


Unidade IV:

Ordenação Interna - Heapsort



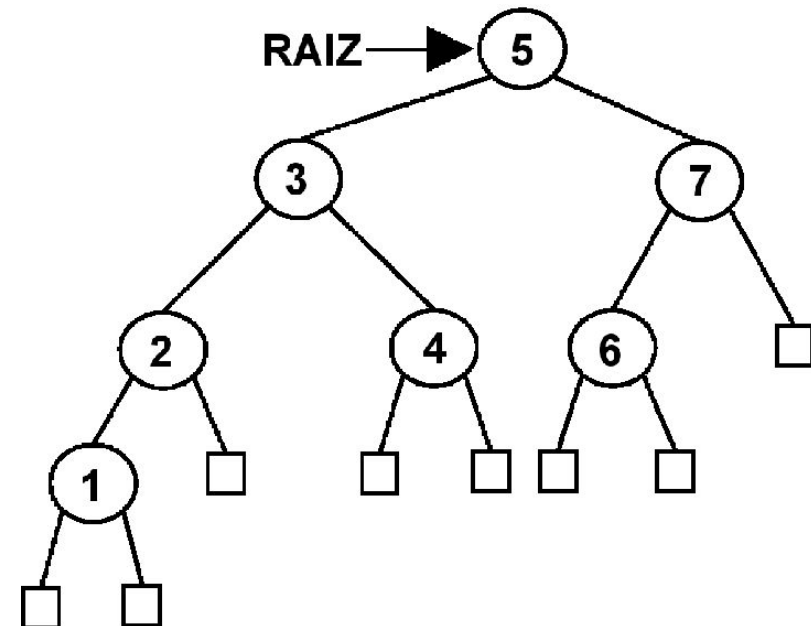
PUC Minas

Instituto de Ciências Exatas e Informática
Departamento de Ciência da Computação

- **Definição de Heap** 
- Funcionamento básico
- Algoritmo em *C like*
- Análise dos número de comparações e movimentações

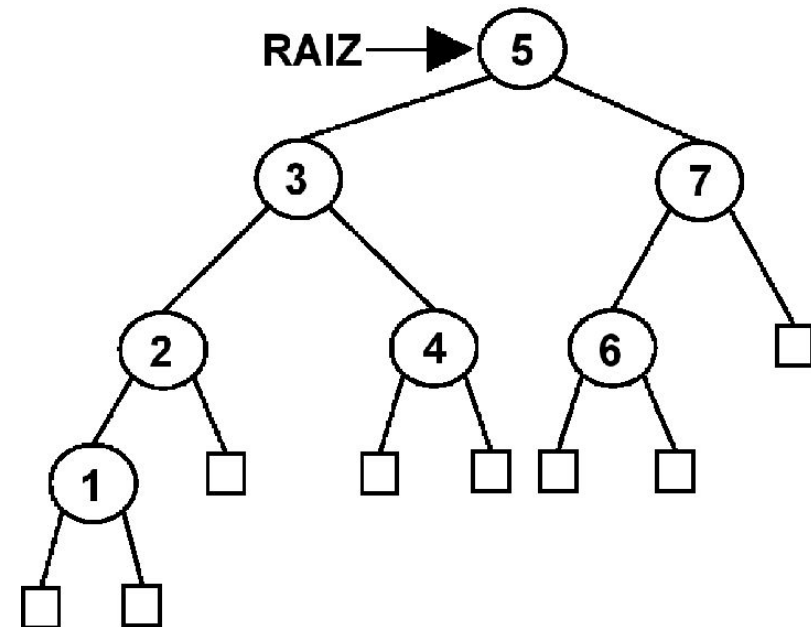
- O Heapsort é **um algoritmo de seleção** que encontra o maior elemento em uma lista, troca-o com o último e repete o processo
- Sua diferença em relação ao Algoritmo de Seleção é que o Heapsort utiliza um Heap Invertido para selecionar o maior elemento de forma eficiente
- Neste momento, precisamos conhecer os conceitos de árvore e heap

- Estrutura de dados cujas operações de inserção, remoção e substituição possuem a mesma eficiência
- Estrutura de dados que contém um conjunto finito de vértices (nós) e outro de arcos (arestas) que conectam os vértices



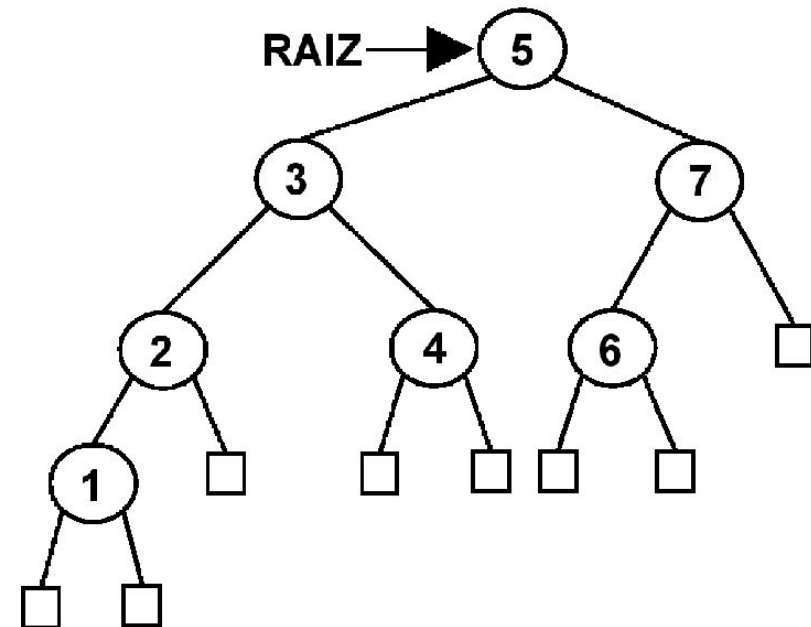
- Estrutura de dados cujas operações de inserção, remoção e substituição possuem a mesma eficiência
- Estrutura de dados que contém um conjunto finito de vértices (nós) e outro de arcos (arestas) que conectam os vértices

O nó 5 é denominado nó raiz e ele está no nível 0



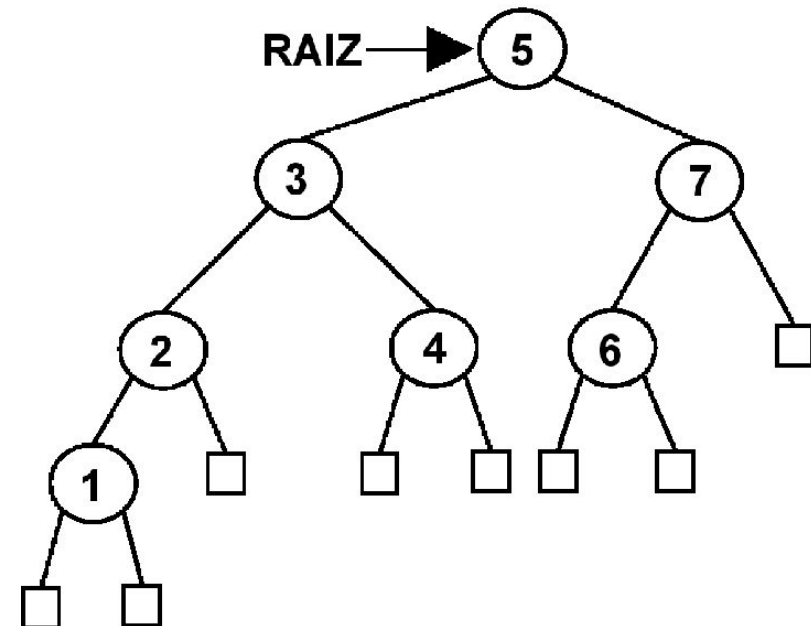
- Estrutura de dados cujas operações de inserção, remoção e substituição possuem a mesma eficiência
- Estrutura de dados que contém um conjunto finito de vértices (nós) e outro de arcos (arestas) que conectam os vértices

