Lecture 36: Radix Sort:

key idea: Sorting requires $\Omega(N \log N)$

graphing $\log(n!)$ vs $n \log n$, we see a strong correlation.

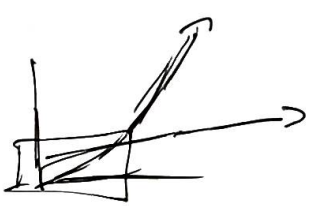
Let's find a $\Theta(N)$ sort!
Remember $\Theta(N)$ might not be faster for certain inputs.

Sleep sort: $N + \max(A)$
↑ A is the largest val in array.

Idea: keep keys for your stored data. Sort them by keys
→ Issue. What if we have key 0000...?

We sorted N items in $\Theta(N)$.
- NO comparisons!
- 0 to N-1 keys.
  - non-unique, consecutive keys.

Counting sort vs Quicksort:
if we have $N < 1000$ items, which is faster?
- at a smaller N, Quicksort wins.
- at a larger N, Counting sort wins.



$\Theta(N+R)$ = Counting sort → total Runtime of N keys w/ alphabet size R.
memory = $\Theta(N+R)$

If $N \geq R$, we expect reasonable performance.

If we have large collections,
Counting sort is faster $\Theta(N+R)$ vs $\Theta(N \log N)$

LSD Radix Sort:

Least significant Digit Sort:

54 8 36

↑ Most sig ↑ least sig.

Not all keys belong to finite alphabets eg Strings.

22
34
41
13

→

4	1
2	2
1	3
3	4

→

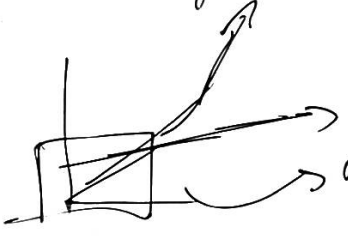
1	3
2	2
3	4
4	1

Now, our "alphabet" is just digits 0–9.

	Memory:	RT:	Stable?
LSD Sort:	$\Theta(N+R)$	$\Theta(WN+WR)$	yes.

N = # of keys R = size of alphabet W = width of longest key.

LSD sort vs Merge Sort!



again, just for small inputs, mergesort is better.

$\sim N < 1000$.

However, if you're sorting very long strings that are highly dissimilar:

- MergeSort is faster. ex: IADSZBJ... / ABCZAYT...

is looks at the letters til it reaches a difference.

But if you have very similar strings:

AAAAA...B
AAAAA...C

→ Merge sort has to go thru every letter! Radix sort better!

*Look at your data when choosing a sort!

MSD: Just like LSD but from left to right.

Best case: First pass sorted: $\Theta(N+WR)$ memory, $\Theta(WR)$ best

$\Theta(WN+WR)$ worst.