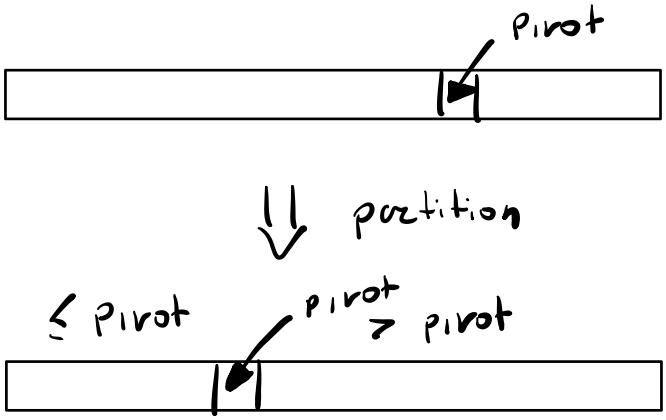
Quick Sort Cormen 7.1, 7.2, 7.3

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

5 3 2 4 2 5 8 6

QS QS

Divide Choose a pivot.

Partition the array $A[p \ldots r]$ in two (possibly empty) subarrays $A[p \ldots q-1]$ and $A[q+1, \ldots, r]$ such that

- $\forall \ell \in A[p \ldots q-1], \ell \leq pivot$
- $A[q] = pivot$
- $\forall \ell \in A[q+1 \ldots r], \ell > pivot$

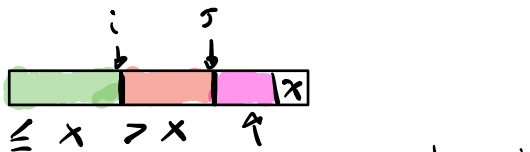
Conquer Stop if $p == r$.

Sort the two subarrays recursively

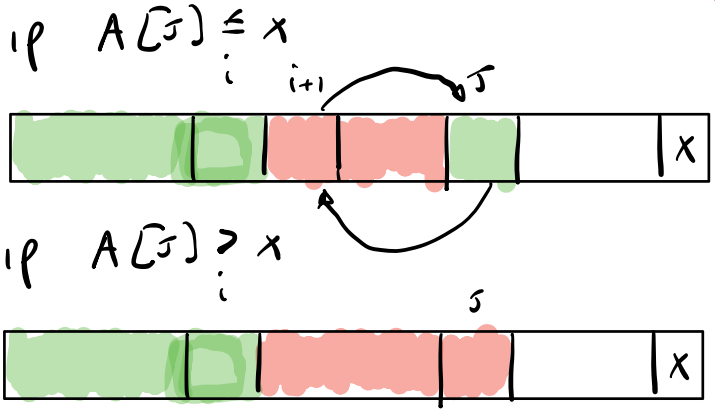
Combine do nothing

```
QuickSort ( A, p, r)
    if p < r
        q = Partition ( A, p, r)
        QuickSort ( A, p, q-1)
        QuickSort ( A, q+1, r)
```

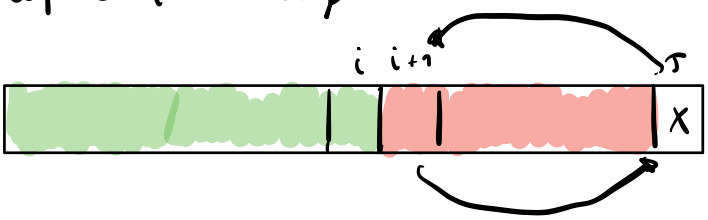
```
Partition ( A, p, r)
    // Choose pivot and move it in A[r]
    x = A[r]
    i = p - 1
    for j = p to r-1
        if A[j] ≤ x
            i = i + 1
            swap ( A[i], A[j])
    swap ( A[i+1], A[r])
    return i+1
```