Os nós 3 e 7 são filhos do 5 e esse é pai dos dois primeiros



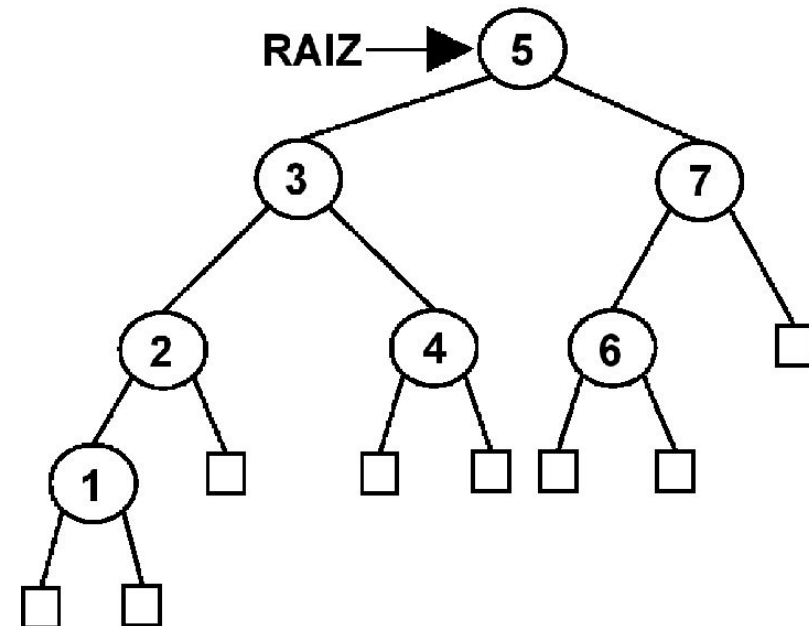
- Estrutura de dados cujas operações de inserção, remoção e substituição possuem a mesma eficiência
- Estrutura de dados que contém um conjunto finito de vértices (nós) e outro de arcos (arestas) que conectam os vértices

Um nó com filho(s) é chamado de nó interno e outro sem, de folha

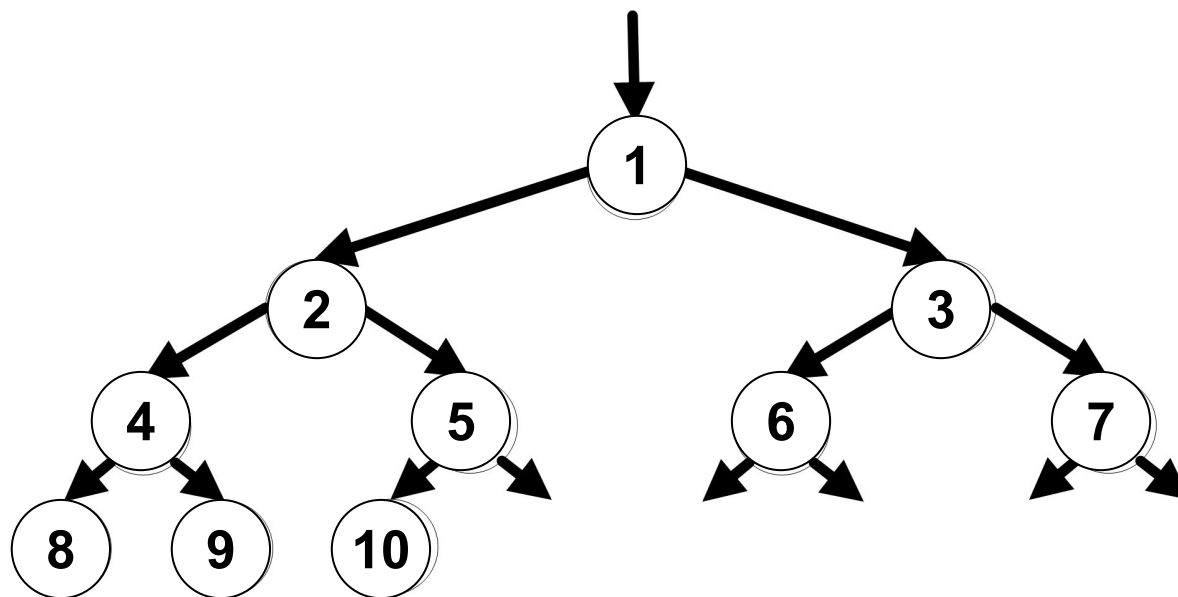


- Estrutura de dados cujas operações de inserção, remoção e substituição possuem a mesma eficiência
- Estrutura de dados que contém um conjunto finito de vértices (nós) e outro de arcos (arestas) que conectam os vértices

Nosso exemplo é uma árvore binária, pois cada nó tem no máximo dois filhos



- Árvore binária em que cada nó é menor ou igual que seus filhos, fazendo com que a raiz seja o menor valor
- Suas folhas ocupam um ou dois níveis sendo que o penúltimo é completo e as folhas do último nível se agrupam o mais à esquerda possível

