Thanks to C.A.R. ('Tony') Hoare in 1950s. Published in CACM.
A divide-and-conquer algorithm.

How is it different from Merge Sort?

i) An inplace algorithm: No auxiliary space needed.
ii) Not a stable algorithm: ie, if $a_i = a_j$ then it is not guaranteed (even if $i < j$) that the order of elements in final sort will remain the same.
iii) Asymptotic time differs between the worst-case and the average-case. Unlike Merge Sort, the worst-case complexity is $\Theta(n^2)$ and the average case is $\Theta(n \lg n)$.

Sketch:

A) Divide: Partition the data $A[p \ldots r]$ s.t. a pivot is found where all elements on the left of the pivot are less or equal to the value at the pivot and all elements on the right side, are greater.

B) Conquer: Partition the array into two subarrays and call Quicksort recursively.

C) Combine: Do nothing.

$A[p \ldots q-1]$ and $A[q+1 \ldots r]$

q (pivot)

Base-Case: When $p = r$, return $A[p]$

Quicksort (A, p, r):

```
1. if p ≥ r, return and do nothing.
2. Else
3.     q ← partition (A, p, r)
4.     Quicksort (A, p, q-1)
5.     Quicksort (A, q+1, r)
```

Please review at least two ways to implement partitioning.

Analyzing Quicksort

Worst-Case

Divide: Partitioning takes $(n-1)$ comparisons and hence $\Theta(n)$ time.

The actual time taken by Quicksort relies on the implementation of the partitioning procedure.

eg., $[A, B, C, D, E, F]$

$[A, B, C, D, E]$ $[F]$ ← picking last element as pivot

then, $T(n) = T(n-1) + \Theta(n)$

Do we recognize this recurrence?

This is the recurrence for Selection Sort. Solve it by unrolling and we get $n + (n-1) + (n-2) + \cdots + 1$

So, in the **worst-case**, Quicksort takes $\Theta(n^2)$ time.

Best-Case: For a divide-and-conquer algorithm, a balanced recursion tree gets an optimal performance

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + \Theta(n)$$

$$= 2T\left(\frac{n}{2}\right) + \Theta(n)$$

$$\therefore T(n) = \Theta(n \lg n)$$

Where do we achieve this? When the pivot is always the **median** (but if we have found the median, what's left?)

Works optimally with the same asymptotic time in the best-case as Mergesort. Why not just use Mergesort then?

Average Case (Algebraic) Analysis of Quicksort.

Suppose, we have an array of n arbitrary elements with all elements likely to be chosen as a pivot with probability $1/n$ each.

Assume, a pivot p is chosen randomly, then Quicksort partitions the array into two subproblems i.e., of sizes $T(p-1)$ and $T(n-p)$. { p is included for ease of algebra.

Then the average running time is

$$T(n) = \frac{1}{n}\left[\sum_{p=1}^{n} T(n-p) + T(p-1)\right] + \underbrace{(n-1)}_{\Theta(n)\ \#\text{ of comparisons in partitioning}} \quad \text{---- (1)}$$

Recall: The sum of Harmonic Series is given as

$$\lg(n+1) \leq H_n = \sum_{i=1}^{n} \frac{1}{i} \leq 1 + \lg n$$

i.e.

$$\int_0^n \frac{1}{x+1}\,dx \leq H_n \leq 1 + \int_0^n \frac{1}{x}\,dx.$$

It is important to understand:

a) Where is the recurrence coming from?

b) How does $\lg n$ comes out from the summation from eq (1)?
The rest is just messy algebra!

Now,

$$\sum_{p=1}^{n} T(n-p) = \sum_{p=1}^{n} T(p-1)$$

i.e.

$$\sum_{p=1}^{n} T(n-p) = T(n-1) + T(n-2) + \cdots + T(1) + T(0)$$

$$\sum_{p=1}^{n} T(p-1) = T(0) + T(1) + \cdots + T(n-2) + T(n-1).$$

So, eq (1) becomes

$$T(n) = \frac{1}{n} \cdot 2 \left[\sum_{p=1}^{n} T(p-1) \right] + (n-1) \quad \text{---(2)}$$

or

$$n\,T(n) = 2 \sum_{p=1}^{n} T(p-1) + n.(n-1) \quad \text{---(3)}$$

How do we get rid of the Summation on R.H.S.?

