6.006 Lecture 11: Sorting IV

		CLRS
□ Radix Sort		8.3
□ A Broader Perspective:	Bucket Sort	8.4
	Quick Sort	Ch 7
	Order Statistics (median)	Ch 9

Radix Sort

- Key idea: view input integers as multidigit numbers in some convenient base (10 for exposition, larger in practice, e.g. base 1024, or base 1,000,000, etc).
- Use counting sort on each digit
- Intuitively, we might sort on Most significant digit first; this is not how Radix sort works; too many invocation of counting sort.

ex:

sort	sort separately	sort, but we are already done here.
329	329	329
457	355	355
657	457	436
839	436	457
436	657	657
720	720	720
355	839	839

- Radix sort does it the other way around

			sorted
329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

- Stability of counting sort is key; would not work otherwise.

Correctness of Radix Sort

- consider two numbers in the input

	6	5	4	3	2	1	0
⋮	2	3	4	5	6	7	8
⋮	2	3	3	5	9	7	8

- The numbers would be put into the correct order when we sort on digit 4; we may have changed their order earlier, but it does not matter.
- When we sort on digits 5 & 6, the numbers retain their (correct) relative positions, because ~~fast~~ counting sort is stable.

Running time of Radix Sort

- For d-digit decimal numbers: $\Theta(\overset{\#\text{ calls to counting sort}}{d}(\overbrace{n+10}^{\text{counting sort}}))$
- For length l ASCII strings $\Theta(l(n+128))$
- In general, we can choose the radix: assume inputs are b-bit binary numbers

1 0 1 1 0 1 0 0 0 1 ← radix 2 range of "digits" is 0-1
← radix-4, range is 0-3
← radix-32, range is 0-31

- ~~Eff~~ Split inputs into $\frac{b}{r}$ r-bit "digits": $\Theta(\frac{b}{r}(n+2^r))$
- The counting sort factor $(n+2^r)$ is asymptotically $O(n)$ for $b \leq \lfloor \lg n \rfloor$

(but grows quickly afterwards; do not choose $r = 2 \lg n$, for example - this gives $2^r = n^2$)

- This is asymptotically optimal.

$\Rightarrow$ inputs $\leq n$: one pass $\frac{b}{r} = 1$ (just counting sort)

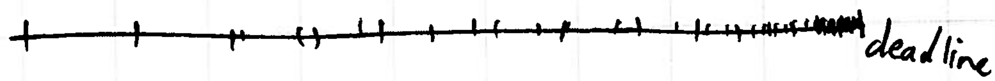
inputs $\leq n^2$: two passes

inputs $\leq n^3$: three passes

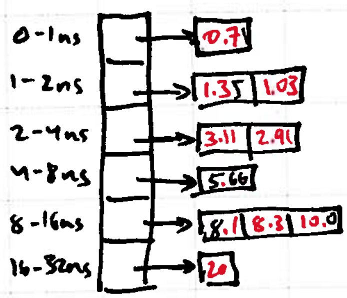
- Another perspective: for $n = 1{,}000{,}000$, two passes sort inputs up to 10^{12}, three up to 10^{18}. pretty good!
- Much more useful than counting sort alone.

Another way to sort in linear time: Bucket Sort

Submission times of PS2A in nanoseconds before deadline:



We can split the range of input keys into buckets, place keys in buckets, sort each bucket (using mergesort or even insertion sort), and concatenate.



If we know the statistical distribution of the input keys and can split them into n buckets such that ~~E[# key in bucket i] = 1~~

$\Pr[\text{key } i \text{ falls in bucket } j] = \frac{1}{n}$

then we can sort using bucket sort in $\Theta(n)$

A Glimpse of Quicksort

- Another divide & conquer algorithm

	mergesort	quicksort
Divide phase $\longrightarrow$	left half - righ half	Choose k, split into $\leq A[k]$, $> A[k]$ (possibly not exactly $\frac{1}{2} \cdot \frac{1}{2}$)
	sort halves recursively	sort ~~but~~ parts recursively
conquer $\longrightarrow$	merge (hard)	concatenate (easy)

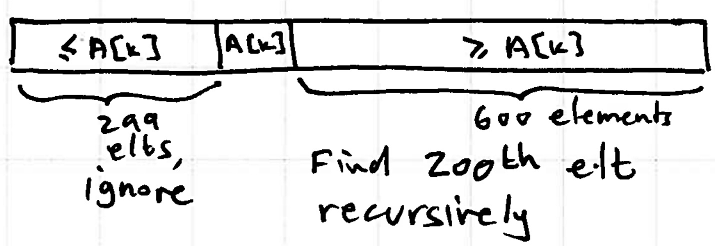
- Splitting can be done in-place (called <u>partition</u>) so concatenate phase does nothing

$\leq A[k]$	$A[k]$	$> A[k]$

- The trick is to choose k so that $\approx \frac{n}{2}$ elts are $\leq A[k]$
- Best techiques are randomized. E.g., choose 3 elts at random and take the middle of them
- Expected running time is $\Theta(n \lg n)$.
  (can do deterministically in $\Theta(n \lg n)$, but more complicated and larger constants).

Order Statistics

- Once we sort a list, we can find max element in $O(n)$, also min, median, 14^{th}, etc.
- We can find the min, max in $\Theta(n)$ time without sorting; can we do the same for the median? (view it as another way to beat the $\Omega(n \lg n)$ comparison-based lower bound, this time by requesting less).
- Yes
- Easy with a randomized algorithm (same structure as quicksort, but at each step continue just with part that contains median



- Can do without randomization, but harder.