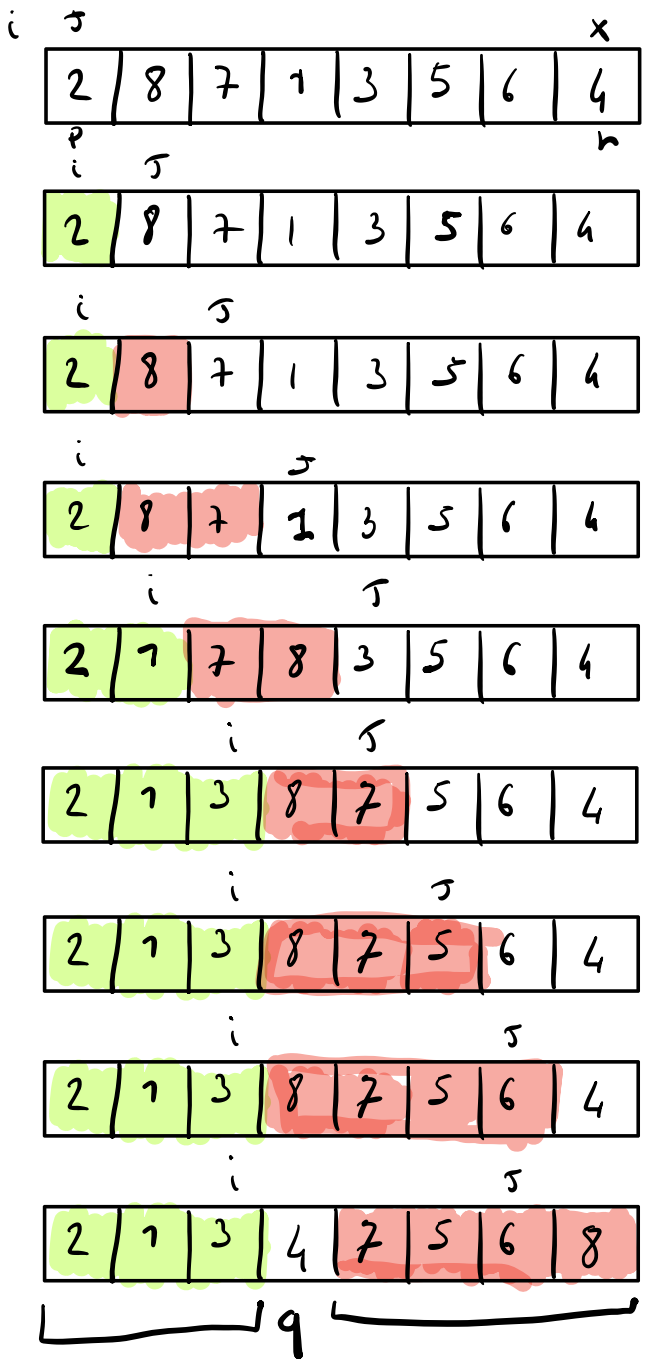


in place
no stable
linear time

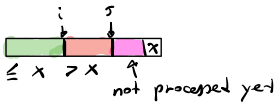


after for loop





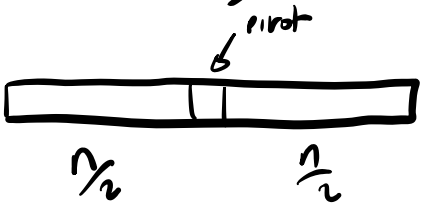

```
Partition ( A, p, r)
    // Choose pivot and move it in A[r]
    x = A[r]
    i = p - 1
    for j = p to r-1
        if A[j] ≤ x
            i = i + 1
            swap ( A[i], A[j])
    swap ( A[i+1], A[r])
    return i+1
```



loop invariant is preserved at every iteration

Analysis of QuickSort

Best Case: Every time the pivot is such that (after partition)

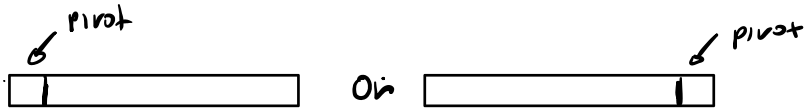


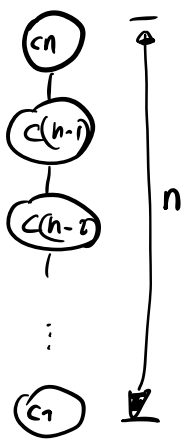
Parts are balanced

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

$T(n) = 2\,T\left(\frac{n}{2}\right) + \Theta(1)$ solves to $\Theta(n)$ time

Worst case:

