

# Limitante Inferior para o Problema da Ordenação

André Vignatti

## Quão rápido podemos ordenar?

Iremos provar um limitante inferior para esse problema.

Já estudamos alguns algoritmos de ordenação.

- Eles têm algo em comum: a única operação que usam para definir a posição relativa dos elementos são **comparações**.
- Isto é, o resultado de comparar  $x_i$  com  $x_j$ , define se  $x_i$  será posicionado **antes ou depois** de  $x_j$ .

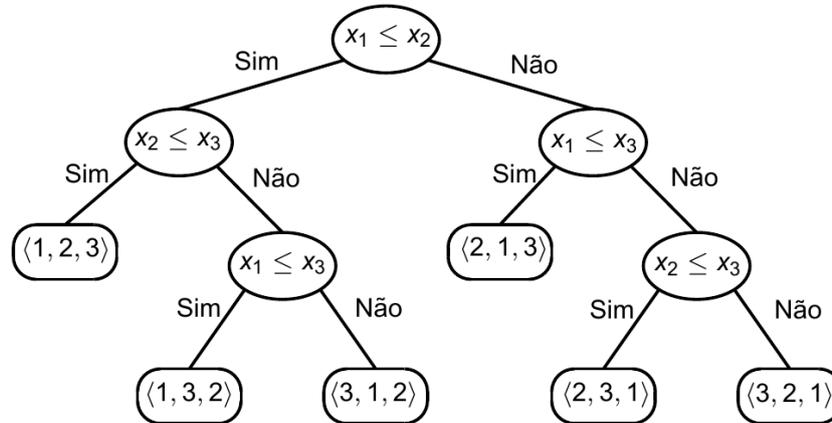
Limitantes Inferiores para a Ordenação:

- $\Omega(n)$  para examinar toda entrada.
- Mas todos os algoritmos vistos levam  $\Omega(n \log n)$ .
- Iremos mostrar que  $\Omega(n \log n)$  é um limitante inferior para ordenação baseada somente em comparações.

Árvore de Decisão:

- Abstração de qualquer ordenação baseada em comparações.
- Representa comparações feitas por
  - um dado algoritmo de ordenação
  - em entradas de um certo tamanho
- Abstrai todo o resto: controle e movimento de dados.
- Iremos contar **somente** comparações.

Figura 1: Árvore do comportamento do *Insertion Sort* para 3 elementos



- Nós representam **comparações** entre dois elementos, digamos  $x_i \leq x_j$ .
- Ramificações representam os possíveis resultados da comparação: **SIM** se  $x_i \leq x_j$ , ou **NÃO** caso contrário.
- Folhas representam **possíveis soluções**: as diferentes permutações dos  $n$  índices.

Quantas folhas existem na árvore de decisão? Há  $\geq n!$  folhas, pois cada permutação aparece pelo menos uma vez.

Para qualquer algoritmo de ordenação:

- Há uma árvore diferente.
- A árvore modela todas os possíveis caminhos de execução.

O caminho mais comprido da raiz a uma folha representa o **pior caso**. Ele depende do algoritmo. No InsertionSort é  $\Theta(n^2)$ , No MergeSort é  $\Theta(n \log n)$ .

**Lema.** Qualquer árvore binária de altura  $h$  tem no máximo  $2^h$  folhas.

*Demonstração.* (indução em  $h$ )

Base:  $h = 0$ . A árvore é só um nó, que é folha.  $2^h = 1$ .

Hipótese: Assuma que o resultado é verdade para altura  $h - 1$ .

Passo: Na árvore de altura  $h - 1$ , aumente a altura em 1 criando o maior número de novas folhas possíveis. Cada folha vira pai de duas novas folhas.

$$\begin{aligned}
\# \text{ de folhas para altura } h &= 2 \cdot (\# \text{ de folhas para altura } h - 1) \\
(\text{hip.}) &= 2 \cdot 2^{h-1} \\
&= 2^h.
\end{aligned}$$

□

**Teorema.** Qualquer árvore de decisão que ordena  $n$  elementos tem altura  $\Omega(n \log n)$ .

*Demonstração.* Seja  $l$  o número de folhas da árvore de decisão.

- $l \geq n!$
- Pelo Lema,  $n! \leq l \leq 2^h$ , ou seja,  $2^h \geq n!$
- Tirando logs,  $h \geq \log_2 n!$ .
- Mas,

$$\begin{aligned}
\log_2 n! &= \sum_{i=1}^n \log i \\
&\geq \sum_{i=\lceil n/2 \rceil}^n \log i \\
&\geq \sum_{i=\lceil n/2 \rceil}^n \log n/2 \\
&\geq (n/2 - 1) \log n/2 \\
&= n/2 \log n - n/2 - \log n + 1 \\
&\geq n/4 \log n, \text{ para } n \geq 16.
\end{aligned}$$

- Então,  $h \in \Omega(n \log n)$ .

□

Consequência do Teorema:

- O melhor algoritmo baseado em comparações será  $\Omega(n \log n)$  no pior caso.
- O MergeSort é um algoritmo ótimo.

Cap 8 do CLRS têm o título “Ordenação em Tempo Linear”. Como isso é possível? O livro está errado?

Algoritmos de ordenação baseados **não somente** em comparações devem **assumir alguma propriedade sobre a entrada**:

- COUNTING SORT: assume que cada um dos  $n$  elementos é um inteiro entre 0 e  $k \Rightarrow \Theta(n + k)$ .
- RADIX SORT: assume que cada dígito dos  $n$  elementos assume  $d$  valores distintos  $\Rightarrow \Theta(d(n + k))$ , com  $k =$  número de elementos distintos.
- BUCKET SORT: assume que os  $n$  números estão distribuídos de acordo com a **distribuição uniforme**  $\Rightarrow$  tempo **esperado**  $\Theta(n)$ .

Uma diferença crucial:

- Até agora no curso: **análise de algoritmo**.
- Hoje: **análise de problema**.

Esse resultado é raro.