# Heap

É um tipo particular de árvore binária. É baseada na estrutura do algoritmo de Heapsort e é muito importante na fila de prioridades.

O Heap possuí características bem definidas:

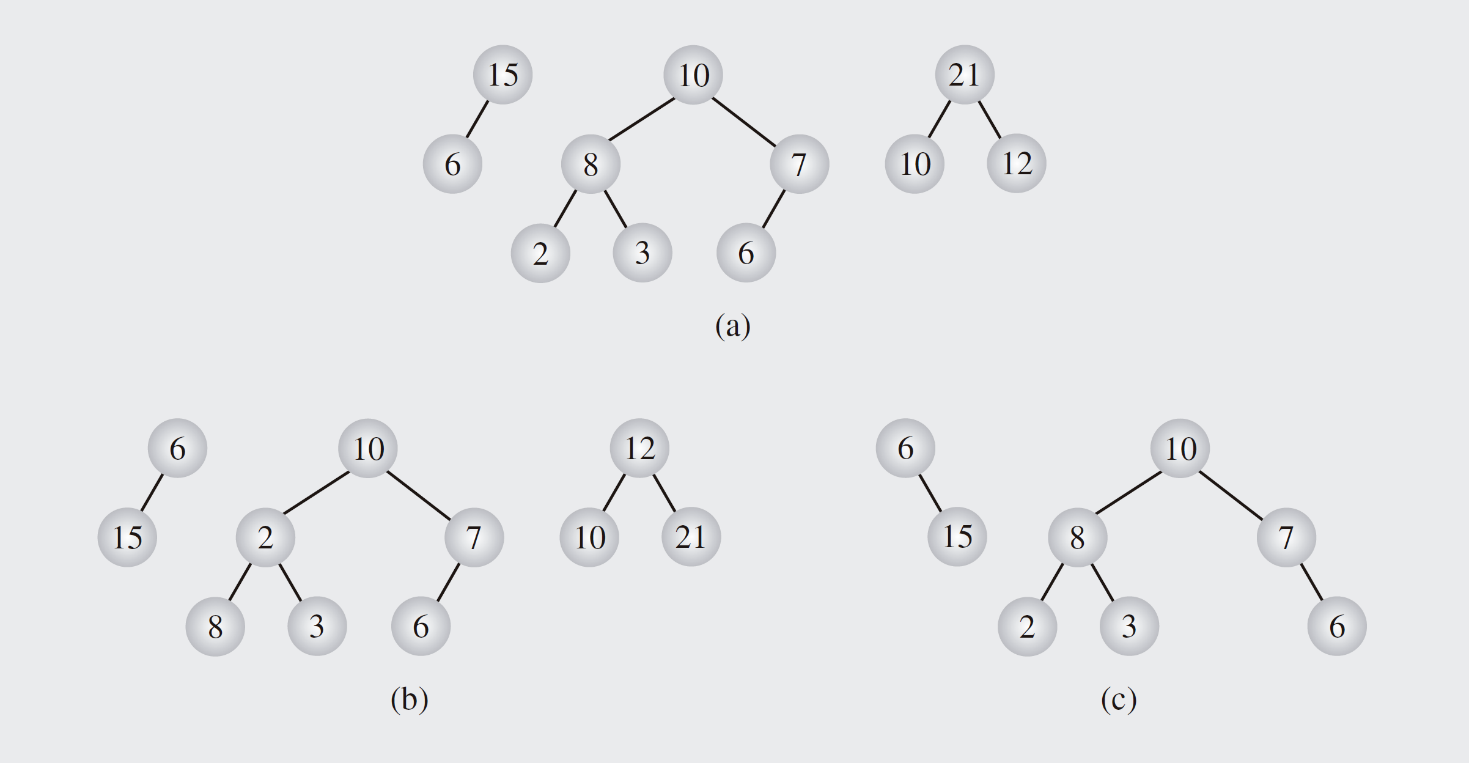
**1.** O valor de cada nó não é menor do que os valores armazenados em cada um dos seus filhos.

**2.** A árvore é perfeitamente equilibrada e as folhas no último nível estão todas nas posições mais à esquerda.

As duas propriedades acima definem uma *heap* máxima. Para definir uma *heap* mínima deve-se modificar **“maior” para “menor”** na primeira propriedade. Isto significa que a raiz de uma heap máxima contém o maior elemento, enquanto a raiz de uma heap mínima, o menor. Uma árvore tem propriedade heap se cada não folha tem a primeira propriedade. Devido à segunda condição, o número de níveis na árvore é O(lg n).