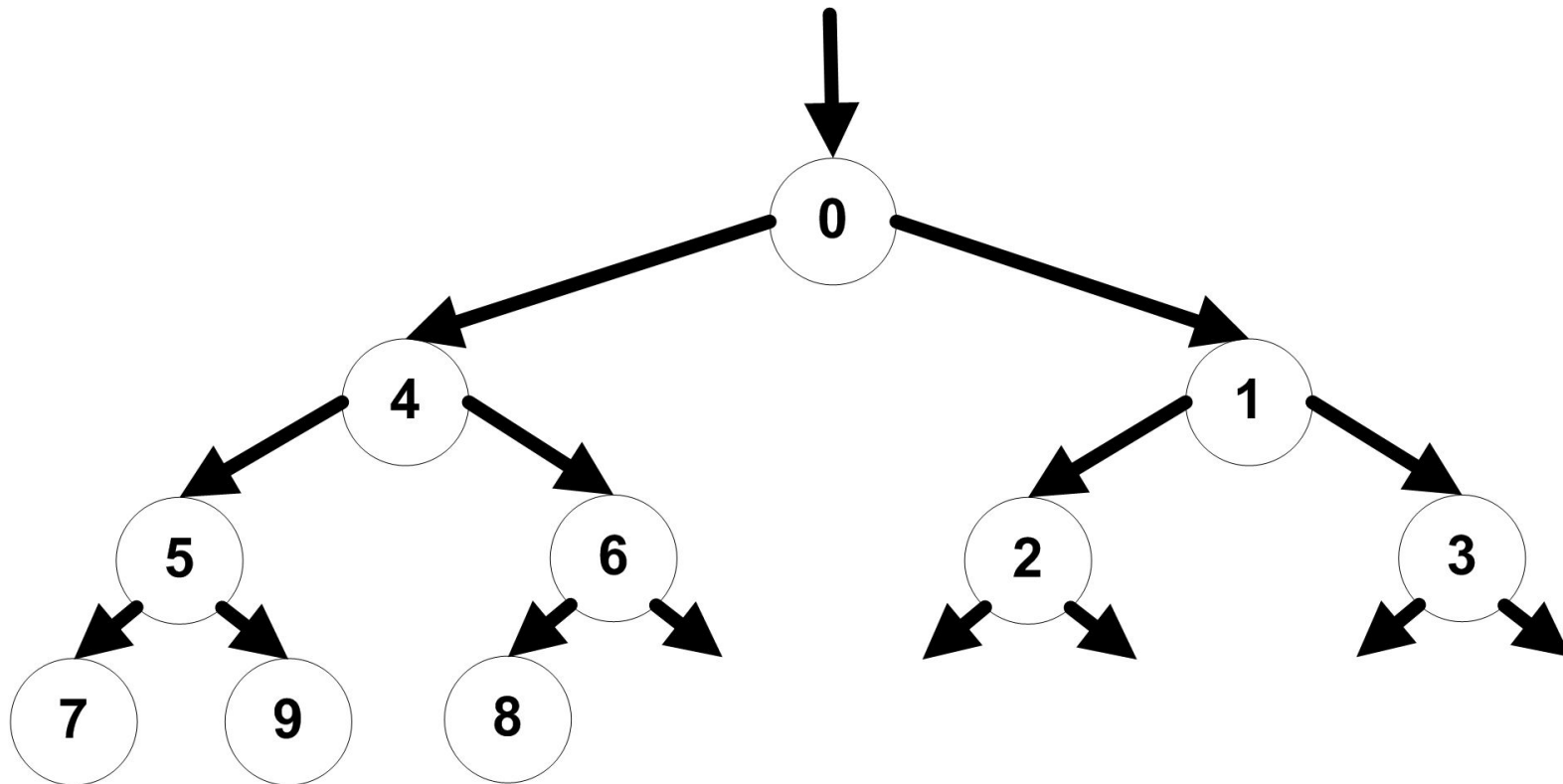


Exercício Resolvido (1)

- Mostre um heap com os elementos 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9

Exercício Resolvido (1)

- Mostre um heap com os elementos 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9



Heap Invertido

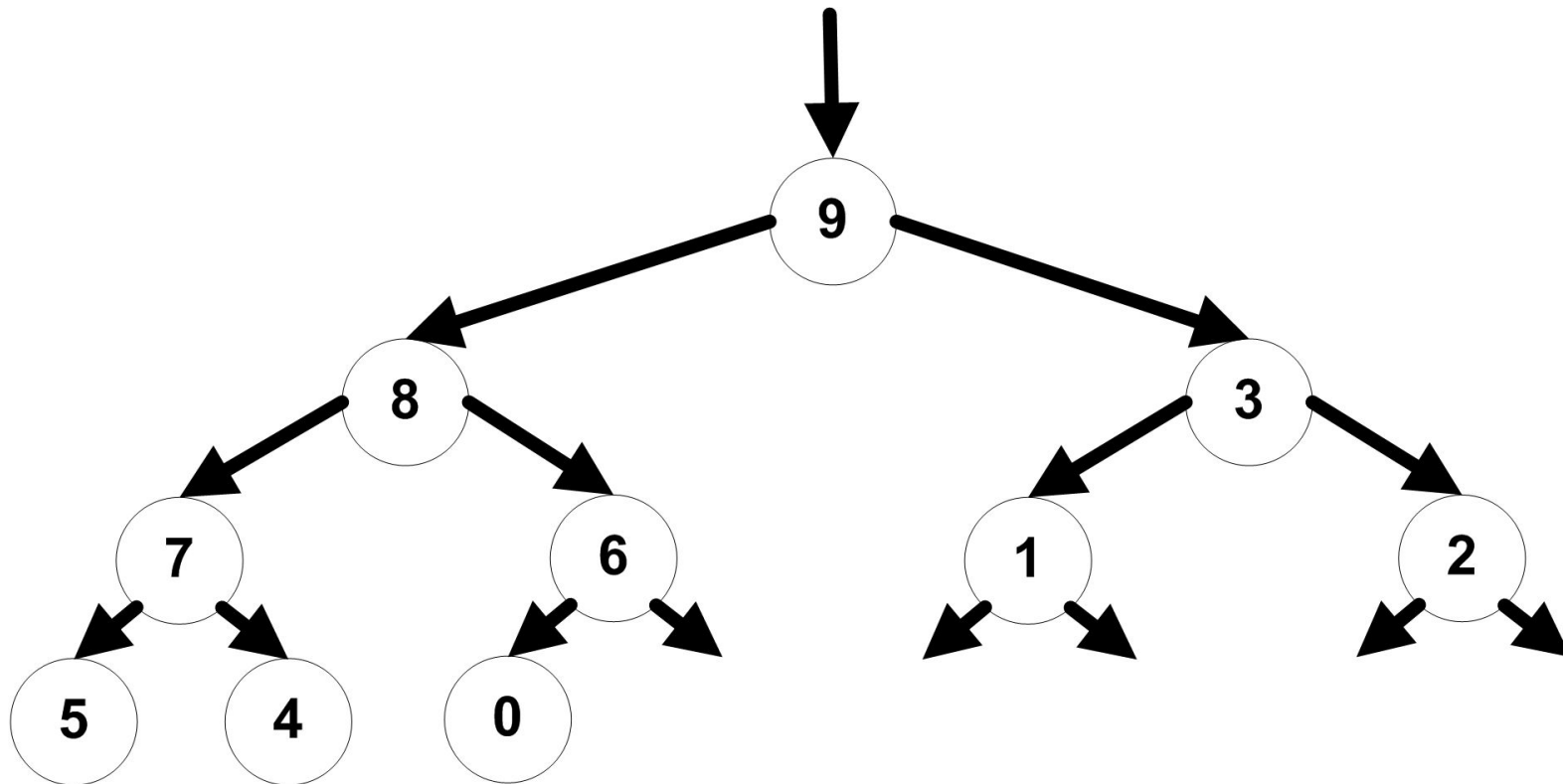
- Árvore binária em que o valor de cada nó é **maior** ou igual ao de seus filhos
- Suas folhas ocupam um ou dois níveis sendo que o penúltimo é completo e as folhas do último nível se agrupam o mais à esquerda possível
- O **maior** elemento encontra-se sempre na raiz

Exercício Resolvido (2)

- Mostre um heap invertido com os elementos 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9

Exercício Resolvido (2)

- Mostre um heap invertido com os elementos 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9



Consideração

- A partir deste ponto, neste material, a palavra heap será usada para designar o heap invertido