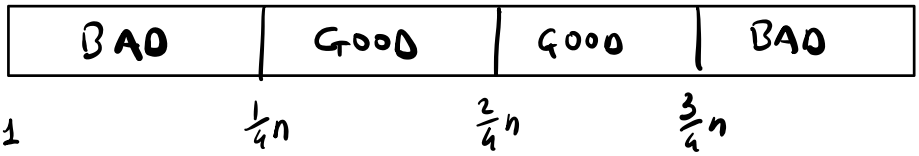




$$T(n) = \begin{cases} \Theta(1) & \text{if } n \le 1 \\ T(n-1) + \Theta(n) \end{cases}$$

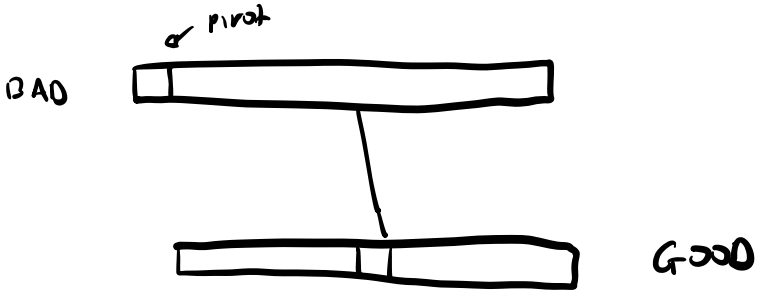
$$= \Theta(n^2) \text{ time}$$

Average Case analysis (only intuition)



