` as a function of W and N is beyond scope of this course (more of a CS70 / CS170 topic).

Goal: Get the linear time of LSD, but also get the advantage of looking at only upper characters.

MSD Radix Sort

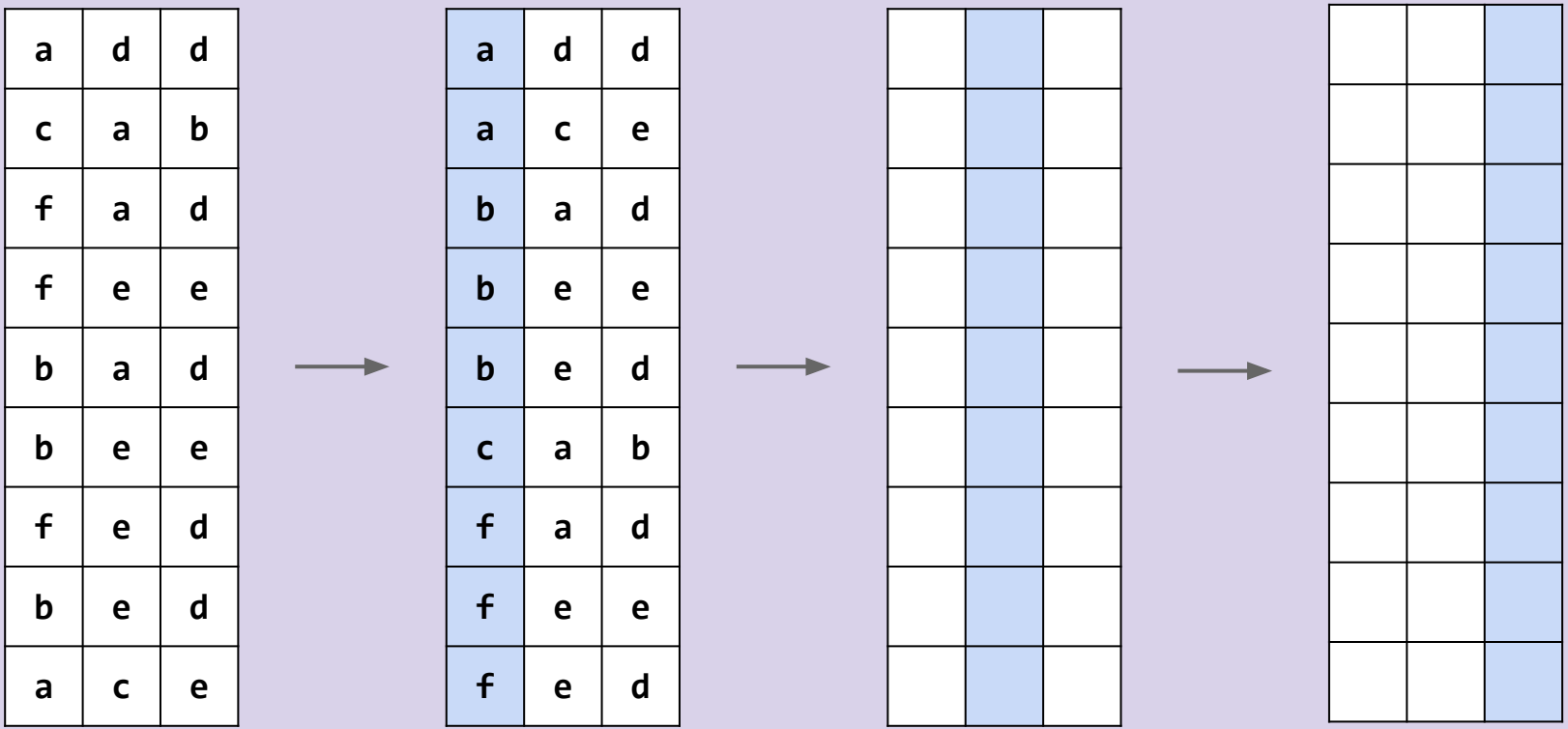
MSD (Most Significant Digit) Radix Sort

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

Pseudopseudohypoparathyroidism
Floccinaucinihilipilification
Antidisestablishmentarianism
Honorificabilitudinitatibus
Pneumonoultramicroscopicsilicovolcanoconiosis