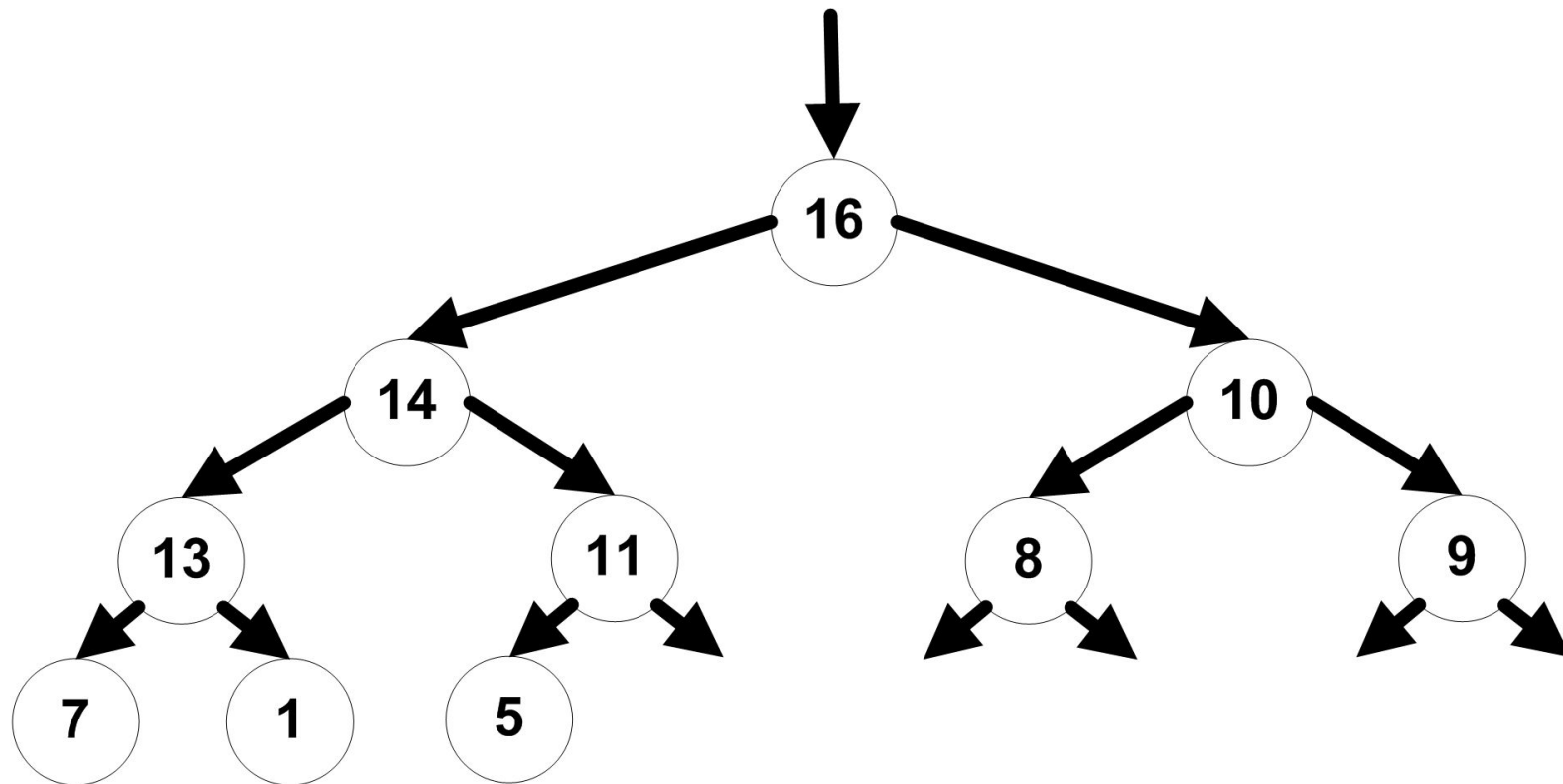
Representação do Heap em um *Array*

- Como representar um heap usando um *array*?
- Afinal, estamos apresentando um algoritmo para ordenar *arrays*...

E agora José?

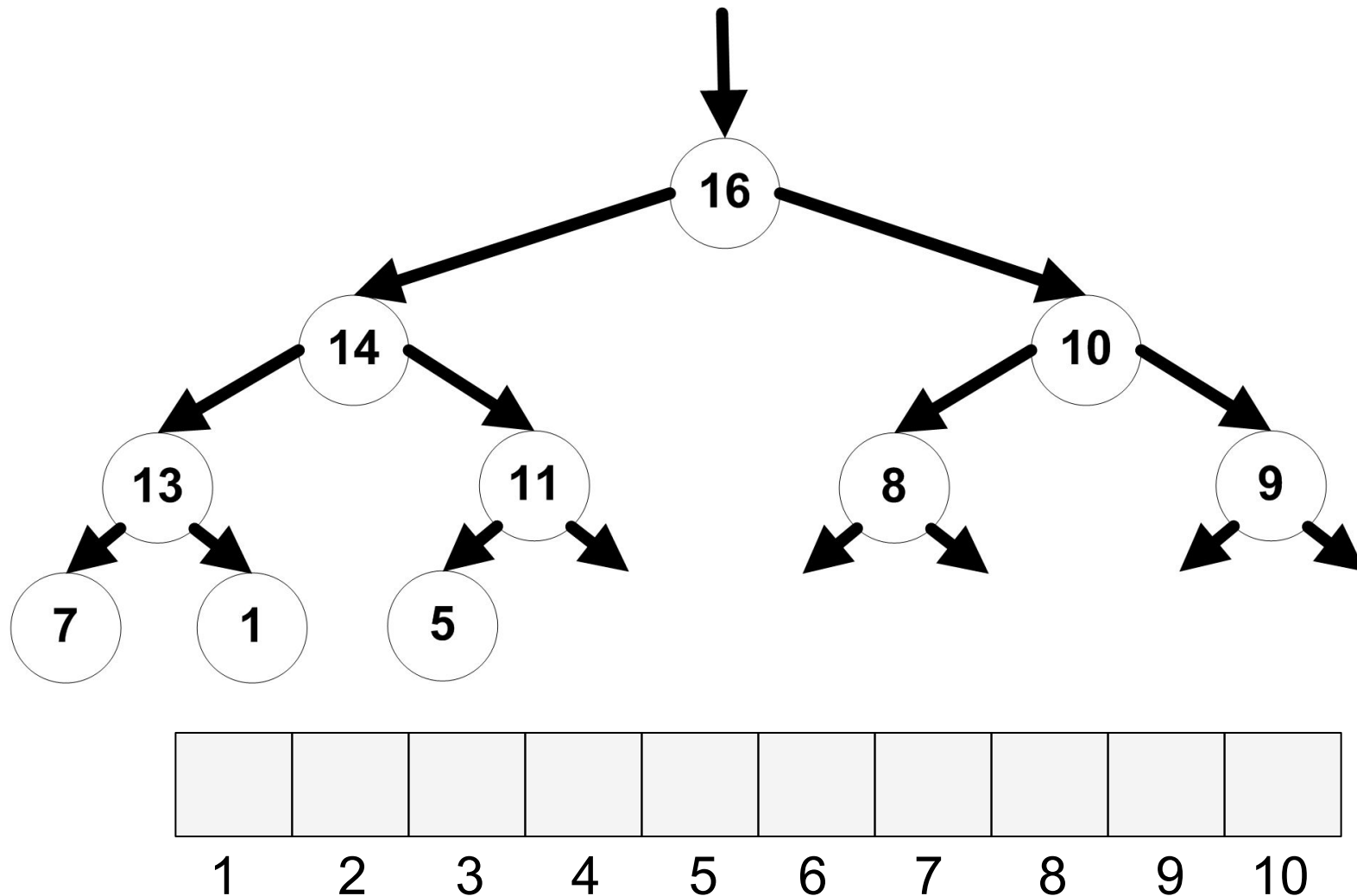
Exercício Resolvido (3)

