A bagunça organizada

Sim, mas esse esquema ainda tem dois problemas.

Quer dizer, a gente ainda está utilizando a lista auxliar para produzir a lista ordenada.

E o outro problema é que esse esquema ainda não é muito flexível.

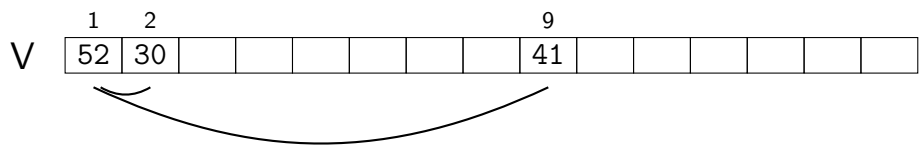
Quer dizer, ele funciona bem quando a lista tem tamanho $2^k - 1$ — (*porque?*)

Isso nem sempre isso é o caso.

Mas a gente sempre pode aumentar o tamanho da lista se for preciso.

O problema maior é o seguinte.

Considere o primeiro elemento (que fica no topo da árvore), e os dois elementos que ficam abaixo dele



O ponto é que o segundo elemento se encontra aproximadamente na metade da lista.

Só que o tamanho da lista pode mudar — (*se ocorrerem inserções ou remoções*)

Mas se o tamanho da lista mudar, nós perdemos a estrutura de árvore que foi construída.

Em resumo, nós construímos uma estrutura de dados dentro de uma estrutura de dados — (*uma árvore dentro de um vetor*)

Mas, a nossa estrutura de dados não suporta as operações de inserção e remoção.

O que significa que isso é uma estrutura de dados muito pouco flexível.

E agora?

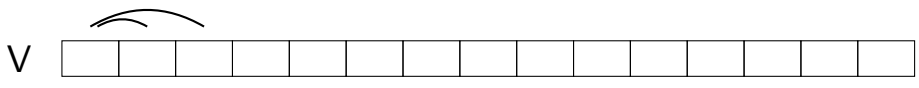
Bom, agora nós temos um problema.

E quando a gente tem um problema, a gente procura uma solução.

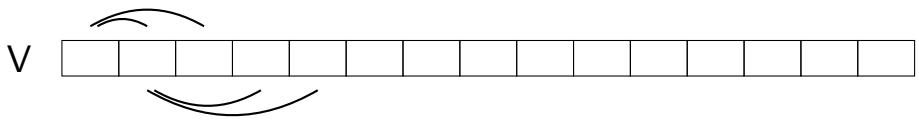
A boa notícia é que nesse caso a solução está ao alcance da mão.

Quer dizer,

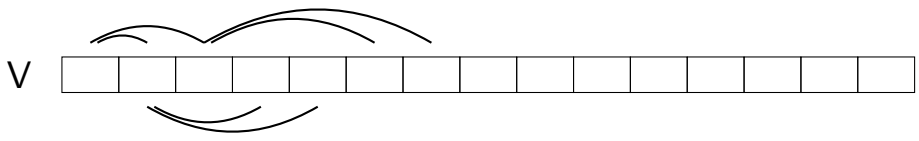
- abaixo do primeiro elemento, nós temos os dois próximos elementos da lista



- abaixo do segundo elemento, nós temos os dois elementos seguintes

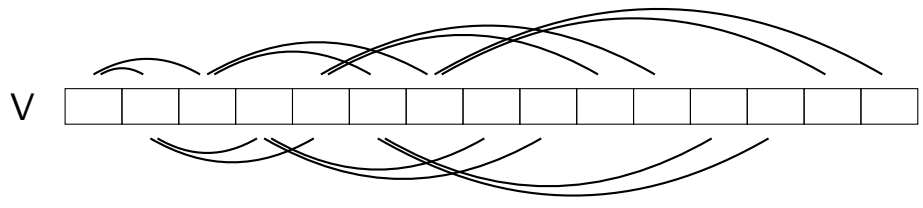


- abaixo do terceiro elemento, nós temos os dois elementos seguintes



- e por aí vai ...

Fazendo isso até o fim, a gente obtém o seguinte



Qual é a regra?

Bom, aqui é bem fácil de ver

→ *abaixo do elemento na posição* `k`

estão os elementos nas posições `2k` *e* `2k + 1`

Legal, não é?