ID1020 2018-09-07 #3

Analysing algorithms to understand their performance

Observations, Mathematical models, Running time classes
Complexity theory, memory complexity.

Memory complexity is important.
Strive to solve problems faster with less memory. Enviroment!

Analysis is needed to understand what is possible
DoS can exploit worst cases

Different views: Functioning, Efficient, Understandable

Estimate running time, compare algorithms

Discrete Fourier transform (DVD, JPEG, MRI)
Brute force: N^2 FFT agorithm: $N \log N$

Big-O Complexity $O(1) \to O(n!)$

Observate, model, predict, check
Experiments should be reproducable. Falsifabillity

Brute force "tests all cases"
Doubling ratio-hypothesis. Run the program and double problem size.
System dependent vs System independent

Mathematical model $\text{running time} = \sum_{o \in \text{operations}} \text{cost}_o \cdot \text{frequency}_o$

Cost is system dependent, frequency is system independent
Cost model. Analyse one primitive operation
Omit lower-order terms for big N
Tilde approximation, Approximate sum with integral
Approximative models are often sufficient.
order-of-growth classes

$O(1)$	constant	Add two numbers
$O(\log N)$	logarithmic	Binary search
$O(N)$	linear	Find max/min
$O(N \log N)$	linearithmic	Mergesort
$O(N^2)$	quadratic	Check all pairs
$O(N^3)$	cubic	Check all triplets
$O(2^N)$	Exponential	Check all subsets

Binary search $\in O(\log N)$

Rule of thumb. Better (lower) time complexity $\Rightarrow$ Faster algorithms

Best case: Lower bound Worst case: Upper bound
Average case: Expected for "random" input.

Amortized complexity
Some operations/cases with high cost

Complexity theory
- How hard?
- Optimal algorithms

Big Theta (Θ) asymptothic complexity
Big Oh (O) Upper bound
Big Omega (Ω) Lower bound

$f(n) = O(g(n))$ if $f\ \exists c >$ and $n_0 > 0$ where $f(n) \leq c g(n) \forall n \geq n_0$

$\Omega(g(n)) = \{f(n) \exists c > 0$ and $n_0 > 0$ where $0 \leq c g(n) \leq f(n) \forall n \geq n_0\}$

$\Theta(g(n)) = \{f(n) \exists c_1, c_2, n_0 > 0$ where $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \forall n \geq n_0\}$

Class P (polynomial time) P = NP?
Class NP (Non-deterministic Polynomial time)

A problem is NP-complete when it is both NP-hard.

- An NP-hard problem is at least as computationally hard as the hardest problems in NP.

- Design an algorithm
- Prove a lower bound

Is the (theoretical) lower bound lower than the upper bound (of the algorithm)?
- Lower the upper bound (design better algorithm)
- Raise the lower bound (harder)

Golden era of algorithm design: 1970s
In ID1020 we focus on Tilde notation

Memory complexity

64-bit computer (or more precise OS) → 8-byte pointer/adress
Object overhead 16 bytes
Reference 8 bytes
Padding

Array overhead 24 bytes

Shallow memory usage
Deep memory usage

boolean:	1 B
byte:	1 B
char:	2 B
int:	4 B
float:	4 B
long:	8 B
double:	8 B

Memory profiling

Emperic analysis
Mathematical analysis
Scientific method