Quicksort

Basic Plan

- shuffle the array
- partition so that, for some j
  - entry a[j] is in place
  - no larger entry to the left of j
  - no smaller entry to the right of j
- sort each piece recursively

Partitioning

I) • repeat until pointers i and j cross
  • scan i from left to right as long as a[i] < a[lo]
  • scan j from right to left as long as a[j] > a[lo]
  • exchange a[i] with a[j]

II) • when pointers cross → exchange a[lo] a[j]

- partitioning in place : using an extra array makes partitioning easier (and stable) but not worth the cost
- terminating the loop : testing if the pointers cross is a bit trickier than it might seem
- staying in bounds : the (j = = lo) test is redundant (why) but the (i = = hi) test is not
- preserving randomness - shuffling is needed for performance guarantee.
- equal keys - when duplicates are present it is better to stop on keys equal to the partitioning item's key

Quicksort : best case analysis

best : no. of compares is $\sim N \lg N$

worst : no. of compares is $\sim 1/2\, N^2$

$\hookrightarrow$ if the random shuffle places the array in order (sorted)

- **average case analysis**

Proposition : the average no. of compares C_N to quicksort an array of N distinct keys is $\boxed{\sim 2N \ln N}$ and the number of exchanges is $\boxed{\sim 1/3\, N \ln N}$

Pf. C_N satisfies the recurrence $C_0 = C_1 = 0$ and for $N \geq 2$

left $\qquad$ right

$$C_N = (N+1) + \left(\frac{C_0 + C_{N-1}}{N}\right) + \left(\frac{C_1 + C_{N-2}}{N}\right) + \dots + \left(\frac{C_{N-1} + C_0}{N}\right)$$

partitioning

partitioning probability

multiply by N

$$\boxed{N C_N = N(N+1) + 2(C_0 + C_1 + \dots + C_{N-1})}$$

subtract the same equation for $(N-1)$

$$\boxed{N C_N - (N-1)C_{N-1} = 2N + 2C_{N-1}}$$

$$n(n+1) - (n-1)n = n^2 + n - (n^2 - n)$$

$$= n^2 - n^2 + n + n$$

$$= 2n$$

$$N C_N = (N-1)C_{N-1} + 2 \cdot C_{N-1} + 2N$$

$$N C_N = (N+1) C_{N-1} + 2N \quad / \text{div } N(N+1)$$

$$\boxed{\frac{C_N}{N+1} = \frac{C_{N-1}}{N} + \frac{2}{N+1}} =$$

telescopes

$$= \frac{2}{3} + \frac{2}{4} + \dots \frac{2}{N+1} \simeq \int_3^{N+1} \frac{1}{x} dx$$

$$C_N = 2(N+1)\left(\frac{1}{3}+\frac{1}{4}+\cdots\frac{1}{N+1}\right)$$

$$\sim 2(N+1)\int_3^{N+1}\frac{1}{x}\,dx$$

$$C_N \sim 2(N+1)\ln N \approx \boxed{1.39 N \lg N}$$

Summary

- worst case : no. of compares is quadratic (random -> so not really)
- av. case : 35% more compares than mergesort (faster in practice, less data move.)

* can limit the depth of recursion by doing the small subarray before the large subarray

$\boxed{\text{Quicksort}}$ is not stable !!!

Improvements — insertion sort for tiny sub arrays

- best choice of pivot item - median
- estimate true median by taking median of sample
- median-of-3 (random items)

Selection

- given an array of N numbers find the k^{th} smallest number

- $N \lg N$ upper bound - sort
- easy N upper bound for $k = 1, 2, 3 \ldots$
- easy N lower bound

is selection as hard as sorting? $N \lg N$

or can it be done linear (N)

Partition array so that

- a[j] is in place
- no larger entry to the left of j
- no smaller entry to the right of j

= repeat in one subarray, depending on j
finished when j equals to k =

Mathematical analysis

$\boxed{\text{prop.}}$ Quick select takes linear time on average

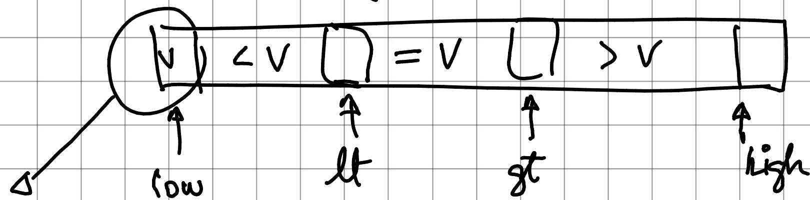
Duplicate keys

- merge sort – between 1/2 N lg N and N lg N
- quicksort – quadratic time if we don't stop partitioning on equal keys

! stop scans on items equal to the partitioning item

3-way partitioning

- partition array into 3 parts



~~pivot~~ partition item

- entries between lt and gt are equal to the partition item
- no larger entries to the left of lt
- no smaller entries to the right of gt

Plan

- v = partitioning item a[low]
- scan i from left to right

a) $\boxed{a[i] < v}$ swap a[lt], a[i]
increment lt, i

b) $\boxed{a[i] > v}$ swap a[gt], a[i]
--gt

c) $\boxed{a[i] == v}$ ++i

- Sorting lower bound
- quicksort with 3-way partitioning is entropy optimal
- linearithmic to linear in a broad class of applications

System sorts

Arrays.sort() - complex

QS - primitive types

Mergesort - Objects

* Tukey's ninther - median of the median of 3 samples (3 entries each)

- ! killer input exists - no random shuffling