Agora, nós vamos entender melhor as coisas.

E ao fazer isso, nós vamos resolver todos os nossos problemas — (*pelo menos no que diz respeito à ordenação de listas ...*)

Quer dizer, a gente começa perguntando

- *Aonde está o ganho de eficiência do Mergesort ?*

Bom, na realidade são duas coisas.

A primeira delas é que o algoritmo nunca compara o mesmo par de elementos duas vezes.

Porque, quando a comparação acontece, os elementos são colocados em uma lista ordenada.

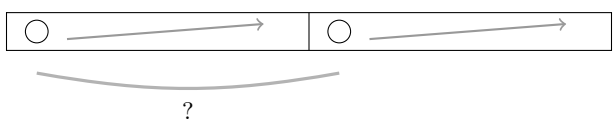
E os elementos dessa lista nunca são comparados outra vez — (*porque não precisa ...*)

Mas, isso não basta para tornar um algoritmo de ordenação eficiente — (*porque?*)

Daí vem a segunda coisa.

E a segunda coisa é a transitividade.

Quer dizer, quando a gente compara os primeiros elementos de duas listas ordenadas, e descobre quem é o menor



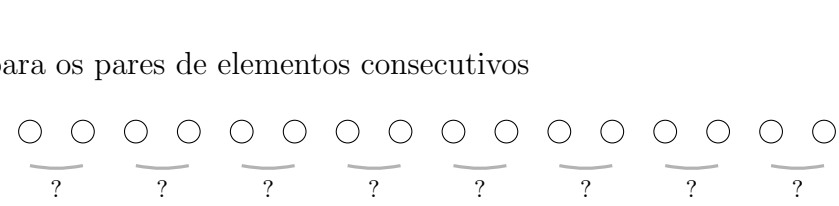
a gente ganha de graça o fato de que ele é menor do que todos os elementos da outra lista — (*e isso é um monte de comparações que a gente não precisa fazer*)

Legal.

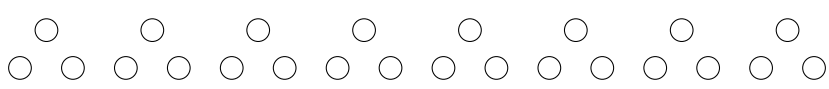
Agora que a gente entendeu, nós podemos considerar o problema da ordenação outra vez.

Quer dizer, imagine que os elementos da lista estão um ao lado do outro — (*em uma fila*)

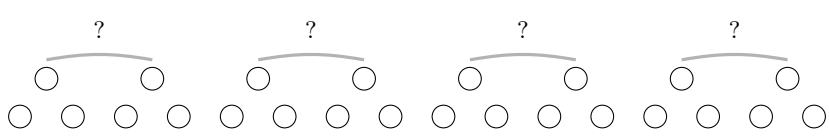
Daí, a gente compara os pares de elementos consecutivos



E daí, como a gente não quer fazer essas comparações outra vez, a gente copia o menor de cada par para cima

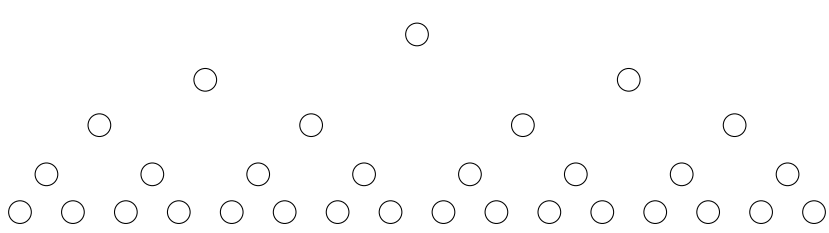


Daí, o próximo passo é comparar os pares de elementos consecutivos que foram copiados para cima



Já deu pra ver o que está acontecendo, não é?

Quer dizer, a gente está construindo uma estrutura de árvore, onde o menor elemento fica no topo

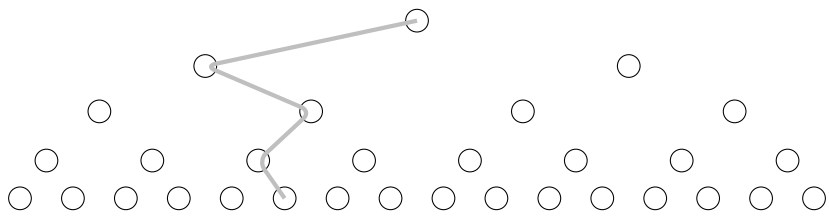


E nesse ponto a gente pergunta

- *Como é que a gente acha o segundo maior ?*

Bom, é fácil

- basta percorrer outra vez o caminho que foi feito pelo menor
- e encontrar o menor elemento que foi comparado com ele



E daí vem uma esperteza.

Quer dizer, a ideia é apagar o caminho do segundo maior — (*fazendo os elementos que foram comparados com ele subir no caminho*)

E fazer o segundo menor percorrer o caminho do maior para chegar ao topo.

Porque daí, a gente está na mesma situação outra vez.

E pode encontrar o terceiro maior, o quarto maior, etc ...

Quer dizer, aqui a gente tem mais um algoritmo de ordenação.

E daí, a gente pergunta

- *Qual é o tempo de execução desse algoritmo ?*

Bom, a primeira etapa leva tempo

$$\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \ldots + 2 + 1 = O(n)$$

Na segunda etapa, a gente precisa descer e subir na árvore para obter o próximo menor elemento — (*e ajustar os caminhos*)

Logo, a coisa toda leva tempo

$$O(n) + O(n \log n) = O(n \log n)$$

E aqui nós temos mais um algoritmo eficiente de ordenação.

Mas, a gente não ganhou nada.

Porque esse algoritmo precisa de memória adicional para armazenar a árvore — (*e a lista auxiliar que recebe os menores elementos, como o Mergesort*)

E agora?