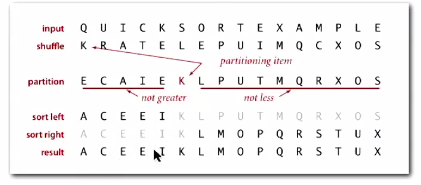
Quicksort

One of the most important algorithms of the 20th century

Basic idea: recursion done *after the work* (as opposed to mergesort, which recurs before the work)

1. Randomly shuffle the array
2. Partition the array so that for *j*
   1. Entry a[j] is in place
   2. No larger entry to the left of *j*
   3. No smaller entry to the right of *j*
3. Sort each piece recursively



How does it work?

Arbitrarily (randomly) choose the first point in the shuffled array to partition around.

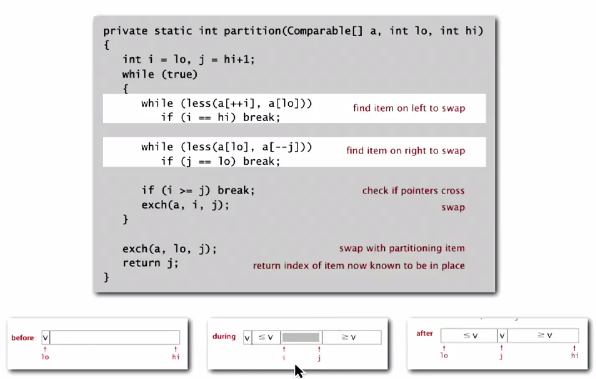
*Until pointers* ***I*** *and* ***J*** cross:

* Scan *I*  from left to right so long as (a[i] < a[lo])
* Scan *J* from right to left so long as (a[j] > a[lo])
* Exchange a[i] with a[j]

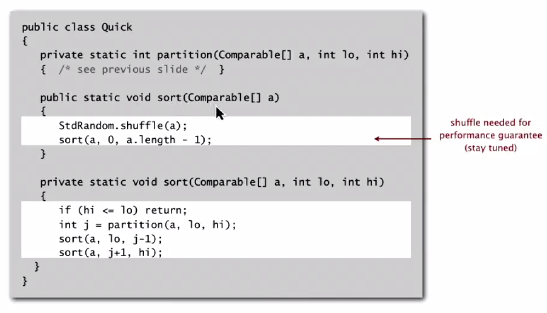


Partitioning for quicksort

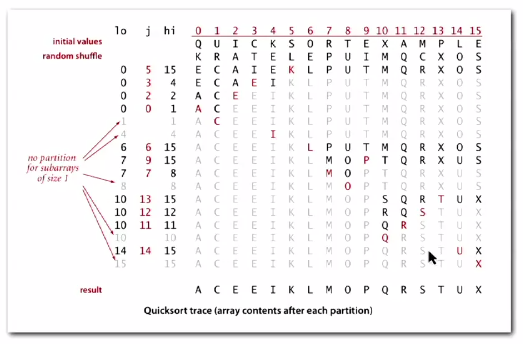
Implementation of partitioning



Quicksort implementation



Quicksort trace



Quicksort implementation details

* SORTS IN PLACE (memory is efficient)
* Terminating loop (when pointers cross) can be tricky, *especially in presence of duplicate keys)*
* Staying in bounds is challenging
* Key point: array needs to be shuffled for performance guarantee
* Equal keys: it is better to stop on keys equal to the partitioning item’s key

**Best case** time is N log N (when quicksort divides everything exactly in half

**Worst case** is ½ N2 (random shuffle puts items exactly in order, partitioning doesn’t do much of anything, discovering everything to the right is greater); THIS IS EXTREMELY RARE

**Average case**: 1.39N log N… 39% more compares than mergesort, but faster in practice because of less data movement

Caveat emptor: many textbook/ web implementations go **quadratic** if array

* Is sorted or reverse sorted
* Has many duplicates (even if randomized!)

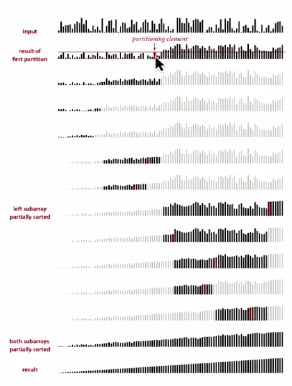
Considerations:

* Partitioning is constant extra space
* Depth of recursion is logarithmic extra space (high probability)
* QUICKSORT IS NOT STABLE

Improvements:

* Insertion sort for subarrays of 10-20 items (improve speed by 20%) or
  + Delay insertion sort until one pass at the end
* Estimate partitioning element (pivot point) as near the middle:
  + Not good for large samples
* Sample items and take median of sample
  + Median-of-three: Get median of first, middle and last elements of the array (10% faster)

Quicksort visualized with improvements



Selection

Given an array of N items, find the kth largest

Eg. Min is (k = 0), max is (k = N -1), median is (k = N/2)

What is the efficiency?