Outra característica de heap é que seus elementos não são perfeitamente ordenados. A ordem dos elementos obedece a uma linha linear de descendentes, desprezando-se as linhas laterais.

Abaixo vemos um exemplo:

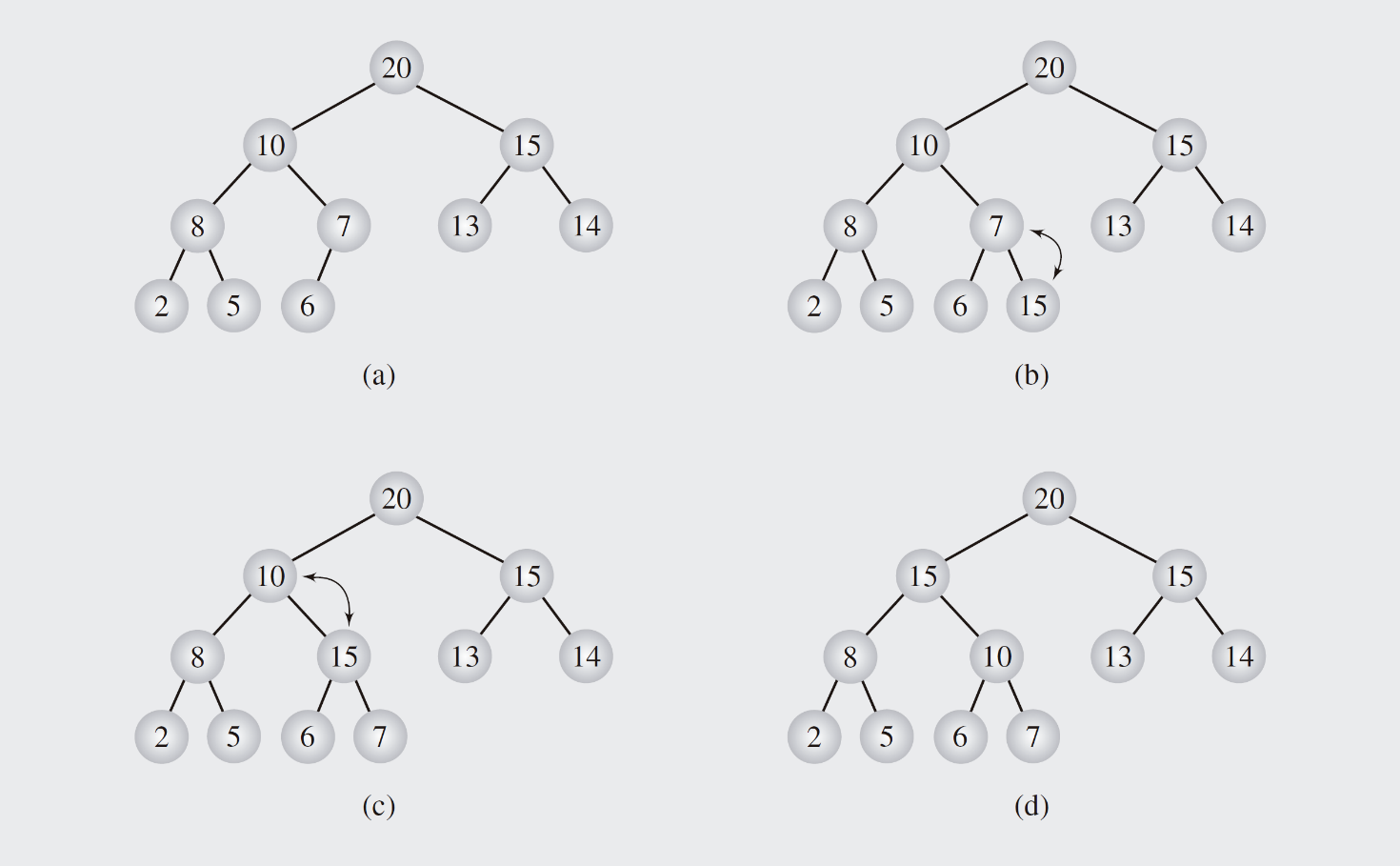


Nessa figura, as árvores (a) são heaps, as (b) violam a primeira propriedade e a (c) viola a segunda propriedade.

