Lecture 2: Basics of Algorithm Analysis

subject	TDT4121
topics	Runtime
created	@August 29, 2022 3:01 PM

Chapter 2 - Basics of Algorithm Analysis

> Analyzing algorithms involves thinking about how their resource requirements—the amount of time and space they use—will scale with increasing input size.

Goals 🏁

The student should be able to:

- Understand the different definitions of an efficient algorithm
- Understand asymptotic notation and the asymptotic bounds O, Ω, Θ as well as their properties
- Understand how the stable matching problem is implemented using arrays and lists
- Have knowledge of common running-time bounds and some of the typical approaches that lead to them
- Understand what a priority queue is and how heaps are used to implement them

Computational Tractability

> "For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious.

> But once unlocked, they cast a brilliant new light on some aspect of computing." - *Francis Sullivan*

Polynomial-Time

Brute Force

For many non-trivial problems, there is a natural brute force search algorithm that checks every possible solution.

- Typically takes 2^N time or worse for inputs of size N.
    - This is $N!$ for stable matching with n men and n women
- Unacceptable in practice.

Desirable scaling property

When the input size doubles, the algorithm should only slow down by some constant factor C.

- There exists constants $c > 0$ and $d > 0$ such that on every input of size N, its running time is bounded by cN^d steps.

> An algorithm is poly-time if the above scaling property holds - choose $C = 2^4$.

Worst case running time

Obtain bound on largest possible running time of algorithm on input of a given size N.

- Generally captures efficiency in practice.
- Draconian view, but hard to find effective alternative.

Average case running time

Obtain bound on running time of algorithm on random input as a function of input size N.

- Hard (or impossible) to accurately model real instances by random distributions.

- Algorithm tuned for a certain distribution may perform poorly on other inputs.

Worst-case polynomial-time

An algorithm is efficient if its running time is polynomial

Justification - it really works in practice

- although $6,02 * 10^{23} * N^{20}$ is technically poly-time,it would be useless in practice
- In practice, the poly-time algorithms that people develop almost always have low constants and low exponents.
- Breaking through the exponential barrier of brute force typically exposes some crucial structure of the problem.

Exceptions

- Some poly-time algorithms do have high constants and/or exponents, and are useless in practice.
- Some exponential-time (or worse) algorithms are widely used because the worst-case instances seem to be rare.

Asymptotic Growth Rate

The **asymptotic** behaviour of a function $f(n)$ (such as $f(n) = c * n$ or $f(n) = c * n^2$, etc.) refers to the growth of $f(n)$ as n gets large. We typically ignore small values of n, since we are usually interested in estimating how slow the program will be on large inputs. A good rule of thumb is: the slower the asymptotic growth rate, the better the algorithm (although this is often not the whole story).

Asymptotic Upper Bounds

$T(n)$ is $O(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$ we have $T(n) \leq c * f(n)$.

- You can think of it as an “at most” statement

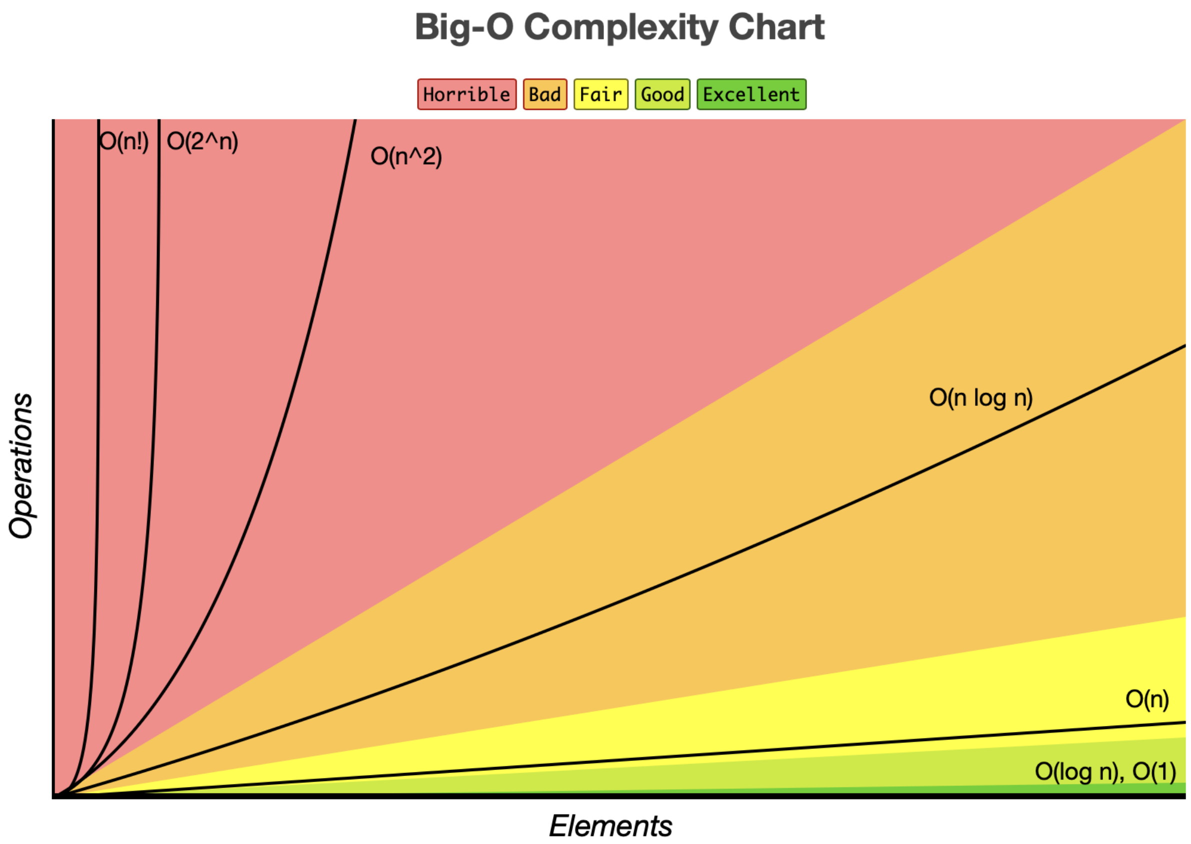
Asymptotic Lower Bounds

$T(n)$ is $\Omega(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$ we have $T(n) \leq c * f(n)$.

- You can think of it as an. “at least” statement

Asymptotically Tight Bounds

$T(n)$ is $\Theta(f(n))$ if $T(n)$ is both $O(f(n))$ and $\Omega(f(n))$



Properties of Asymptotic Growth Rates

Transitivity: A first property is *transitivity*: if a function *f* is asymptotically upper-bounded by a function *g*, and if *g* in turn is asymptotically upper-bounded by a function *h*, then *f* is asymptotically upper-bounded by *h*. A similar property holds for lower bounds. We write this more precisely as follows.

Example:

If $f = O(g)$ and $g = O(h)$, then $f = O(h)$
If $f = \Omega(g)$ and $g = \Omega(h)$, then $f = \Omega(h)$

n log *n* ≤ *n*2 for all *n* ≥ 1.

Asymptotic Bounds for Some Common Functions

Polynomials: a0 + a1n + ... + ad nd is □(nd) if ad > 0

Polynomial time: Running time is for some constant d independent of the input size n.

Logarithms: O(log a n) = O(log b n) for any constants a, b > 0 can avoid specifying the base
Logarithms. For every x > 0, log n = O(nx).

log grows slower than every polynomial

Exponentials: For every r > 1 and every d > 0, nd = O(rn).