Sim.

E isso nos leva à próxima pergunta

- *Será que existe um algoritmo de ordenação que executa em tempo menor que* $O(n \log n)$ *?*

E a resposta é: *Não!*

Só que, agora é preciso explicar porque.

Quer dizer, a gente vai argumentar que

→ *Não existe algoritmo de ordenação baseado em comparações que executa em tempo menor que* $O(n \log n)$

Porque?

Bom, porque a comparação de dois elementos gera muito pouca informação.

Daí que, é preciso fazer um monte de comparações para ordenar a lista completamente.

Daí que, qualquer algoritmo de ordenação baseado em comparações vai levar um boca de tempo.

Certo, isso faz sentido.

Mas, porque $O(n \log n)$ *?*

Bom, fixe o tamanho n da lista.

E imagine que a lista contém os elementos 1, 2, 3, ..., n — (*em uma ordem qualquer*)

Quer dizer, os valores dos elementos realmente não importam.

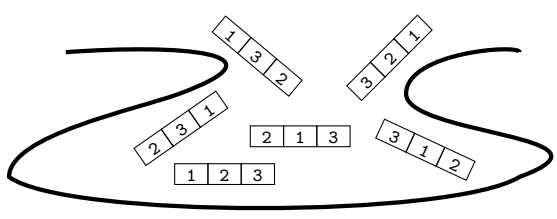
Porque a única coisa que o algoritmo faz é comparar pares de elementos.

Certo.

Antes da coisa começar, o algoritmo ainda não sabe nada sobre a ordem em que os elementos estão

— (*e por isso, ele não pode fazer nada ainda ...*)

A gente representa esse fato como um saco, onde se encontram todas as configurações possíveis da lista



Note que cada configuração da lista corresponde a uma permutação dos números de 1 a n.

Logo, existem $n!$ configurações no saco.

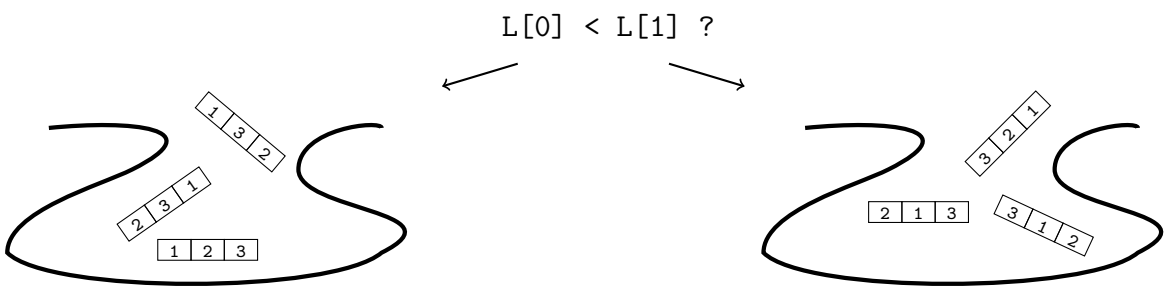
Legal.

Agora, imagine que o algoritmo compara os dois primeiros elementos da lista.

O que acontece?

Bom, isso depende do resultado da comparação.

E abaixo, nós temos os dois resultados possíveis



Quer dizer, a coisa importante é que o número de configurações dentro do saco foi reduzido à metade — (*porque?*)

E agora, a gente pode imaginar que uma nova comparação reduz o número de configurações à metade outra vez.

Bom, nem sempre é isso o que acontece — (*porque?*)

Mas, isso é o que acontece no melhor caso — (*porque?*)

E agora, a gente pode pensar sobre o que acontece quando todas as comparações correspondem ao melhor caso.

Bom, o que acontece é que o saco vai sendo reduzido sucessivamente à metade.

Até que reste apenas uma configuração lá dentro.

Nesse ponto, o algoritmo já sabe exatamente qual é a configuração em que os elementos se encontram.

E daí, ele pode aplicar as trocas necessárias para ordenar a lista completamente

— (*sem a necessidade de nenhuma comparação adicional ...*)

Certo.

Mas, quantas comparações são necessárias para chegar nesse ponto?

Bom, a resposta é

$$\log_2 n!$$

Sim, mas quanto é isso?

Bom, para descobrir isso, a gente reescreve a coisa assim

$$\log_2 \Big(1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n \Big)$$

e depois assim

$$\log_2 1 + \log_2 2 + \log_2 3 + \ldots + \log_2 n$$

E agora?

Bom, agora é a hora para uma pequena esperteza.

Quer dizer, primeiro a gente joga metade dos termos fora

$$\log_2 1 + \cancel{\log_2 2 + \log_2 3} + \ldots + \log_2 \tfrac{n}{2} + \ldots + \log_2 n$$

Depois, a gente observa que isso é maior que

$$\log_2 \frac{n}{2} + \log_2 \frac{n}{2} + \ldots + \log_2 \frac{n}{2} \qquad \text{(n/2 vezes)}$$

E depois, a gente nota que

$$\frac{n}{2} \cdot \log_2 \frac{n}{2} \quad = \quad O(n \log n)$$

Quer dizer, para ordenar a lista completamente, é preciso fazer ao menos $\frac{n}{2} \cdot \log n$ comparações.

E isso leva tempo $O(n \log n)$.

Legal, não é?