# Projeto e Análise de Algoritmos

Flávio L. C. de Moura e Rafael Monteiro Rodrigues
Departamento de Ciência da Computação
Universidade de Brasília[1]

[1]flavio@flaviomoura.info

# Capítulo 1

# Introdução

A Análise de Algoritmos consiste no estudo de técnicas que nos permitem provar a correção de algoritmos, assim como analisar a sua complexidade de tempo e espaço. Mas o que significa dizer que um algoritmo é correto? Intuitivamente, significa que o algoritmo sempre fornece respostas corretas para qualquer entrada possível. Por exemplo, se $AlgOrd$ é um algoritmo de ordenação de inteiros, então espera-se que o resultado da execução de $AlgOrd$ em um vetor $A$ qualquer, retorne uma permutação de $A$ que esteja ordenada. Isto significa que as respostas dadas pelo algoritmo são corretas para todas as entradas possíveis, e que estas respostas são geradas em tempo finito. Como veremos, em alguns casos a prova de correção pode ser simples, mas em geral esta é uma tarefa complexa.

A complexidade de tempo é outro aspecto importante a ser observado em um algoritmo, ou seja, estamos interessados em saber se o algoritmo é rápido ou não. Mas como analisar a velocidade de um algoritmo? Este é um detalhe importante e que, como veremos, não é tão simples de se responder. Uma maneira imediata de tentar responder a esta questão consiste em implementar o algoritmo em alguma linguagem de programação, e medir diretamente o tempo de execução para entradas distintas. É natural esperar que quanto maior for a entrada, mais tempo será exigido para a sua execução, de forma que o tempo de execução será uma função que depende do tamanho da entrada. No entanto, este tempo de execução pode mudar dependendo de diversos fatores relacionados a esta implementação, como o *hardware* utilizado, a linguagem de programação escolhida, e a própria implementação, já que implementações diferentes podem ter tempos de execução distintos. Adicionalmente, esta abordagem possui outras limitações:

- As medições só podem ser feitas em um número limitado de entradas;
- É difícil comparar a eficiência de dois algoritmos a menos que os experimentos tenha sido feitos no mesmo *hardware* e com o mesmo *software*.

Precisamos de uma metodologia que nos permita comparar diferentes algoritmos que resolvam um determinado problema, independentemente de implementação, i.e. uma metodologia que possa ser feita a partir de uma descrição de alto nível do algoritmo, e que portanto seja independente de *hardware* e *software*. Adicionalmente, gostaríamos que esta metodologia nos permitisse levar

em conta todas as possíveis entradas, e não apenas aquelas que são eventualmente testadas via uma implementação. Um dos objetivos deste curso é estudar as ferramentas matemáticas que nos permitirão analisar a complexidade de algoritmos. A metodologia a ser utilizada vai associar a cada algoritmo uma função $f(n)$ que caracteriza o tempo de execução em função do tamanho $n$ da entrada. Como exemplo, suponha que uma operação (de um algoritmo) possa ser realizada em 1 microssegundo ($1\mu s = 10^{-6}s$). Qual seria o tamanho máximo para um problema, que requer 400n operações, ser resolvido em 1 segundo? Temos 1 segundo para realizar 400n operações, cada uma levando $10^{-6}$ segundos, ou seja, queremos determinar $n$ na equação $400n.10^{-6} = 1$, o que resulta em 2500. Isto é, podemos resolver um problema de tamanho máximo igual a 2500 nas condições dadas. E qual o tamanho máximo do problema se o mesmo puder ser resolvido em 1 minuto? Agora, queremos determinar $n$ na equação $400n.10^{-6} = 60$, o que nos dá $n = 150000$. A tabela abaixo[1] nos mostra a importância de um algoritmo bem projetado porque um algoritmo com comportamento assintótico mais lento é batido, para valores grandes de $n$, por um algoritmo que tenha comportamento assintótico mais rápido ainda que o fator constante do algoritmo mais rápido seja pior:

| | 1 segundo | 1 minuto | 1 hora |
|---|---|---|---|
| $400n$ | 2500 | 150000 | 9000000 |
| $20n.\lg\ n$ | 4096 | 166666 | 7826087 |
| $2n^2$ | 707 | 5477 | 42426 |
| $n^4$ | 31 | 88 | 244 |
| $2^n$ | 19 | 25 | 31 |

A notação ($\lg\ n$) denota o logaritmo na base 2, isto é, $\lg\ n = \log_2\ n$. Uma ferramenta que será utilizada frequentemente nas provas de correção de algoritmos é a indução matemática. Ao analisarmos a eficiência dos algoritmos também faremos o uso de diversas ferramentas matemáticas (somatórios, conjuntos, funções, matrizes, etc). O apêndice VIII do livro [2] pode ser usado para revisar estes temas.

## 1.1 Exercício:

Complete a tabela a seguir com o tamanho máximo $n$ que um problema pode ter para ser resolvido no tempo $t$ considerando que cada operação do algoritmo que resolve o problema é executada em 1 microssegundo[2].

| | 1 seg | 1 min | 1 hora | 1 dia | 1 mês | 1 ano | 1 século |
|---|---|---|---|---|---|---|---|
| $\lg\ n$ | | | | | | | |
| $\sqrt{n}$ | | | | | | | |
| $n$ | | | | | | | |
| $n.\lg\ n$ | | | | | | | |
| $n^2$ | | | | | | | |
| $n^3$ | | | | | | | |
| $2^n$ | | | | | | | |
| $n!$ | | | | | | | |

[1] Extraída de [1]
[2] Extraído de [2]