

Quick Sort

Cormen 7.1, 7.2, 7.3

Another sorting algorithm based on

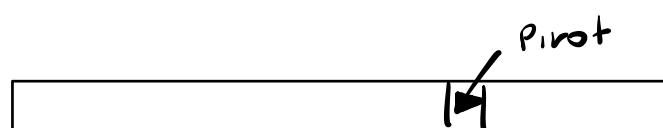
- D & C approach
- comparisons

Merge Sort

Divide : Constant time
Combine : Linear time

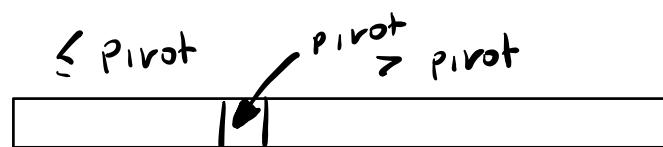
Quick Sort

Divide : Linear time
Combine : Constant time



5 6 3 2 5 8 4 2

↓ partition



↓

5 3 2 4 2 5 8 6
QS QS

Divide Choose a pivot.

Partition the array $A[p \dots r]$

in two (possibly empty) subarrays

$A[p \dots q-1]$ and $A[q+1, \dots, r]$
such that

- $\forall l \in A[p \dots q-1], l \leq \text{pivot}$
- $A[q] = \text{pivot}$
- $\forall l \in A[q+1 \dots r], l > \text{pivot}$

Conquer Stop if $p == r$.

Sort the two subarrays recursively

Combine do nothing

Quicksort (A, p, r)

if $p < r$

$q = \text{Partition} (A, p, r)$

Quicksort ($A, p, q-1$)

Quicksort ($A, q+1, r$)

Partition (A, p, r)

// Choose pivot and move it in A[r]

x = A[r]

i = p - 1

for j = p to r-1

if A[j] ≤ x

i = i + 1

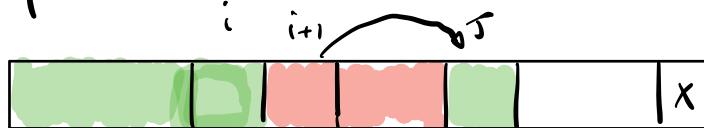
swap (A[i], A[j])

swap (A[i+1], A[r])