- Represente o heap abaixo em um *array*



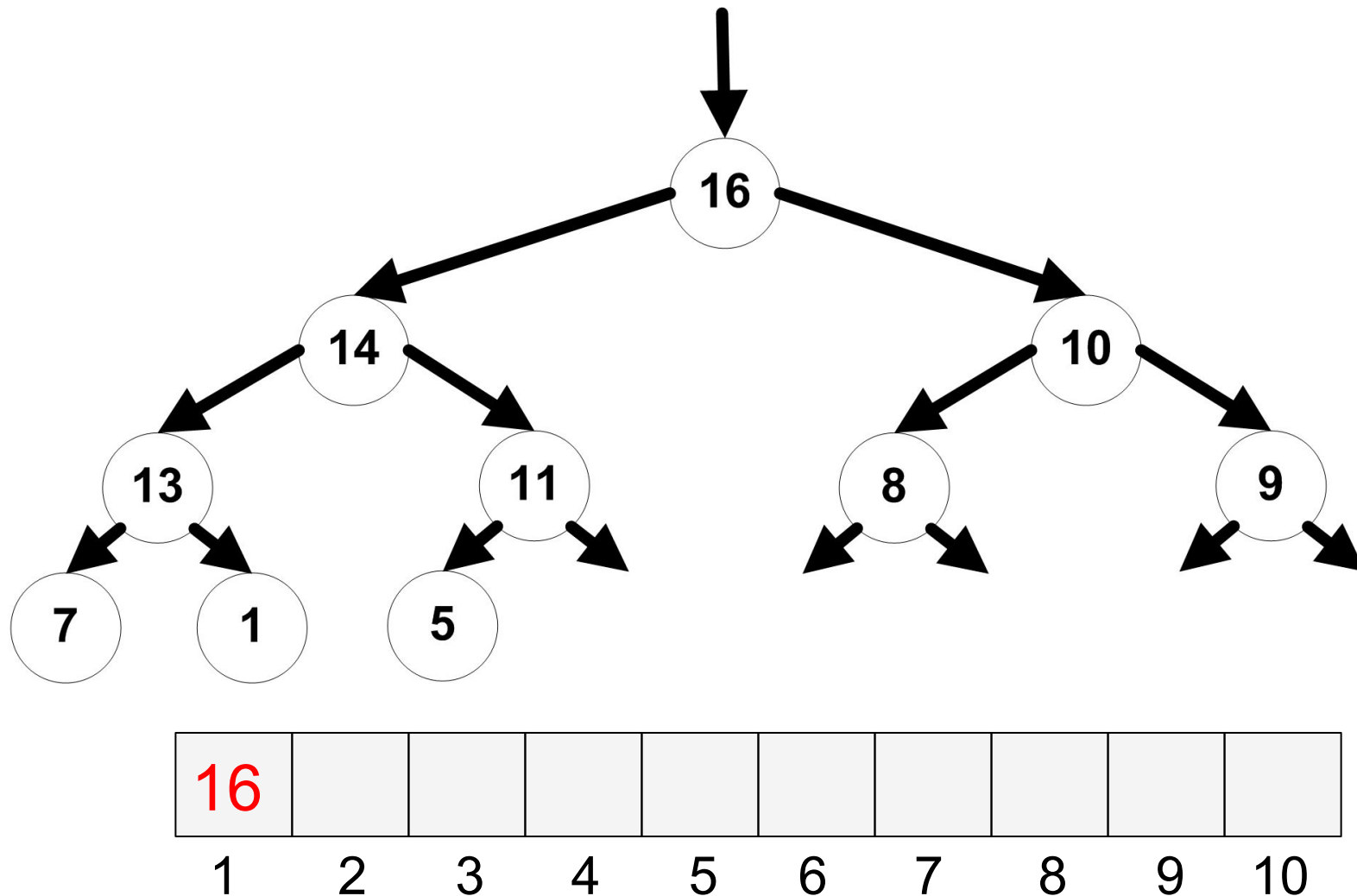
Exercício Resolvido (3)

- Represente o heap abaixo em um *array*



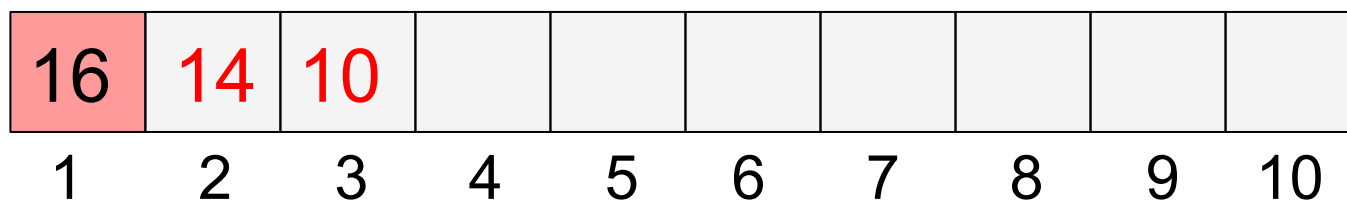
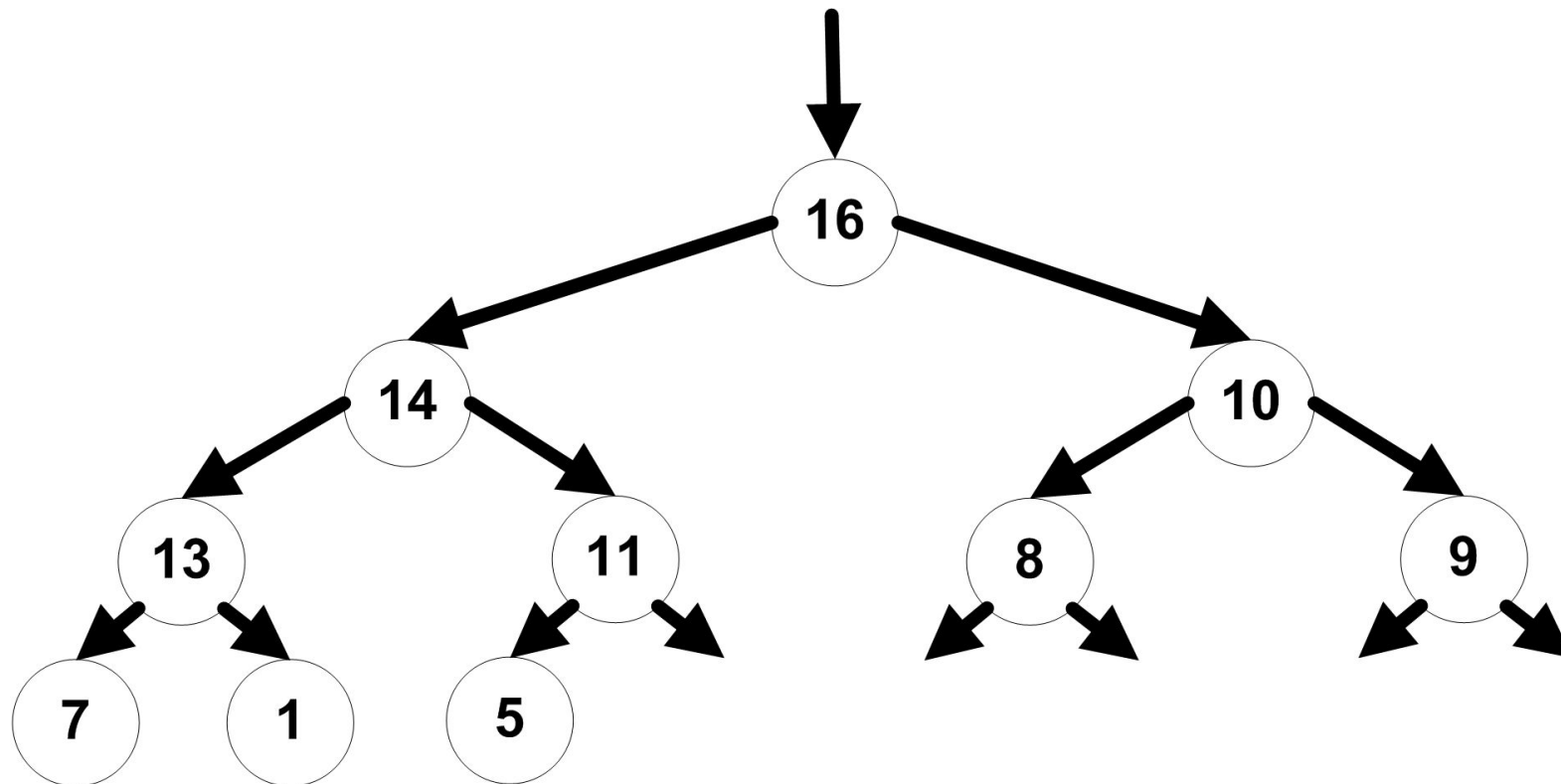
Exercício Resolvido (3)

