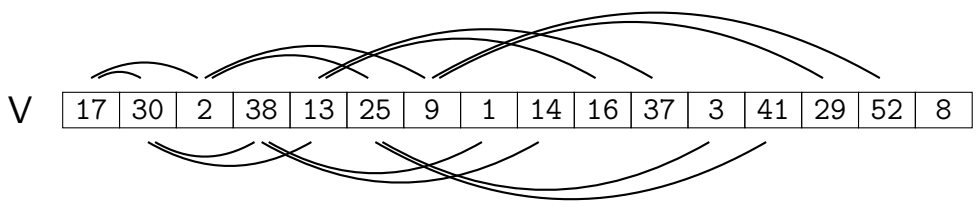
## A construção da árvore

Sim é legal, mas a gente ainda não acabou.

Quer dizer, quando tudo começa a lista está toda desorganizada

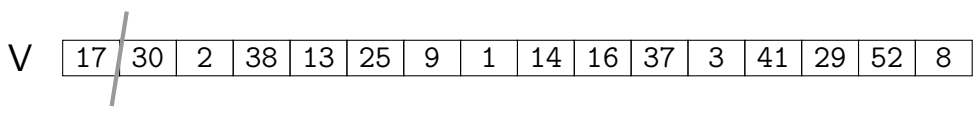



e os elementos não precisam ser maiores do que aqueles que estão abaixo deles na árvore.

*E agora?*

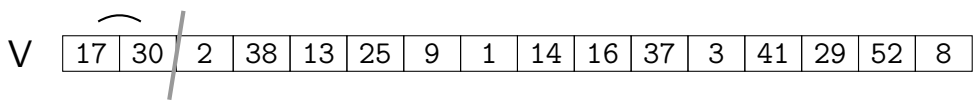
Bom, agora é a hora para uma esperteza.

E a esperteza consiste em imaginar que a lista tem tamanho 1 — (*ignorando os demais elementos*)




Daí, uma lista de tamanho 1 sempre está organizada — (*porque?*)

E o primeiro passo consiste em imaginar que o tamanho da lista vai aumentar para 2

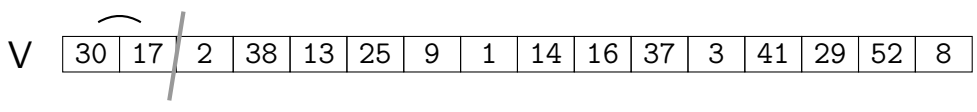



Quer dizer, isso aconteceu porque o 30 entrou na lista.

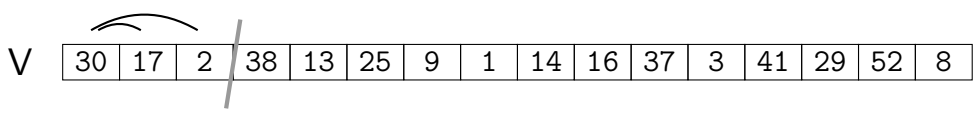
E daí, ele precisa ser menor do que o elemento que está acima dele.

Só que ele não é.

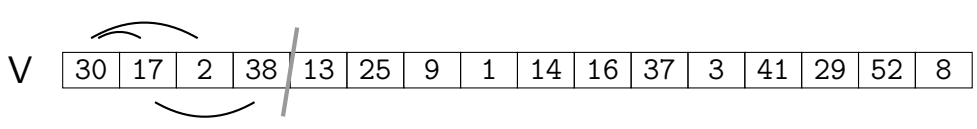
Então, a gente troca os dois de lugar



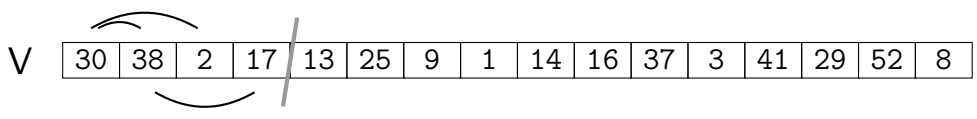

A seguir, quem entra é o 2, e dessa vez não acontece nada




Daí, é a vez do 38

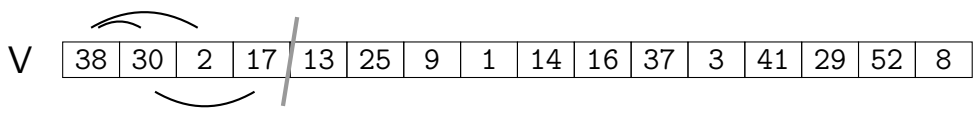



Como o 38 é maior do que o elemento acima dele, os dois trocam de lugar




Só que após a troca, o 38 continua sendo maior do que o elemento acima dele.

Então, ocorre uma nova troca




Quer dizer, um novo elemento que chega na lista pode chegar até o topo — (*por meio de trocas sucessivas*)

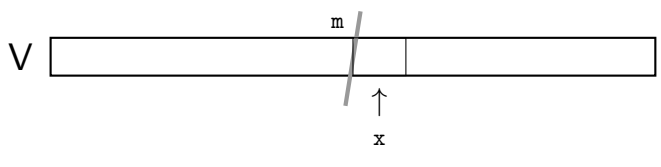
E a construção da árvore pode ser feita por meio de inserções sucessivas.

Certo.

Agora que a gente entendeu, está na hora de escrever o algoritmo.

E a gente começa com a função de inserção

Imagine que a árvore já foi construída até a posição `m`, e que agora nós vamos inserir mais um elemento




```
func  Insercao_HeapMax ( x, V, m )
     m++;    V[m] = x                 // colocando o x na lista
     i = m                            // colocando o dedo sobre ele
     Enquanto ( i > 1 )               // e levando ele para o lugar certo
        j = i/2                       // a posição de cima
        Se ( V[i] > V[j] )
           Troca ( V, i, j );    i = j
        Senão         Quebra
```

E agora que a gente tem a função de inserção, é fácil realizar a construção da árvore

```
func  Construção_Árvore ( V, 1, N )
     m = 1
     Para  i = 2  Até  N
        Inserção_HeapMax ( V[i], V, m )
```

**Análise**

Como a árvore tem altura $\log_n n$, o laço da função de inserção dá no máximo $\log_2 n$.

Logo, essa função executa em tempo $O(\log n)$.

E agora, é fácil ver que a construção da árvore leva tempo

$$O(n \log n)$$