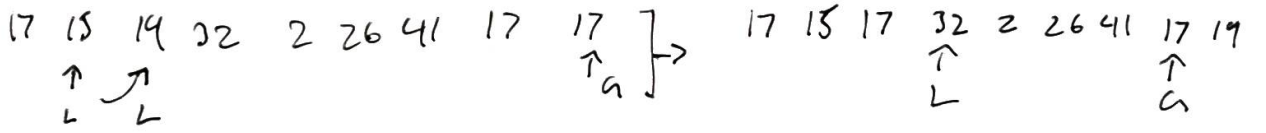
More Quicksort Lecture 34.

Quicksort is the fastest in certain conditions.
How to make Quicksort faster than merge sort?

Tony Hoares In-Place Partitioning:
- Left pointer loves small items. → Relative to pivot.
- Right pointer loves large items. →
- Big idea: Walk towards each other, swap what they don't like.




- Walk pointers towards each other until it meets a hated item.
  - When both pointers stop, swap & move pointers by one.
  - When pointers cross, you're done walking.
- Swap pivot w/ G.
- Repeat for the "less" and "greater" side with the pivot being the first item.

Median:
Cost to compute median is too high Runtime goes too high

QuickSelect
- Partitioning can be used to find the median.

9 550 14 6 10 5 330 817 913

Partitioning

6 5 9 550 14 10 330 817 913

median has to be in middle, so we look to the right of the current pivot!

□ □ □ 550 14 10 330 817 913

□ □ □ 14 10 330 550 817 913

14 10 330 → 10 14 330 → 14 is median

Time Complexity of QuickSelect:

[1, 2, 3 ... N] → $\Theta(N^2)$ Worst case

[1 2 ... ]

[1 2 3 4 ... [N/2] ... N]

Expected RT: $\Theta(N)$

Quick select to find exact median is still pretty slow.

## Stability, Optimizations

A sort is stable if order of equivalent items is preserved.

| | Stable? |
|---|---|
| Heapsort | no |
| Insertion | yes |
| Merge | yes |
| Quicksort (Tony Hoare) | No |

## Additional tricks

- Switch to Insertion sort when size is $\leq 15$
- Make adaptive sorts: exploit existing order in array.
- Exploit restrictions on set of keys. If number of keys is some constant, we can sort in $\Theta(N)$.
- Quick sort: If you go too deep, swap to insertion sort.

## Shuffling

Easiest way:

- Generate N random numbers, attach one to each array item
- Sort the items by the attached random number.