CS Lecture: QuickSort

Quicksort is based on partitioning.
- fastest in most situations.

Partitioning:

Choose a pivot, everything left of pivot <= pivot, everything right of pivot => pivot.

*Partitioning has nothing to do with sorting (yet)!

Quicksort / Partition sort:

[5] 3 2 1 7 8 4 6

3 2 1 4 [5] 7 8 6

Observations:
- 5 is in the right place / where it would be if arr was sorted.
- Can sort left & right separately. (Recursive).

Runtime:

Best case: Pivot Always Lands in Middle.

total work at each level

1: N
2: $\frac{N}{2} + \frac{N}{2} = N$
3: $\frac{N}{4} * 4 = N$

$\Theta(N \log N)$

Worst Case: pivot is the beginning of array.

total work = N^2

$\Theta(N^2)$

So why is QuickSort fastest?

Quicksort is really BST sort.

Random data is very rarely worst-case level.

Avoiding Worst Case

performance is based on
- how you select pivot
- how you partition around pivot
- other optimizations

Avoiding Worst Case:

1) Randomness: pick a random pivot / shuffle
2) Smarter pivot selection: choose the median.
3) Introspection: swap to a safer sort if recursion is too deep.
4) Preprocess the array: Analyze array to see if QS will be slow or not.

Randomness

↳ Bad ordering: Array in sorted order

Bad Clumping: Array w/ all duplicates

Sorting so far:

	Memory	Time	Notes
Heapsort	$\Theta(1)$	$\Theta(N \log N)$	Bad Caching
Insertion	$\Theta(1)$	$\Theta(N^2)$	$\Theta(N)$ if almost sorted
Merge	$\Theta(N)$	$\Theta(N \log N)$	
Random QuickSort	$\Theta(\log N)$ ↑ call stack	$\Theta(N \log N)$	Fastest sort.