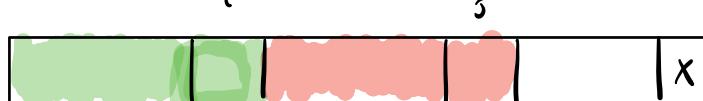
return i + 1

in place
no stable
linear time

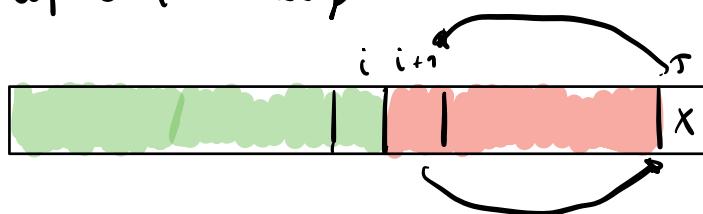
if A[j] ≤ x

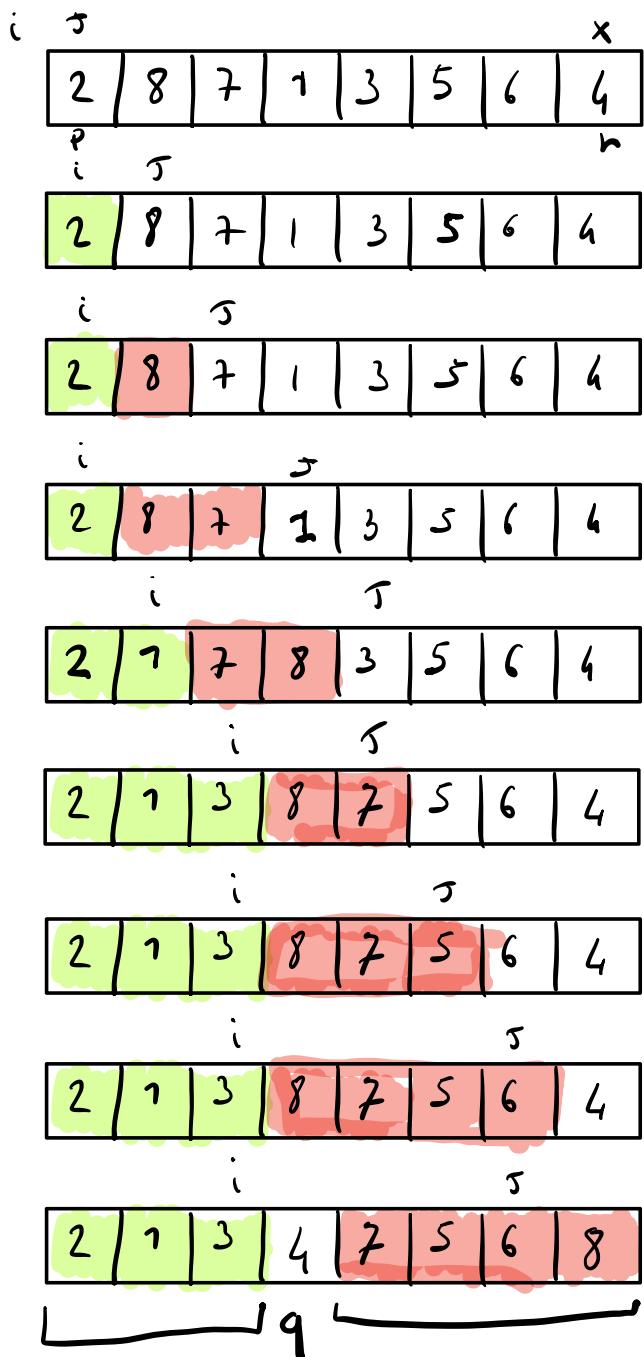


if A[j] > x



after for loop





Partition (A, p, r)

// Choose pivot and move it in A[r]

x = A[r]

i = p - 1

for j = p to r-1

if A[j] ≤ x

i = i + 1

swap (A[i], A[j])

swap (A[i+1], A[r])

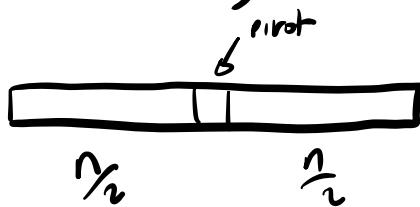
return i + 1

loop invariant is
preserved at every
iteration

Analysis of Quick Sort

Best Case : Every time the pivot is such that

(after partition)



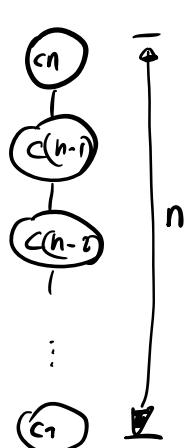
Parts are balanced

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \quad (\text{as MS}) \\ 2 T\left(\frac{n}{2}\right) + \Theta(n) & = \Theta(n \log n) \text{ time} \end{cases}$$

$$T(n) = 2 T\left(\frac{n}{2}\right) + O(1)$$

solves to $\Theta(n)$ time

Worst case:



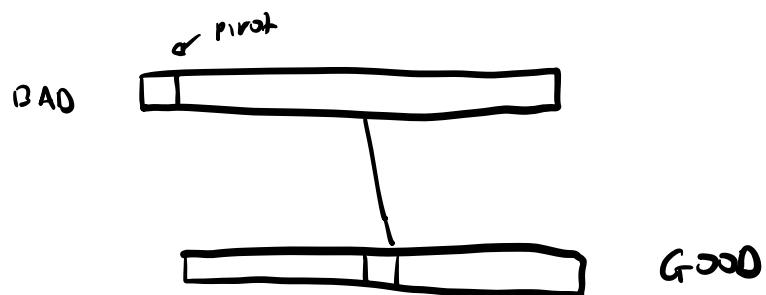
$$\begin{aligned} T(n) &= \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ T(n-1) + \Theta(n) & \end{cases} \\ &= \Theta(n^2) \text{ time} \end{aligned}$$

