Lecture 3: Analysis of simple algorithms

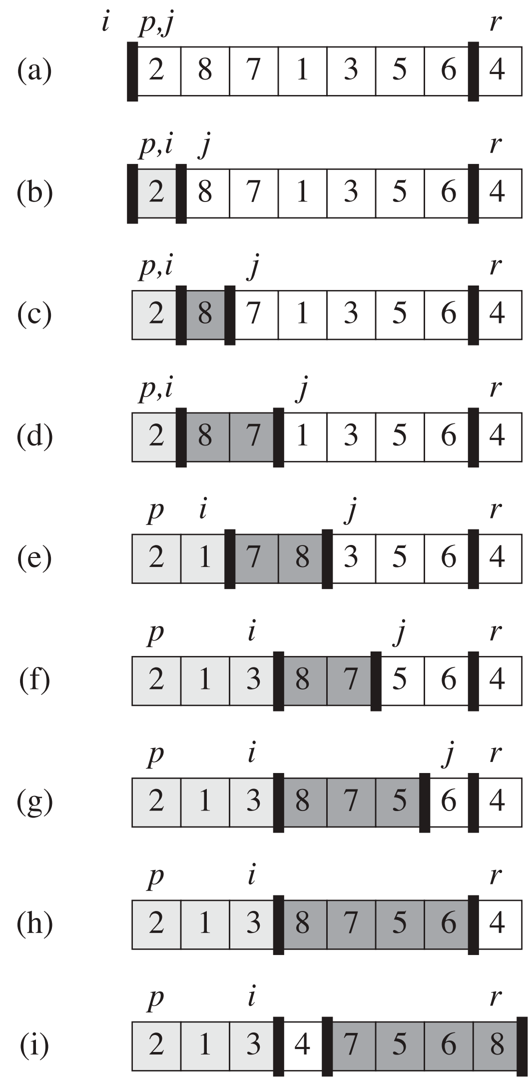
- Quicksort and its complexity
- Bisection and its complexity

Quicksort algorithm

- a vector of numbers c of length n, start location for sort p, end location for sort q
- a recursive algorithm using "divide-and-conquer"
- quicksort peudocode

```
quicksort(c, p, q)
r := findpivot(c, p, q)
quicksort(c, p, r-1)
quicksort(c, r+1, q)
```

Demo: http://me.dt.in.th/page/Quicksort/


(a) i p,j r
2 8 7 1 3 5 6 4
(b) p,i j r
2 8 7 1 3 5 6 4
(c) p,i j r
2 8 7 1 3 5 6 4
(d) p,i j r
2 8 7 1 3 5 6 4
(e) p i j r
2 1 7 8 3 5 6 4
(f) p i j r
2 1 3 8 7 5 6 4
(g) p i j r
2 1 3 8 7 5 6 4
(h) p i r
2 1 3 8 7 5 6 4
(i) p i r
2 1 3 4 7 5 6 8

Complexity

- analysis of algorithm: to determine the amount of resources (time and storage) needed to execute an algorithm

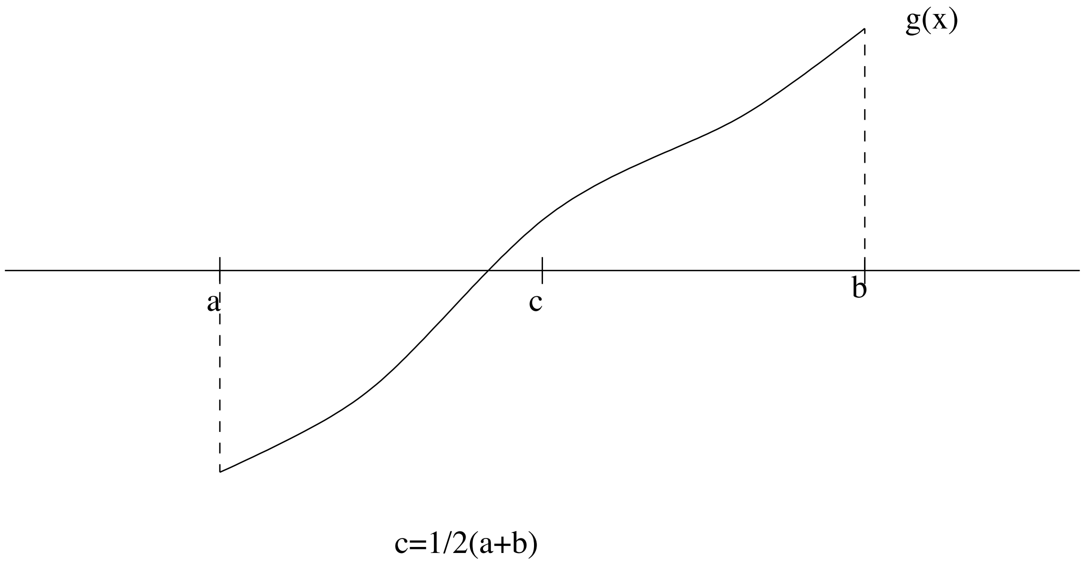
- time complexity: count the # of operations (flops)

- asymptotic analysis and big O notation
e.g. $f(n) = 9 \log n + 5(\log n)^3 + 3n^2 + 2n^3 = \mathcal{O}(n^3)$, as $n \to \infty$

- average complexity of quicksort algorithm: $\mathcal{O}(n \log n)$
Proof

Bisection

- for a non-decreasing function $g(x)$, find x such that $g(x) = 0$
- bisection:
  - start with $g(a) < 0 < g(b)$
  - take $c = \frac{1}{2}(a + b)$
  - if $g(c) < 0$, consider right half interval $[c, b]$
  - if $g(c) > 0$, consider left half interval $[a, c]$
  - repeat the above corresponding subinterval



- after n iterations, the final bracketing interval has length $2^{-n}(b-a)$
- the solution ("crossing point") is included in this interval
- the length of the interval converges to 0 as $n \to \infty$ (exponential convergence rate)