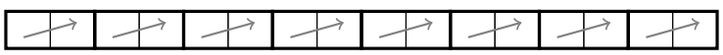
Bom, agora é a hora para mais uma esperteza.

Quer dizer, a ideia é armazenar a árvore dentro da própria lista.

E para isso, a gente relembra a intuição inicial do algoritmo

→ *organizar as coisas na memória para lembrar os resultados das comparações*

Outra maneira de fazer isso é comparar pares de elementos consecutivos, e mover o menor para a esquerda

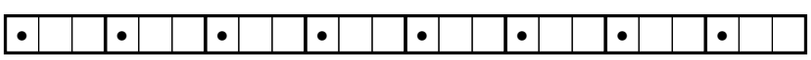


como o Mergesort.

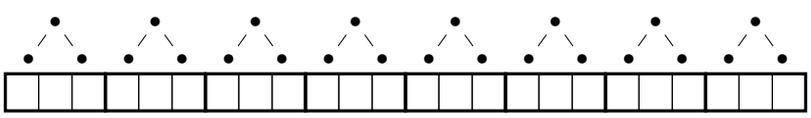
Mas, não é por aí que a gente vai ...

Quer dizer, a ideia é dividir a lista em grupinhos de 3.

Daí, a gente move o menor dos 3 para a primeira posição do grupo

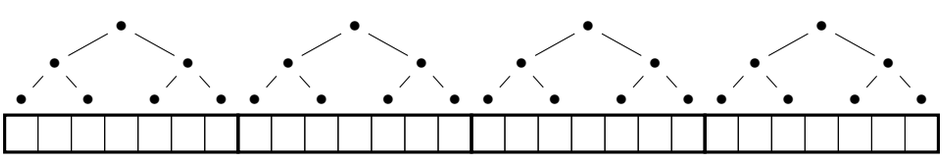


E daí, a gente visualiza (ou imagina) a coisa assim

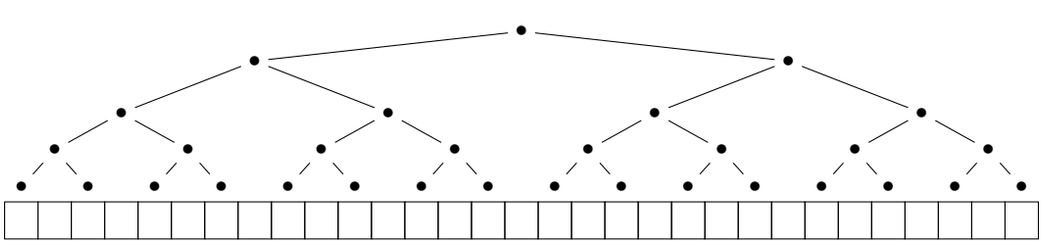


Sim! são arvorezinhas.

Trabalhando com grupos de 7 a gente obtém árvores maiores



E continuando assim, a gente monta uma árvore completa



Ou melhor, isso é um heapMin — (*onde cada elemento é menor do que os seus dois filhos*)

Só que a gente também pode pensar que isso é o heap-Max — (*porque é a mesma coisa*)

Daí, a ideia é fazer a remoção do elemento no topo do heap — (*o maior elemento da lista*)

E colocá-lo na última posição da lista — (*que foi liberada pela remoção*)

Daí, a gente remove o novo elemento do topo — (*o segundo maior elemento da lista*)

E colocá-lo na penúltima posição da lista.

Mas, para a coisa funcionar, é preciso utilizar o esquema tradicional de indexação do heap

— (*onde os filhos do elemento na posição $k$ ficam nas posições $2k$ e $2k+1$*)

Dessa maneira, a gente ordena a lista em tempo $O(n \log n)$ — (*no pior caso*)

Sem a necessidade de usar uma lista auxiliar.