Average Case analysis (only intuition)

BAD	GOOD	GOOD	BAD
1	$\frac{1}{4}n$	$\frac{2}{4}n$	$\frac{3}{4}n$

$$P(\text{BAD}) = P(\text{GOOD}) = \frac{1}{2}$$

penalty of a Bad Pivot at a certain recursive call is canceled by the benefit of a good one in the next recursive call



QS expected running time is $\Theta(n \log n)$

(with high probability)

$$P(\text{run in } \Theta(n \log n \text{ time})) \geq 1 - \frac{1}{n}$$

Asymptotic Notation

Forget about constants and lower order terms

$$\cancel{100n} + \cancel{1000} = \Theta(n)$$

$$\cancel{1000} n^2 + \cancel{100n} = \Theta(n^2)$$

Θ -notation

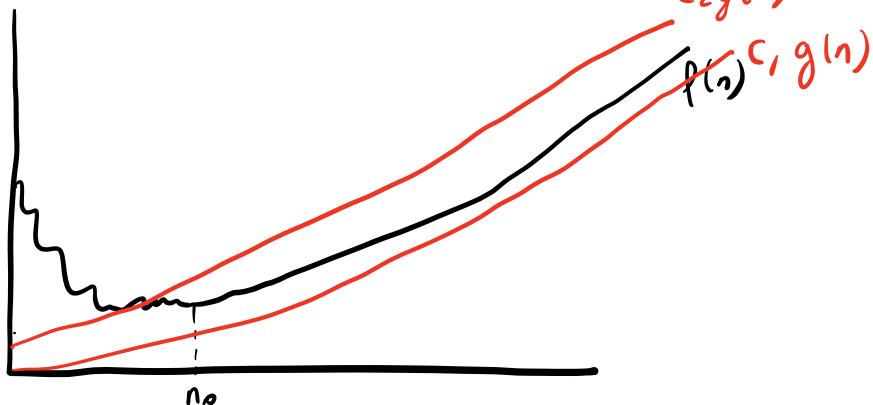
For a given function $g(n)$

$\Theta(g(n))$ denotes a set of functions

$\Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2 \text{ and } n_0 \text{ such that}$

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

for all $n \geq n_0\}$



$$P(n) \overset{=} \Theta(g(n)) \quad P(n) \overset{=} \Theta(g(n))$$

We say that $g(n)$ is an asymptotical tight bound for $P(n)$

c_1, c_2

$$\cancel{\frac{1}{2} n^2 - 3n} \in \Theta(n^2)$$

$$\frac{1}{1000000} n^3 + n^2 + 3n \notin \Theta(n^3)$$

$$\cancel{\notin} \Theta(n^4)$$

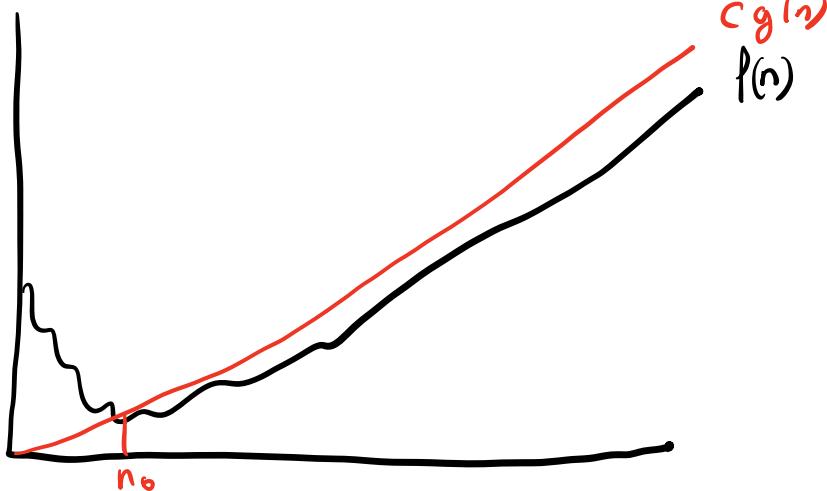
O-notation (Big-Oh)

