

# Projeto e Análise de Algoritmos

Flávio L. C. de Moura

11 de maio de 2023

Os algoritmos de ordenação estudados até aqui são baseados na comparação de chaves, ou seja, a operação básica destes algoritmos é a comparação entre elementos do vetor ou lista que se quer ordenar[1]. Neste contexto, vimos que *Insertion Sort* e *Quicksort* têm, no pior caso, complexidade de tempo quadrática, *i.e.*, estão em  $O(n^2)$ . Já *Merge sort* é capaz de, no pior caso, ordenar um vetor com  $n$  elementos em tempo  $O(n \lg n)$ . A dúvida que surge naturalmente agora é: seria possível construir um algoritmo baseado na comparação de chaves que consiga ordenar um vetor com  $n$  elementos, no pior caso, realizando menos do que  $c \cdot n \lg n$  comparações (para alguma constante positiva  $c$ ? Ou seja, seria possível melhorar a cota  $O(n \lg n)$  no pior caso?

Para responder a esta pergunta, assumiremos que os  $n$  elementos  $x_1, x_2, \dots, x_n$  a serem ordenados são distintos. Além disto, o processo de ordenação será modelado por *árvores de decisão*, que são árvores binárias completas que representam as comparações realizadas entre elementos por um algoritmo de ordenação particular em uma entrada de tamanho dado. Assim, para construirmos a árvore de decisão para um algoritmo  $A$  (baseado na comparação de chaves) que recebe como entrada um vetor com  $n$  elementos distintos, anotaremos seus nós internos com  $i : j$ , para  $1 \leq i, j \leq n$ , e cada folha com a permutação  $\langle \pi(1), \pi(2), \dots, \pi(n) \rangle$ . A execução do algoritmo corresponde a um caminho da raiz até uma folha. Cada nó interno corresponde a comparação  $a_i \leq a_j$  (a rigor  $a_i < a_j$  já que os elementos são distintos), e a subárvore à esquerda indica as comparações subsequentes. A subárvore à direita indica as comparações subsequentes quando  $a_i > a_j$ , e uma folha indica que o algoritmo ordenou o vetor de forma que  $a_{\pi(1)} \leq a_{\pi(2)} \leq \dots \leq a_{\pi(n)}$ . Como qualquer algoritmo de ordenação correto tem que ser capaz de produzir todas as permutações de uma entrada, cada uma das  $n!$  possíveis permutações do vetor de tamanho  $n$  dado com entrada tem que aparecer como uma folha. O caminho mais longo da raiz de uma árvore de decisão até uma folha representa o número de comparações no pior caso do algoritmo em consideração. Ou seja, o número de comparações no pior caso de um algoritmo corresponde a altura de sua árvore de decisão. A cota inferior da altura de todas as possíveis árvores de decisão será então a cota inferior, no pior caso, para qualquer algoritmo de ordenação baseado na comparação de chaves.

**Teorema 1.** *Qualquer algoritmo baseado na comparação de chaves requer  $\Omega(n \lg n)$  comparações no pior caso.*

*Prova.* Precisamos determinar a altura de uma árvore de decisão onde cada permutação do vetor de entrada apareça como uma folha. Considere a árvore de decisão de altura  $h$  contendo  $l$  folhas de um algoritmo para ordenar  $n$  elementos distintos. Como cada uma das  $n!$  permutações do vetor de entrada aparece como uma folha, temos que  $n! \leq l$ , e como uma árvore binária de altura  $h$  não possui mais do que  $2^h$  folhas, temos que  $n! \leq l \leq 2^h$ . Portanto  $h \geq \lg(n!) = \Omega(n \lg n)$ .  $\square$

**Exercício 2.** *Sejam  $l$  o número de folhas em uma árvore binária, e  $h$  sua altura. Prove que  $l \leq 2^h$ .*

## Referências

- [1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.