Complexity Theory

Disclaimer

This follows the lessons from Mr. Berrachedi given in course, major additions, changes or rearrangements can be applied on the course for the sake of clarity, better explainations or just subjective opinions, written by HADIOUCHE Azouaou. USE THIS AT YOUR OWN RISK.

Chapter 1

Introduction

Complexity theory revolves around the idea of measuring execution time of an algorithm giving a measure of the execution time given some taken parameters about the inputs.

To introduce the concept of computation complexity, we start by considering the calculation of a determinant of a square matrix, we can calculate it with two different ways, first by the definition of a determinant, the second with LU decomposition.

1. We start by determining how many steps needed for calculating the determinant of an $n \times n$ matrix by definition. Let $C_1(n)$ be the number of steps for evaluating the determinant of an $n \times n$ matrix $A = \left(A_i^j\right)$. By definition we have that

$$\det A = \sum_{\sigma \in S_n} \operatorname{sgn} \sigma \prod_{i=1}^{n} A_i^{\sigma(i)}.$$

   We have $\# S_n = n!$, if we suppose that $\operatorname{sgn} \sigma, \sigma(i)$ can be calculated in a constant time, then the remaining product has n steps to evaluate thus we have that $C_1(n) = n \cdot n!$
2. Now for the calculation of determinant using the LU method, denote $C_2(n)$ the number of steps needed for this calculation, if we denote $T(n)$ be the amount of steps needed to do the LU decomposition, its easy to calculate that $T(n) = ((n-2)(n-1)(2n-3))/3$ and then we get that $A = LU$, with L, U triangular, and $\det L = 1$, then $\det(A) = \det(LU) = \det L \cdot \det U = \det U = \prod_{i=1}^{n} u_{ii}$ thus $C_2(n) = T(n) + n$.

Now that we calculated the amount of steps needed for each one, we will assume that each step takes a second and that we want to calculate the determinant of 100×100 matrix. Using the first algorithm, we get that it will take $C_1(100) = 100 \cdot 100!$ seconds which is approximately $3 \cdot 10^{150}$ years, for comparision, we have that the life span of the universe is approximately 10^{10} years, if we use the second method, it will take $C_2(100) = T(100) + 100$ which takes approximately a week and a day to calculate.

For our interest, we usually do not check exactly how the algorithm behaves at each point, but how it behaves asymptotically, thus we created our comparision notations.

Definition 1.1 (Comparision Notations): *Let $f, g : \mathbb{N} \to \mathbb{R}^+$, we define the following notations:*
- $f = o(g) \Leftrightarrow \forall \varepsilon > 0, \exists N \in \mathbb{N}, \forall n \geq N, f(n) \leq \varepsilon \cdot g(n)$.
- $f = O(g) \Leftrightarrow \exists c > 0, \exists N \in \mathbb{N}, \forall n \geq N, f(n) \leq c \cdot g(n)$.
- $f = \Theta(g) \Leftrightarrow \exists c_1, c_2 > 0, \exists N \in \mathbb{N}, \forall n \geq n, c_1 g(n) \leq f(n) \leq c_2 g(n)$.

Example:
- if we consider our previous problem then $C_1(n) = O(n!)$ and $C_2(n) = O(n^3)$ which makes it really easy for us to compare the asymptotic behavior.
- if $f(n) = 100n, g(n) = n^2$ then $f = O(g)$ and $f = o(g)$.
- if $f(n) = n^2, g(n) = 100n^2 + 2n + 1$ then $f = \Theta(g)$.

Proposition 1.2: *Let $f, g : \mathbb{N} \to \mathbb{R}^+$, suppose that, then*
- *if* $\exists N \in \mathbb{N}, \forall n > N, g(n) \neq 0$ *then* $f = o(g) \Leftrightarrow \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$.
- $f = O(g) \Leftrightarrow \exists M > 0, \exists N \in \mathbb{N}, \forall n > N, \frac{f(n)}{g(n)} < M$.

notice that n usually represents some variable of the quantity of data given by the algorithm. Depending on the context, we may take it to be the amount of elements in a list, the number of digits in a number and multiple other things.

Example: Consider the following linear programming problem with $A \in \mathbb{D}^{n \times m}$ and $b \in \mathbb{D}^m$, here $\mathbb{D}$ is the set of numbers that can be represented in the decimal base with finite digits.

$$(I) \quad \begin{cases} \max\limits_{c \in \mathbb{R}^n} c^t x \\ Ax = b \end{cases}$$

we can calculate the amount of bits needed to some $\alpha \in \mathbb{D}$ with this formula $\lceil \log_2(|\alpha| + 1) \rceil$ and thus we get that the amount of bits needed to represent the problem (I) is $T = \lceil \log_2(|\alpha_1| + 1) \rceil nm + \lceil \log_2(|\alpha_2| + 1) \rceil m$ where the first term is to calculate how much the matrix A needs in bits and the second is for the vector b.