- Represente o heap abaixo em um *array*



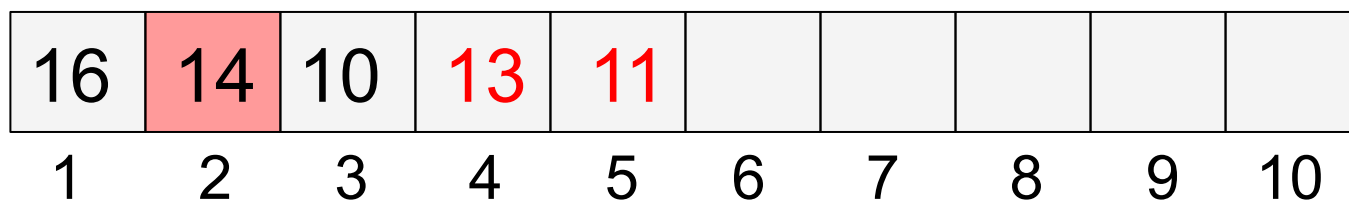
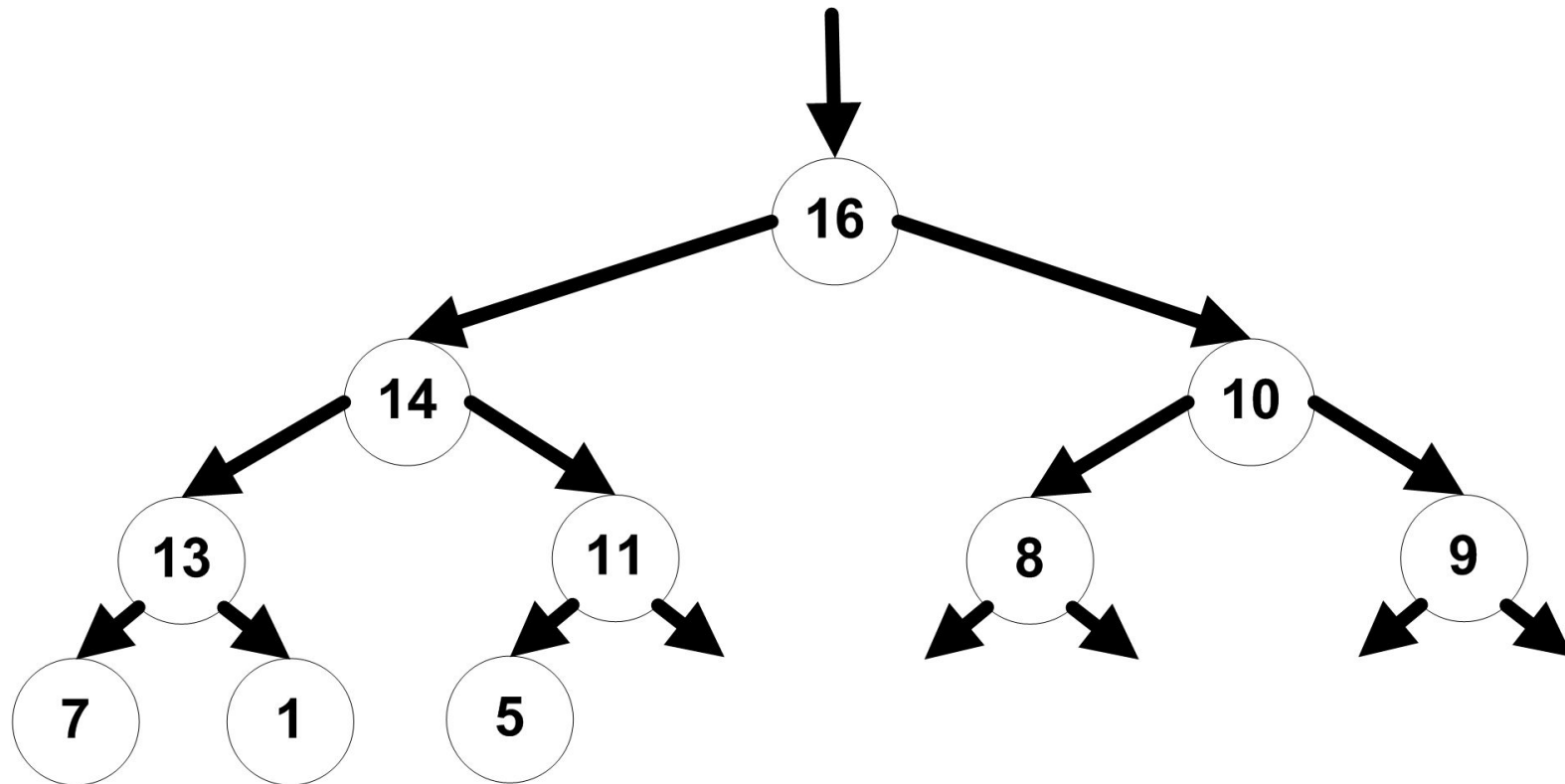
Exercício Resolvido (3)