$O(g(n))$ pronounced big-Oh of g of n

$O(g(n)) = \{ p(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that}$

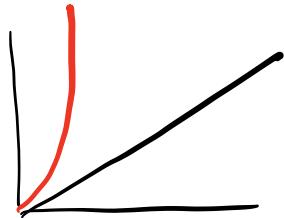
$$0 \leq p(n) \leq c g(n)$$

for all $n \geq n_0\}$



$$\cancel{n^2} + \cancel{s^n} \in O(n^2)$$

$$\begin{array}{c} \cancel{n^2} \\ \cancel{s^n} \end{array} \Rightarrow \begin{array}{l} O(n^3) \\ O(n^4) \\ O(n!) \\ O(2^n) \end{array}$$



$$p(n) \in \Theta(g(n)) \Rightarrow p(n) \in O(g(n))$$

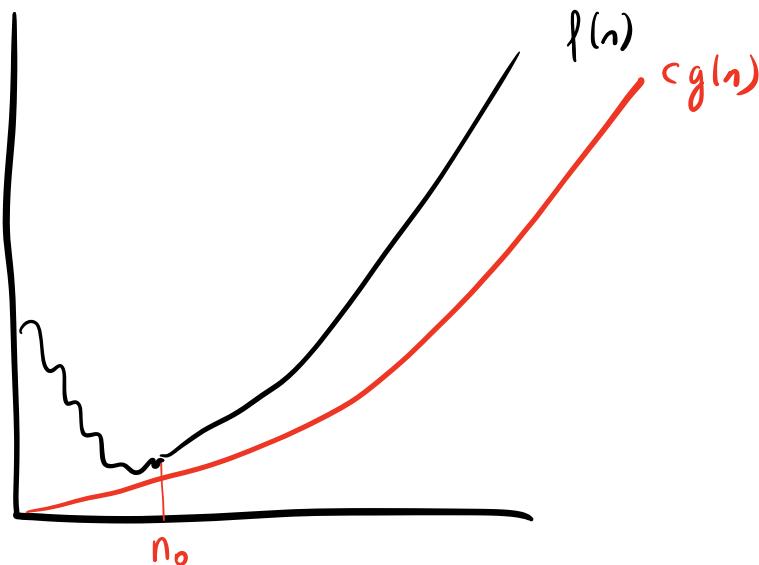
$$O(1) \subset O(\log n) \subset O(\sqrt{n}) \subset O(n) \subset O(n \log n) \subset O(n^2)$$

↑
faster

↓
slower → $O(n^2)$

Ω - notation Asymptotic Lower bound

$\Omega(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 < cg(n) \leq f(n) \text{ for all } n \geq n_0 \}$



$$3n^2 + 4n \in \Omega(n^2)$$

$$\subset \Omega(n)$$

$$\cancel{\Omega(1)} \quad \Omega(1)$$

$$\cancel{\Omega(n^3)}$$

↙ opposite direction

$$\Omega(1) \supset \Omega(\log n) \supset \Omega(\sqrt{n}) \supset \Omega(n) \supset \Omega(n \log n) \supset \Omega(n^2)$$

↗ faster

↙ slower $\rightarrow \Omega(n^3)$

For any functions $g(n)$ and $f(n)$

$$we have \quad f(n) = \Theta(g(n))$$

if and only if

$$f(n) = O(g(n)) \quad \text{and} \quad f(n) = \Omega(g(n))$$

————— O —————

QS runs in $O(n^3)$ time

QS runs in $O(n^4)$ time

QS runs in $\Theta(n^2)$ time

~~QS runs in $\Theta(n^4)$ time~~

$\Theta(f(n))$ if worst case is $O(f(n))$

and there is an instance such that
your algorithm takes $\Omega(f(n))$ time

Exercise

$f(n) = 3n^2 \in$	$O(n^2)$	✓
	$O(n^4)$	✓
	$O(n!)$	✓
	$O(2^n)$	✓
	$O(n \log n)$	✗
	$O(n^2 \log n)$	✓
	$\Theta(n^2)$	✓
	$\Theta(n \log n)$	✗
	$\Omega(n^2)$	✓
	$\Omega(n)$	✓
	$\Omega(n^3)$	✗