[by substracting a lesser term]

Replace n with $(n-1)$ in eq (3):

$$(n-1).T(n-1) = 2. \sum_{p=1}^{n-1} T(p-1) + (n-1)(n-2) \quad \text{---(4)}$$

Subtracting '4' from '3', we get:

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

Also,

$$\sum_{p=1}^{n} T(p-1) = T(0) + T(1) + \cdots + T(n-2) + T(n-1)$$

$$\sum_{p=1}^{n-1} T(p-1) = T(0) + T(1) + \cdots + T(n-3) + T(n-2)$$

So,

$$n\,T(n) - (n-1)\,T(n-1) = 2\,T(n-1) + 2(n-1)$$

or $$n\,T(n) = 2\,T(n-1) + 2(n-1) + (n-1)\,T(n-1)$$

Or,

$$n\,T(n) = T(n-1)\,(2+n-1) + 2(n-1)$$

or

$$n\,T(n) = (n+1)\,T(n-1) + 2(n-1) \quad \text{— (5)}$$

We've gotten rid of the summation but still to work with the recurrence:

Dividing throughout by $n(n+1)$, we get [$\because n \geq 0$]

$$\frac{T(n)}{(n+1)} = \frac{T(n-1)}{n} + \frac{2(n-1)}{n(n+1)} \quad \text{— (6)}$$

Let,

$$a_n = \frac{T(n)}{(n-1)}, \text{ then } a_{n-1} = \frac{T(n-1)}{n}$$

i.e,

$$a_n = a_{n-1} + \frac{2(n-1)}{n(n+1)}$$

let, $a_0 = a_1 = 0$

$$a_2 = \underbrace{a_1}_{=0} + \frac{2(1)}{2(2+1)} = 0 + \frac{2.1}{2.3}$$

$$a_3 = a_2 + \frac{2(2)}{3(4)} = \frac{2.1}{2.3} + \frac{2.2}{3.4}$$

$$\vdots$$

$$a_n = 2 \cdot \sum_{i=2}^{n} \frac{(i-1)}{i(i+1)}, \quad n \geq 2$$

[Now, $\frac{i-1}{i(i+1)} = \frac{2}{i+1} - \frac{1}{i}$] [$\frac{2i-i-1}{i(i+1)} = \frac{i-1}{i(i+1)}$]

So,

$$a_n = 2 \sum_{i=2}^{n} \left[\frac{2}{i+1} - \frac{1}{i} \right], \quad n \geq 2$$

or,

$$a_n = 2 \left[\sum_{i=2}^{n} \frac{2}{i+1} - \sum_{i=2}^{n} 1/i \right], \quad n \geq 2 \quad \text{---} (7)$$

Now,

$$\sum_{i=2}^{n} \frac{2}{i+1} = \sum_{i=3}^{n+1} \frac{2}{i} \text{ (verify!)}$$

$$= \sum_{i=1}^{n} \frac{2}{i} - 2 - 1 + \frac{2}{(n+1)}$$

Also,

$$\sum_{i=2}^{n} 1/i = \sum_{i=1}^{n} 1/i - 1$$

So, eq (7) becomes,

$$a_n = 2 \left[\sum_{i=1}^{n} \frac{2}{i} - \sum_{i=1}^{n} 1/i - 2 - 1 + \frac{2}{(n+1)} - 1 \right], n \geq 2$$

$$\Rightarrow a_n \simeq 2 \left[\sum_{i=1}^{n} \frac{1}{i} + \frac{2}{(n+1)} - 2 \right]$$ → can ignore!

negligible

$$\simeq 2[\ln n + \gamma] \qquad [\text{where, } \gamma \leftarrow \text{Hn} - \ln n \text{ when } n \to \infty]$$

$$\Rightarrow a_n \simeq 2(\ln n + \gamma)$$

$$\Rightarrow \frac{T(n)}{(n+1)} = 2(\ln n + \gamma) \Rightarrow T(n) \simeq (n+1)2(\ln n + \gamma)$$

or $\boxed{T(n) = O(n \lg n)}$

Now,

The worst-case can be avoided by a few simple heuristics: (We'll return to them in the end):

1) Use the middle element as a pivot (why problematic?)
2) Use a random element.
3) Take the median-of-three (first, middle, last)

Randomization

Quicksort is good on average but bad on a certain number of cases., but suppose we pick a pivot randomly; by either picking a random pivot or scrambling the pivot

> "with high probability, randomized Quicksort runs in $O(n \lg n)$."