Theory:

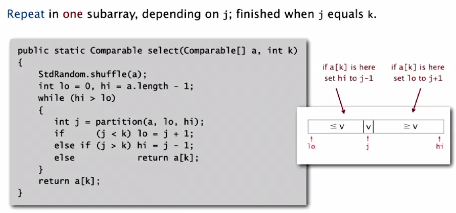
* N log N in general (upper bound)

Sort array and look for smallest, median and largest values in first, median and last indices respectively

* N lower bound  
  search through everything in small array and choose lowest/ highest/ median

Use a version of quicksort (quickselect) to find value quickly

Implementation:



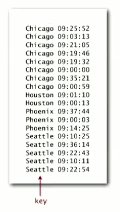
Random shuffle at beginning is important for performance

Quickselect takes **linear time on average**

**Quadratic time ½ N2 in worst case** for same reason as quicksort

Duplicate keys

Why? Often, sorts are used to bring items with equal keys together



Mergesort always uses between ½ N log N and N log N compares, so quicksort has no effect

**Quicksort is QUADRATIC unless partitioning stops on equal keys!!** Why?

Mistake: All items equal to partitioning item are placed on one side, leading to ½ N2 compares when all keys are equal

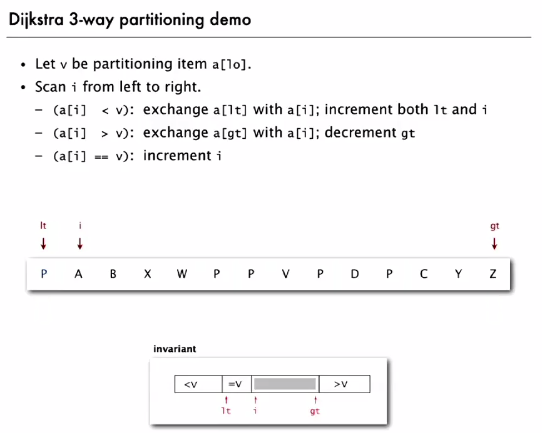
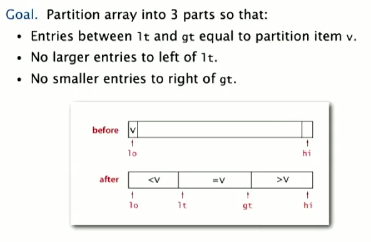


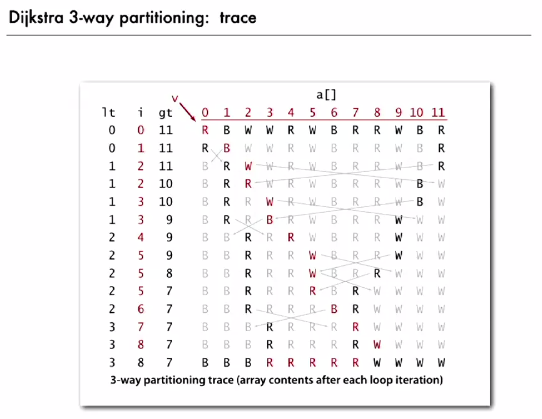
INSTEAD: Stop scan on items equal to partitioning item (N log N)

C:\Users\Zach\AppData\Local\Microsoft\Windows\INetCache\Content.Word\qsdn2.png

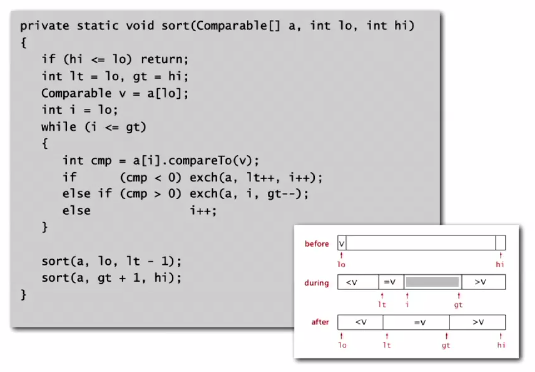
BEST: Put all items equal to the partitioning item in orderC:\Users\Zach\AppData\Local\Microsoft\Windows\INetCache\Content.Word\qsdb.png

Solution: 3-way partitioning





Dijkstra 3-way partitioning implementation



Bottom line: randomized quicksort with 3-way partitioning reduces running time from Linearithmic to linear in broad class of applications.

Sorting algorithm applications

* Obvious:
  + List of names
  + Organize MP3 library
* Problems easier with sorting:
  + Find the median
  + Identify statistical outliers
* Non-obvious
  + Data compression
  + Computer graphics

Summary of sorting algorithm running times

