Fundamental Algorithmic Techniques II

January 28, 2026

Outline

Name and Definition

Definition

Explicit, precise and unambiguous instructions describing mechanically executable sequence to achieve specific purpose.

Algorithms are omnipresent in modern world!
In our understanding of the world!

Confused Name!

- $\alpha\lambda\gamma o\zeta$ (algos) = "pain", $\alpha\rho\iota\theta\mu o\zeta$ (arithmos)= "number"!?
- Muhammad ibn Musa al-Khwarizmi, c.780–c.850
  Algebra/Null, born in/near Kazakhstan

Al-Khwarizmi $\longrightarrow$ "Algorism" in medieval Italy
$\longrightarrow$ **Algorithm** by "correction/confusion"

Algorithms Description

1. **What?** Specify the Problem
2. **How?** Describe Algorithm (Pseudocode and english)
3. **Why?** Proof (induction, ...)
4. **Performance?** Analysis (time and space complexity, ...)

"Thinking and solving Algorithms":

- **Healthy, powerful Basis for thinking!**
- Basis and support for Communication (1. & 4.)!

Importance of Algorithms for Developer

For oneself:

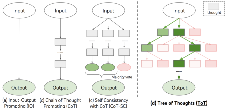
- Solve Problems!
- **Toolbox for cleaner, better, faster designs**
- Better communication/confidence
- Know/intuit what is a good/bad/solvable/unsolvable problem
- Helpful working with AI

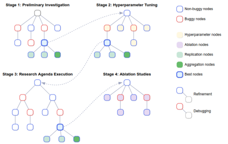
Industry:

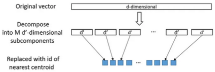
- Way to screen candidates
- Performance/costs awareness or optimisation
- Communication is key!

Omnipresence of Algorithms

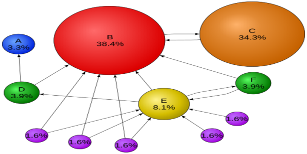
Commercial, Fin., Tech., Industry, Economy, **AI**:

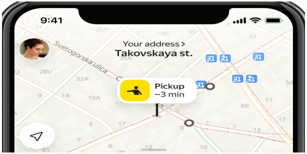






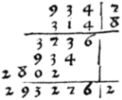
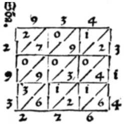
Tree Of Thoughts/ Tree based Experimentation for AI Researcher/ RAG





Left: PageRank, Right: Yandex

Introductory Examples

Left Rhind Papyrus, Right, Fibonacci Lattice

- Multiplication Algorithms:
  - Peasant Multiplication ($\sim$ 2000 BC, Rhind Papyrus)
  - Fibonacci Lattice $\sim$ 1600
- Real World Example: naive N^2 sorting taking days!

Algorithmic Performance Model

Random Access Machine **RAM** model:

- $\approx$ computer independant
- each operation take $\approx$ same compute
- operation input size $\approx$ independant
- input

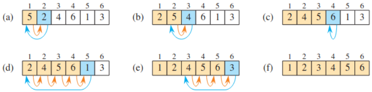
Operations:

1. $+, -, *, /, ...$
2. return and comparisons $==, >, <, \geq, \leq, \%, ...$
3. variables access, allocation or change

!!!Subroutines or iterations are not considered operations!!!

This is an approximation!!!

Full Analysis: Insertion sort



INSERTION-SORT(A, n)	*cost*	*times*
for $i = 2$ **to** n	c_1	n
$key = A[i]$	c_2	$n-1$
// Insert $A[i]$ into the sorted subarray $A[1:i-1]$.	0	$n-1$
$j = i - 1$	c_4	$n-1$
while $j > 0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
$A[j+1] = A[j]$	c_6	$\sum_{i=2}^{n} (t_i - 1)$
$j = j - 1$	c_7	$\sum_{i=2}^{n} (t_i - 1)$
$A[j+1] = key$	c_8	$n-1$

Left: schema, Right: code

Full Analysis:

$$\#(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{i=2}^{n} t_i + (c_6 + c_7) \sum_{i=2}^{n} (t_i - 1) + c_8(n-1)$$

$$= (c_5 + c_6 + c_7)/2n^2 + (c1 + c2 + c4 + c5 - c6 - c7 + c8)n$$

$$-(c2 + c4 + c5 + c8), \quad \text{with} \sum_{i=2}^{n} t_i = \frac{n(n+1)}{2} - 1, \sum_{i=2}^{n} (t_i - 1) = \frac{n(n-1)}{2}$$

... Tedious to analyse like this...

Approximating Performance

- Time Complexity:
  Best, Worst, and Average-case Complexity
- Space Complexity: total memory an algorithm requires to solve a problem

Both quantities vary a lot for some algorithms:
$\Rightarrow$ simplify with assymptotic $\mathcal{O}$ notations.

$\mathcal{O}$, Θ, Ω asymptotic Notations

Ideas:

1. Routines like functions scaling with the problem size $N : f(N)$
2. At N large, $N \ll N\log(N) \ll N^2 \ll ... \ll \exp(N)$
   $\Rightarrow$ For approximation, just scale matters!

- $\mathcal{O}$: **Big O notation**
- Θ: upper bound
- Ω: lower bound

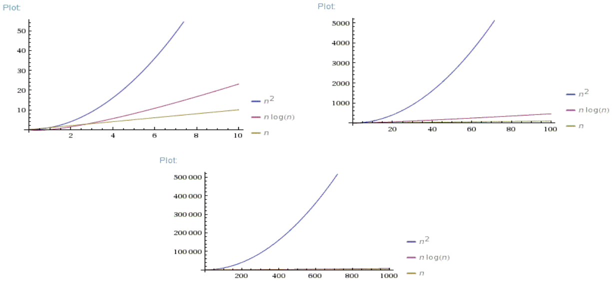
Examples:

1. $\#_1(n) = n$, $\#_2(n) = 3n + 6$, $\#_3(n) = 15n$: $\Theta(n), \mathcal{O}(n), \Omega(n)$.
2. $\#(n) = 7n^3 + 100n^2 + 20n$ is $\Theta(n^3), \mathcal{O}(n^3)$, $\Omega(n^2)$
3. $\#(x) = (13 + 2 + 7 + 1111)n^{45}$ is $\Theta(n^{45}), \mathcal{O}(n^{45})$, $\Omega(n^{45})$

Usage of $\mathcal{O}$ notations

Most of the time, we are interested in $\mathcal{O}$ and use it for:

- Algorithm Analysis
- Algorithm Comparisons



$\mathcal{O}(N) = N^2$ *vs* $N\log N$ *vs* N as N increases

When/Why $\mathcal{O}(n) \approx log_2(n)$

Consider, as for peasant multiplication:

$$T(n) \approx n/2$$

For a given n the algorithms decomposition takes m steps:

$$n = 2^{\mathrm{m}}$$

$$\Rightarrow \log_2(n) = \mathrm{m}$$