- Represente o heap abaixo em um *array*



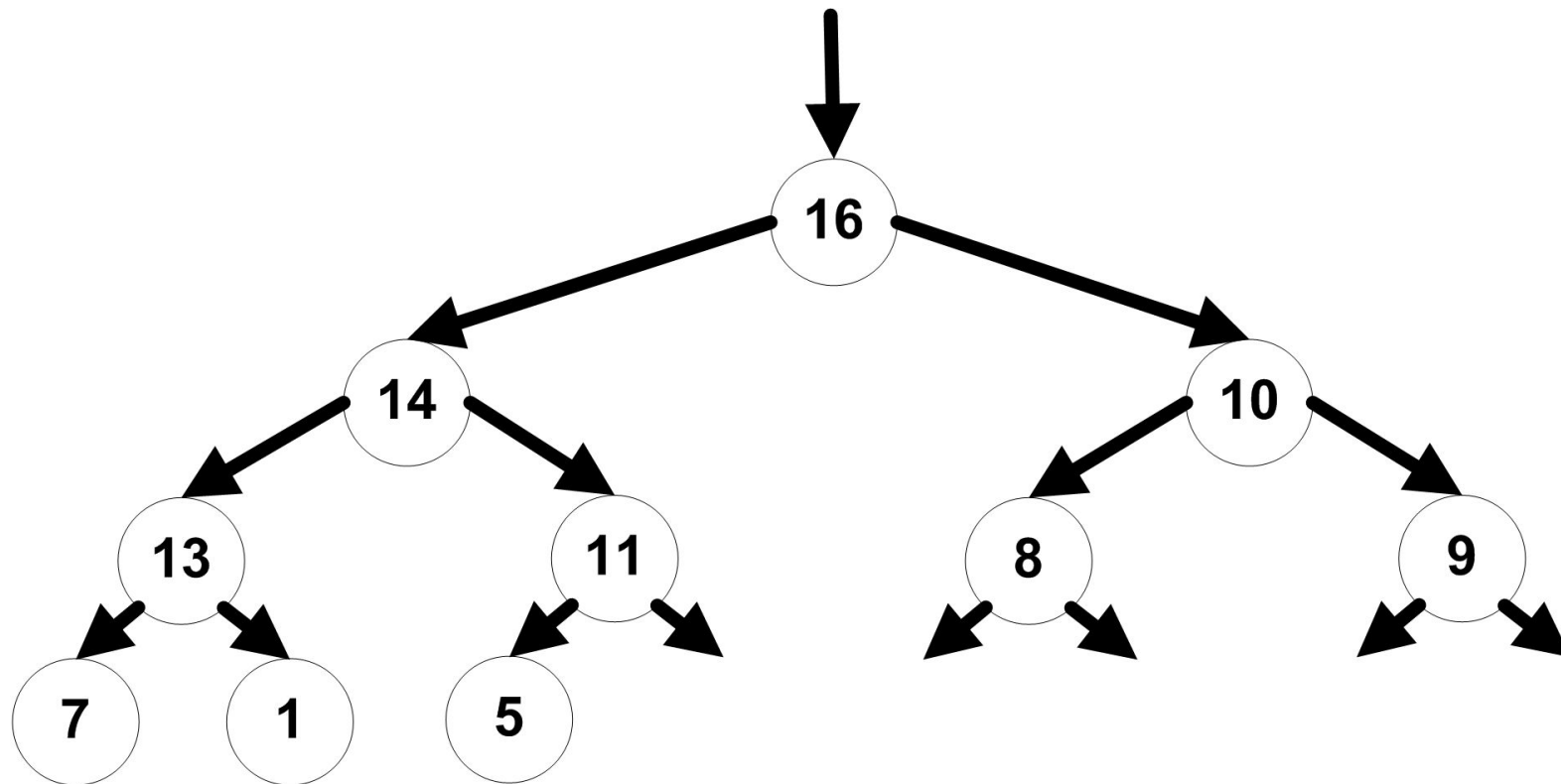
Exercício Resolvido (3)

- Represente o heap abaixo em um *array*



Exercício Resolvido (3)

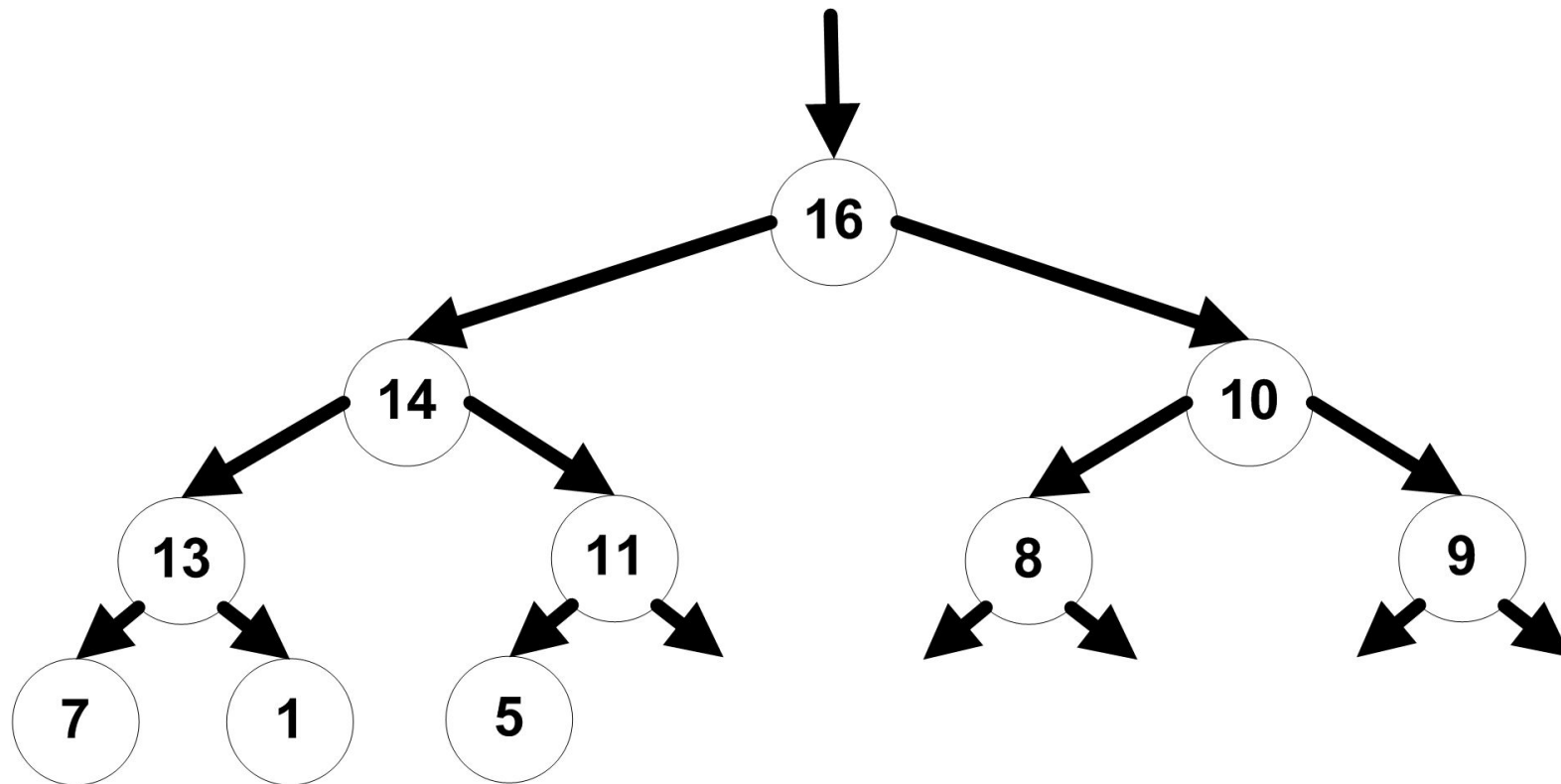
- Represente o heap abaixo em um *array*



16	14	10	13	11	8	9			
1	2	3	4	5	6	7	8	9	10

Exercício Resolvido (3)