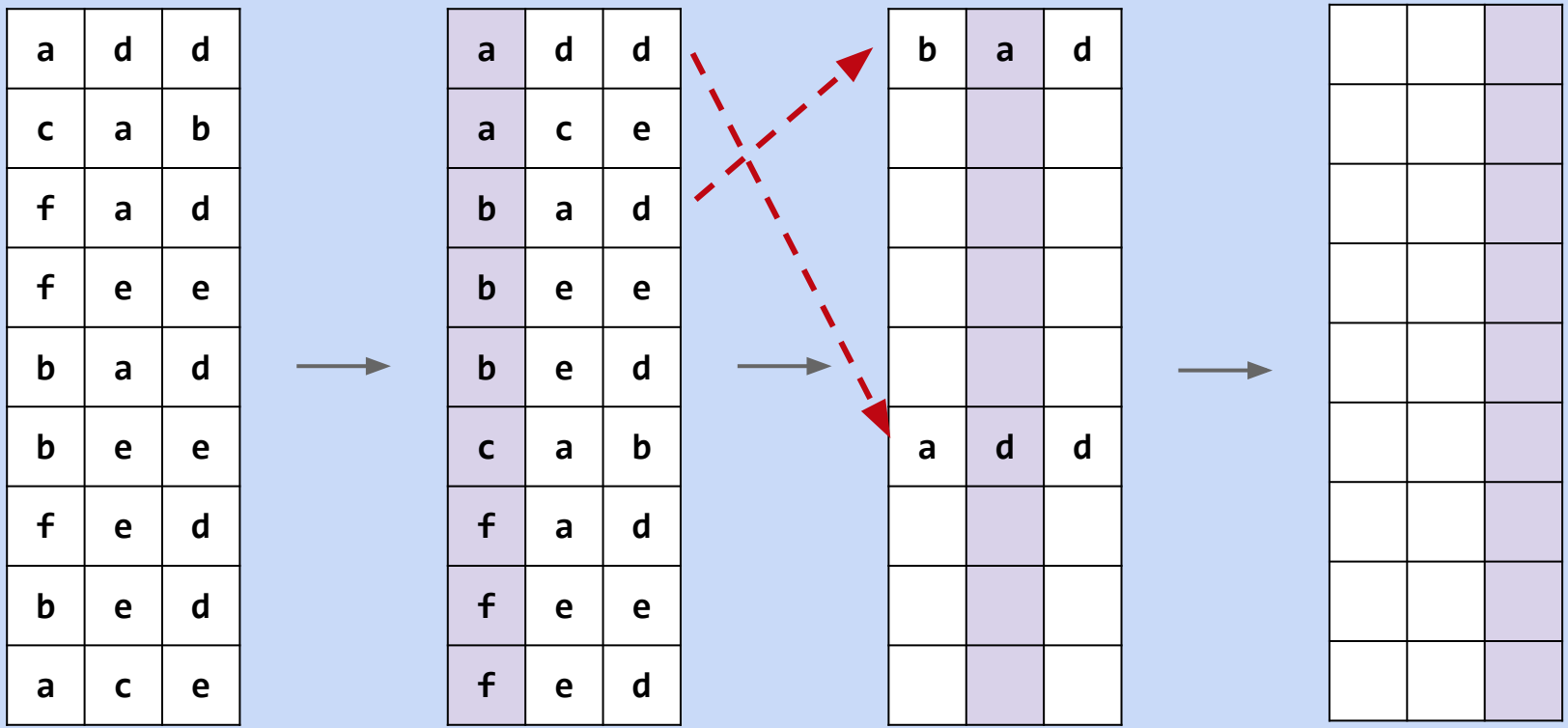
MSD Sort Question:

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



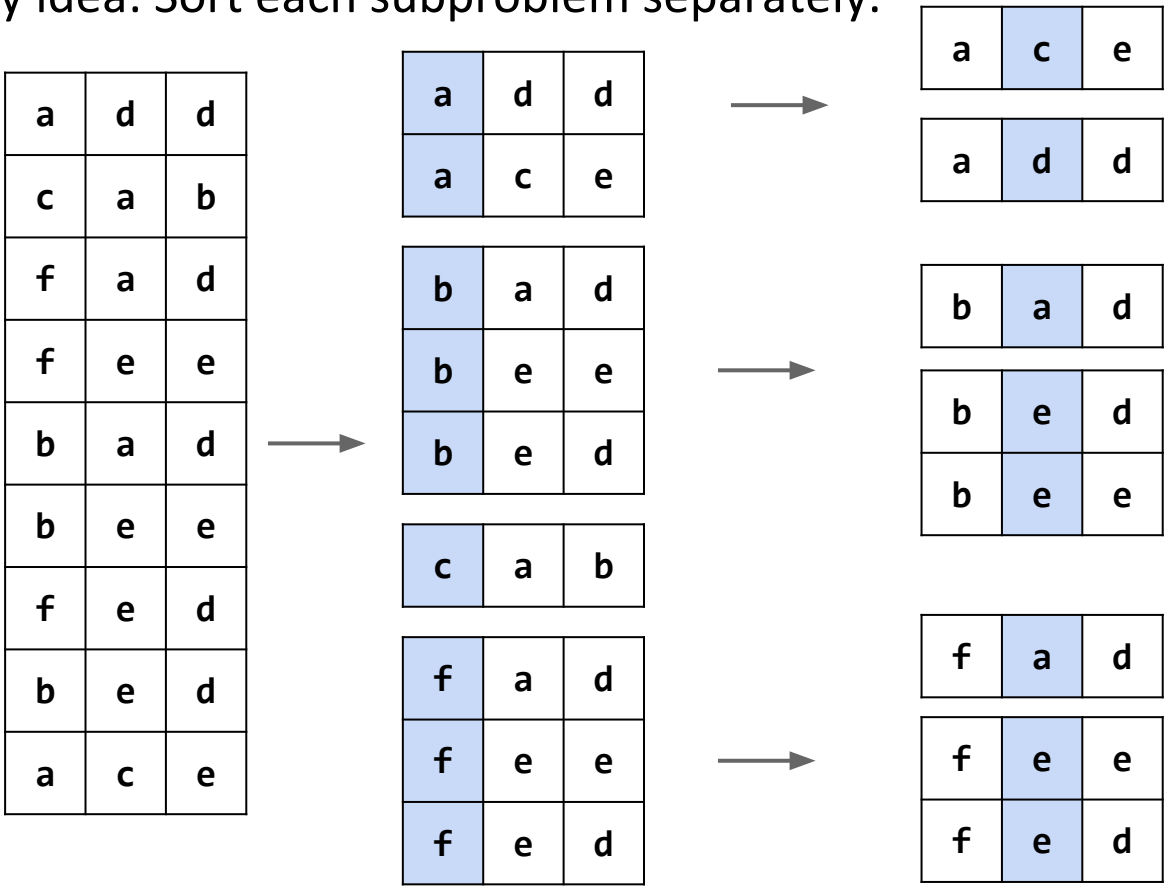
MSD Sort Question

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?



MSD Radix Sort (correct edition)

Key idea: Sort each subproblem separately.



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: $\Theta(WN + WR)$
  - WR because recursion goes W levels deep, and each recursive call creates an array of size R for counting sort.

Sorting Runtime Analysis

	Memory	Runtime (worst)	Notes	Stable?
Heapsort	Θ(1)	Θ(N log N)*	Bad caching (61C)	No
Insertion	Θ(1)	$\Theta(N^2)$*	Fastest for small N, almost sorted data	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
MSD Sort	Θ(N+WR)	Θ(N+R) (best) Θ(WN+WR) (worst)	Bad caching (61C)	Yes

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

*: Assumes constant compareTo time.

Overall

Theoretical performance of radix sorts vs. comparison based sorts is well beyond scope of this course.

- Main idea: Is the log N factor for our comparison sorts more significant than the overhead and width factor for radix sorts?
- In practice: Radix sorts used only in special cases (e.g. sorting integers).

Radix sorts consider each input item as a sequence of characters from a finite alphabet.

- Natural with items like numbers and strings.
- Can be generalized to any input type, but may be quite awkward.

Don’t miss Paul Hilfinger’s interactive sorting demos!

A different LSD Radix Sort implementation: using queues!

The LSD Radix sort we discussed earlier uses counting sort as a start, and uses the counting sort algorithm across all digits / numbers in each element. There is a different version of LSD Radix sort that uses queues to process each element -- a queue is a first-in-first-out data structure, and is perfect for this task.

The algorithm:

1. Determine the length of the longest key (e.g., [1, 100, 27, 42, 754, 6] would have a maximum of 3 because of the 100 and 754).
2. Create R (the size of your alphabet, or base) queues (e.g., 10 for base 10 digits, 0-9)
3. Loop over all the keys from the LSD (right-to-left), populating the queues based on the value of the key digit.
4. Loop through the queues, dequeuing each element back into the original array.

A different LSD Radix Sort implementation: using queues!

The algorithm:

1. Determine the length of the longest key (e.g., [1, 100, 27, 42, 754, 6] would have a maximum of 3 because of the 100 and 754).
2. Create R (the size of your alphabet, or base) queues (e.g., 10 for base 10 digits, 0-9)
3. Loop over all the keys from the LSD (right-to-left), populating the queues based on the value of the key digit.
4. Loop through the queues, dequeuing each element back into the original array.

Animation: https://visualgo.net/bn/sorting

Radix sort on the playground: https://www.youtube.com/watch?v=ibtN8rY7V5k

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?

Citations

Creepy eye thing, title slide: http://photos3.meetupstatic.com/photos/event/a/3/f/4/highres_335141972.jpeg