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

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

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

$$\cancel{\frac{1}{1000}}\, n^2 + \cancel{100}\, n = \Theta(n^2)$$

$\in$

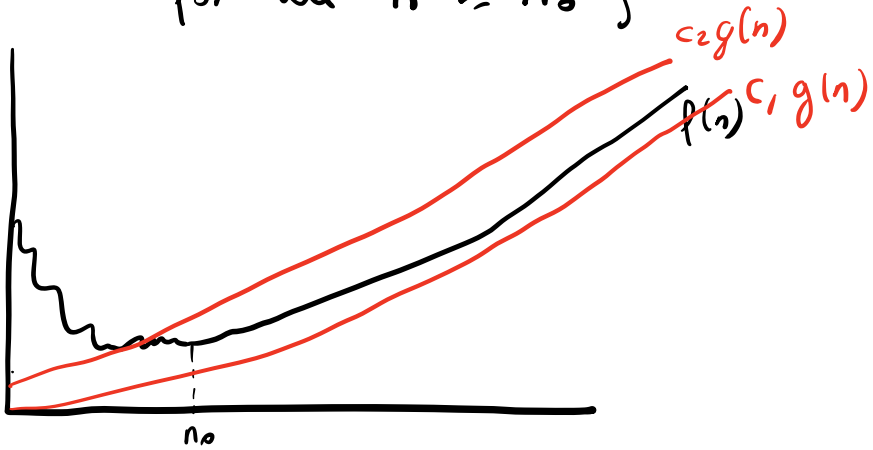
Θ-notation

For a given function $g(n)$

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

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

$$0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0 \}$$



$p(n) \in \Theta(g(n))$ $\quad$ $p(n) \not\in \Theta(g(n))$

we say that $g(n)$ is an asymptotical tight bound for $p(n)$

c_1, c_2

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

$\frac{1}{1000000} n^3 + n^2 + 3n \not\in \Theta(n^2)$

$\in \Theta(n^3)$

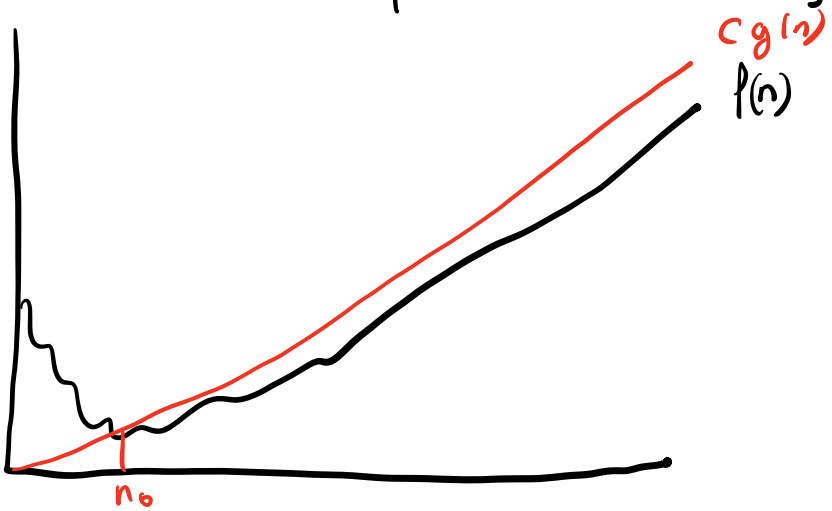
$\not\in \Theta(n^4)$

O - notation (Big-Oh)

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

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

$0 \leq p(n) \leq cg(n)$
for all $n \geq n_0 \}$



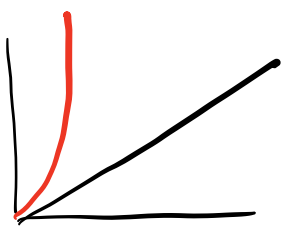
~~3~~n^2 + ~~5~~$n \in O(n^2)$

$\Uparrow \quad \Uparrow O(n^3)$

$\Theta(n^4) \Uparrow \quad O(n^4)$

$\Uparrow \quad O(n!)$

$\Uparrow \quad O(2^n)$



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

$\not\Leftarrow$

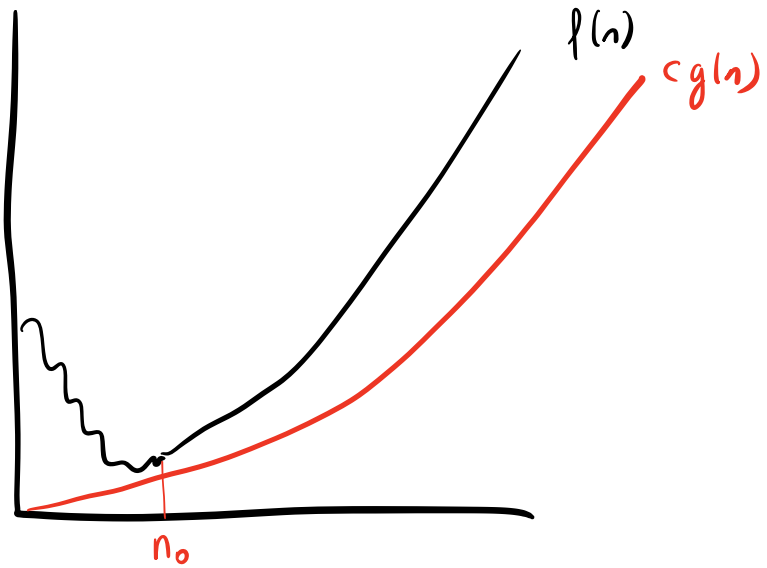
$$O(1) \subseteq O(\log n) \subset O(\sqrt{n}) \subset O(n) \subset O(n \log n) \subset O(n^2)$$

↑ faster

slower → $O(n^3)$

Ω - notation Asymptotic lower bound

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



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

$\subseteq \Omega(n)$

$\not\subseteq \Omega(1)$

$\Omega(n^3)$

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 → $\Omega(n^3)$

For any functions $g(n)$ and $f(n)$
we have $f(n) = \Theta(g(n))$
if and only if
$f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

QS runs in $O(n^2)$ 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

$f(n) = 3n^2 \in$

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