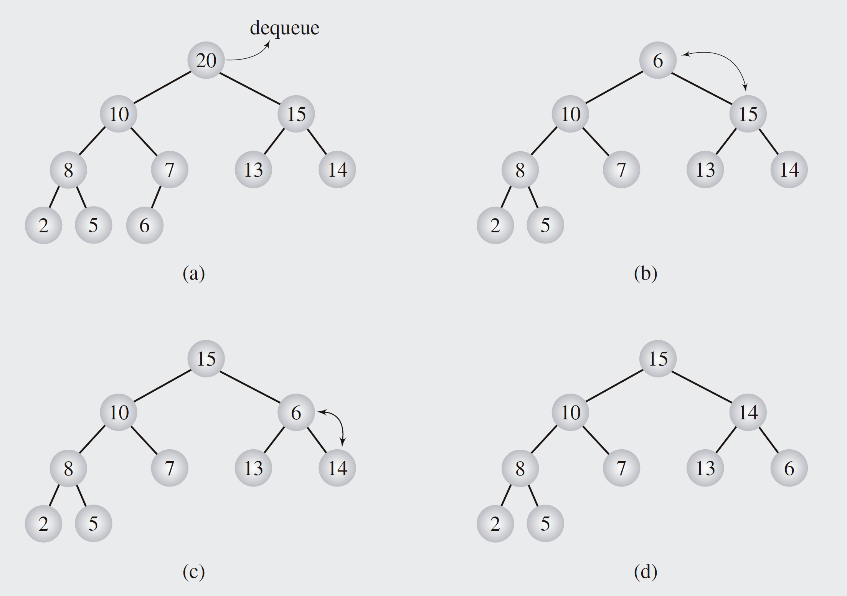
## Heap com filas com prioridades

Heap é um modo muito bom para implementar filas pois é uma árvore perfeitamente equilibrada. Também podem ser usadas para implementar filas com prioridade, para isso, dois procedimentos devem ser implementados para adicionar na fila e remover da fila com prioridade.

Para colocar um elemento na fila, ele é adicionado no final da heap. A restauração da propriedade é obtida movendo-se a última folha em direção a raiz. Abaixo um exemplo:



Tirar da fila um elemento heap consiste em remove-lo da raiz da heap, pois, pela propriedade heap, é aquele com maior prioridade. Então a última folha é colocada em seu lugar e a propriedade de heap quase certamente tem que ser restaurada, desta vez movendo da raiz para baixo da árvora. Abaixo um exemplo:

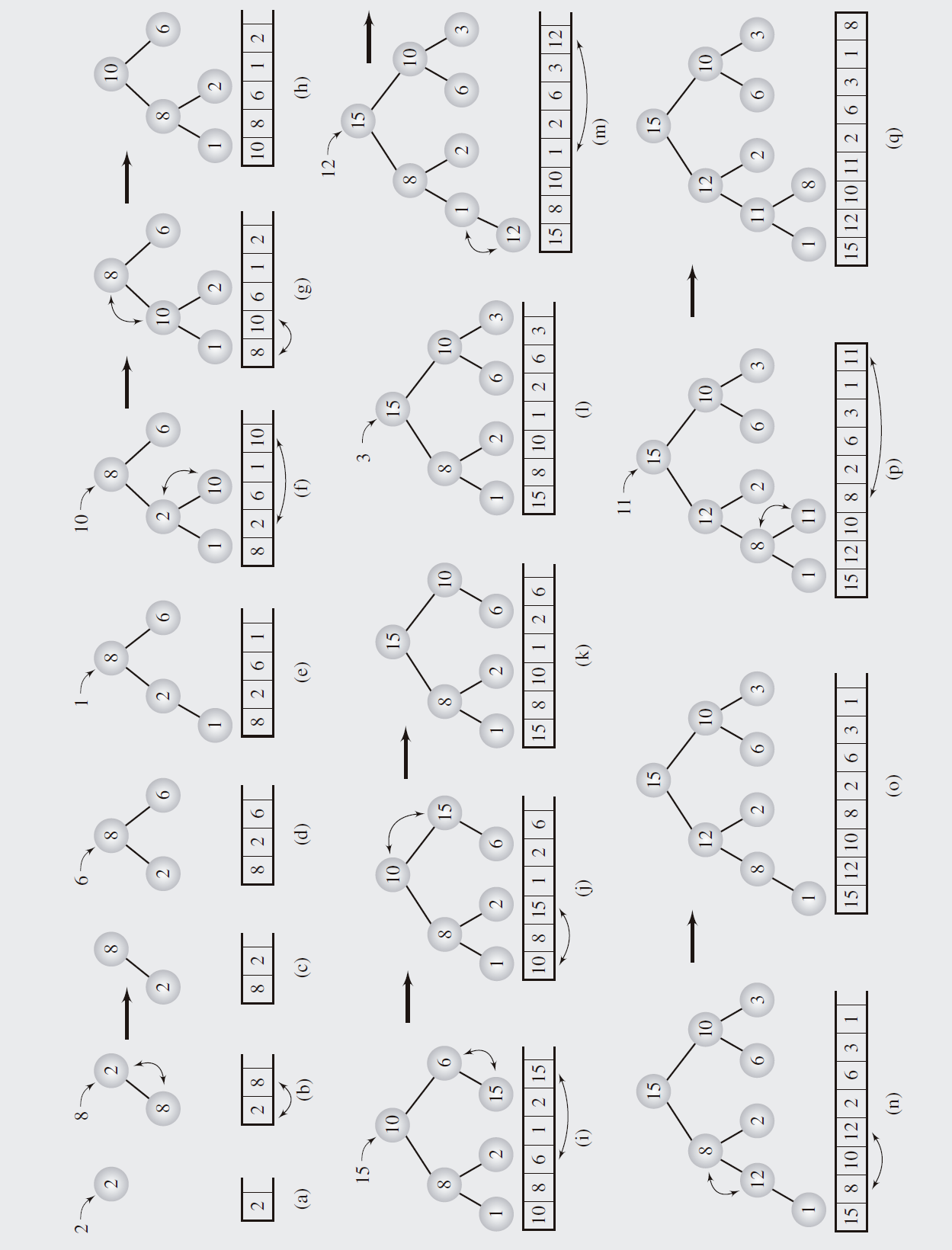


## Organizando matrizes com heaps

Heaps podem ser implementadas como matrizes, dessa forma, cada heap é uma matriz mas nem todas as matrizes são heaps. Em algumas situações , como por exemplo na ordenação de heap, precisamos converter uma matriz em uma heap, ou seja, reorganizar os dados na matriz de modo que o resultado represente uma heap. Existem vários métodos de fazer isso, o mais simples é iniciar com uma heap vazia e sequencialmente incluir elementos em uma heap crescente.

## Análise dos Exemplos

Mostra um exemplo completo do método de cima para baixo de inserção. Primeiro o número 2 é colocado em uma heap inicialmente vazia (a). A seguir, 8 é colocado no final da heap corrente (b), e então trocado com seu ascendendte (c). Colocar o terceiro e o quarto elementos, 6 (d) e então 1 (e), não necessita de troca. Colocar o quinto elemento, 10, equivale a coloca-lo no final da heap (f), então troca-lo com seu ascendente, 2 (g), e com seu novo ascendente, 8 (h), de modo que no final, 10 percorre até a raiz da heap. Colocar o sexto elemento, 15, no final da heap (i), então trocalo com seu acendente até chegar ao topo (j e k). Colocar o sétimo elemento, 3 (l), no final da heap. Adicionar o oitavo elemento, 12 (m), no final da heap, trocar com seu ascendente 1 (n), trocar com seu acendente 8 (2.57o). Adicionar o nono elemento 11 (p), trocar com seu ascendente 8 (q).



**A figura abaixo** Mostra a transformação da matriz data[]=[2 8 6 1 10 15 3 12 11] em uma heap.

Analisando um nó e usando a fórumula [n/2-1], onde n é o tamanho da fila. Comparando o valor do resultado com um dos filhos, caso o nó seja menor que um de seus filhos deverá ser trocado com o maior filho, isso acontece na figura 6.58a. Em seguida uma nova árvore é criada, dessa vez usamos a fórmula [n/2-2] para comparar o nó e seus filhos, caso o nó seja menor deverá ser trocado e isso acontece na figura 6.58b. Agora usamos a fómula [n/2-3] para comparar o nó e seus filhos, caso o nó seja menor deverá ser trocado e isso acontece na c.