Randomization is a general tool to improve algorithms with bad worst-case but good average-case.

When analyzing running time of randomized algorithms, we take the expectation of the running time over the distribution of values generated

iid (independent & identically distributed)

by the underlying random number generator.

Source: CLRS

QuickSort's Average Case (Probabilistic Analysis)

Goal: Compute the total no. of comparsions performed in all calls to partition.

Let $A = [z_1, z_2, \ldots, z_n]$ where z_i is the i^{th} smallest element in A.

Define: $z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$ a set of all elements b/w z_i and z_j (inclusive).

When does Quicksort compare z_i and z_j?

Important: Each pair is compared at most once!

⇒ Elements are compared only to the pivot so, if z_i and z_j are compared it's only when z_i is a pivot or z_j is a pivot.

Once a partition method is called, the pivot element (returned) is never compared to any other elements.

Define an (indicator) random variable (R.V.)

$$X_{ij} = I\{z_i \text{ and } z_j \text{ are compared with each other at any time during execution}\}$$

Since each pair is compared at most once, we characterize the total no. of comparisons (pairwise) performed by

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \quad \text{(all pairs)}$$

Assuming that the random number generator chooses each pivot randomly and independently.

$Pr(z_i \text{ is compared with } z_j) = Pr(z_i \text{ or } z_j \text{ is the first pivot chosen from } z_{ij})$

i.e.,

$Pr(z_i \text{ is compared with } z_j) = Pr(z_i \text{ is the first pivot chosen from } z_{ij}) + Pr(z_j \text{ is the first pivot chosen from } z_j)$

$$= \frac{1}{j-i+1} + \frac{1}{j-i+1}$$

$$\Rightarrow Pr(z_i \text{ is compared with } z_j) = \frac{2}{(j-i+1)} \{x_{ij}\}$$

$$E[X] = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{(j-i+1)}\right]$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{k} \qquad [k \leftarrow j-i+1]$$

or

$$E[X] \leq 2 \sum_{i=1}^{n-1} \Sigma 1/k$$

$$\leq 2 \sum_{i=1}^{n-1} O(\lg n) \qquad [\because \Sigma 1/i \simeq O(\lg n)]$$

$$\therefore \boxed{E[X] = O(n \lg n)}$$

$\because \ln k \leq H(k) \leq 1 + \ln k$

$\Sigma 1/k$ is the Harmonic Mean

$$\int_0^H \frac{1}{x+1} dx \leq Hn \leq 1 + \int_0^H \frac{1}{x} dx$$

or $\ln(n+1) \leq Hn \leq 1 + \ln n$

$= \Sigma 1/i$

$Hn - \ln n = \gamma$

or $\gamma = Hn - \ln n$

γ is the Euler's constant for $n \to \infty$

Quicksort (Summary)

Read CLRS and Dasgupta et al. for different implementations and their time complexity of the partitioning algorithm for Quicksort.

1) The partition function has an optimized loop so very good constants (Top 10 algorithms of the 20th century!)

2) Ideally, we would want the pivot to partition the array into (nearly) equal subarrays, i.e. balanced.

Picking the pivot

1) Default: Pick the first (or the last) element. Would work if the input is randomly arranged.

If the first (or the last element) is chosen on an already sorted array, then Quicksort takes quadratic time for doing nothing!

2) Heuristics: to avoid 'bad' cases:

a) Larger of the first two distinct keys (can be 'bad' as default)

b) Choose the pivot randomly (Hoping that we have a good random number generator!)
Usually safe but wouldn't guarantee a balanced partition.

c) Use the Median [time to find the median? How!] of A[1...n]?

d) Median-of-Three: Median of three randomly chosen elements from A and use it as a pivot (could be the first, middle, and the last elements) — Works typically!