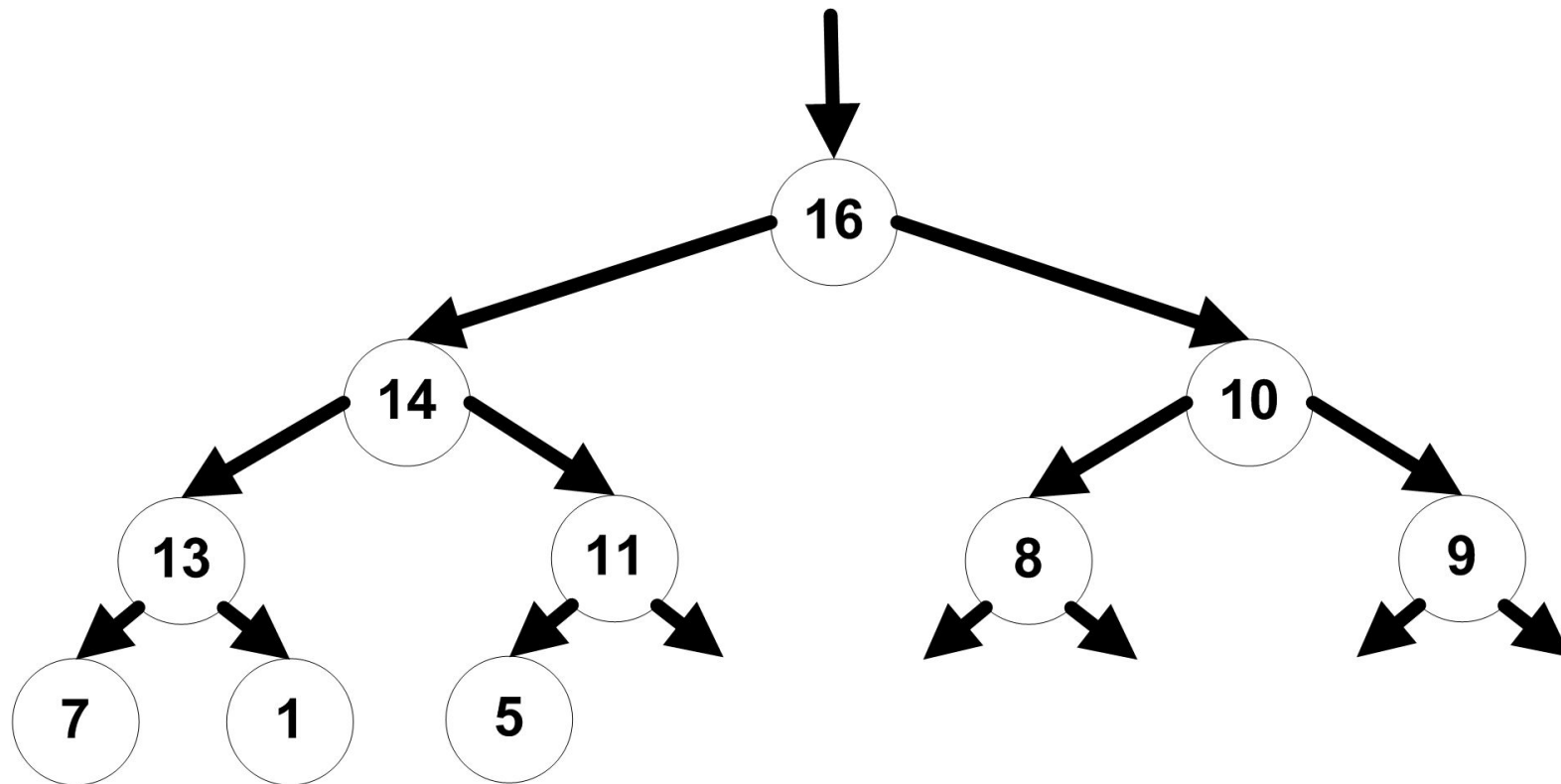
- Represente o heap abaixo em um *array*



16	14	10	13	11	8	9	7	1	
1	2	3	4	5	6	7	8	9	10

Exercício Resolvido (3)

- Represente o heap abaixo em um *array*



16	14	10	13	11	8	9	7	1	5
1	2	3	4	5	6	7	8	9	10

Representação do Heap em um *Array*

- Podemos representar qualquer árvore binária em um *array* fazendo:
 - $array[1]$ é a raiz
 - $array[i/2]$ é o pai de $array[i]$, para $i > 1$
 - Se $array[2 * i]$ e $array[2 * i + 1]$ existem, eles são filhos de $array[i]$
 - Se i é maior que $(n / 2)$, $array[i]$ é uma folha

Exercício Resolvido (4)

- Faça um método que recebe um número inteiro i e mostra na tela o elemento localizado na posição i do heap e todos os ancestrais daquele elemento. Por exemplo, dado o heap da página anterior, se seu método receber o número 10, ele mostra na tela 5, 11, 14 e 16

Exercício Resolvido (4)

- Faça um método que recebe um número inteiro i e mostra na tela o elemento localizado na posição i do heap e todos os ancestrais daquele elemento. Por exemplo, dado o heap da página anterior, se seu método receber o número 10, ele mostra na tela 5, 11, 14 e 16

```
void metodo(int i) {  
    for ( /**/; i >= 1; i /= 2){  
        System.out.println(array[i]);  
    }  
}
```

Exercício Resolvido (5)

- Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

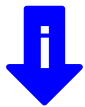
9	8	3	7	6	1	2	5	4	0	31
1	2	3	4	5	6	7	8	9	10	11

Exercício Resolvido (5)

- Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

```
void construir(int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

9	8	3	7	6	1	2	5	4	0	31
1	2	3	4	5	6	7	8	9	10	11



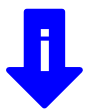
Exercício Resolvido (5)

- Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

```
void construir(int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

true: $11 > 1 \ \&\& \ \text{array}[11] > \text{array}[5]$

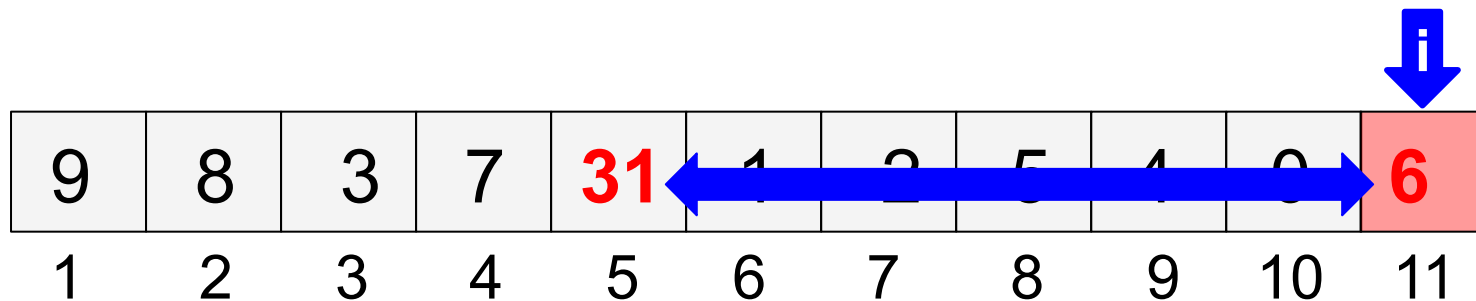
9	8	3	7	6	1	2	5	4	0	31
1	2	3	4	5	6	7	8	9	10	11



Exercício Resolvido (5)

- Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

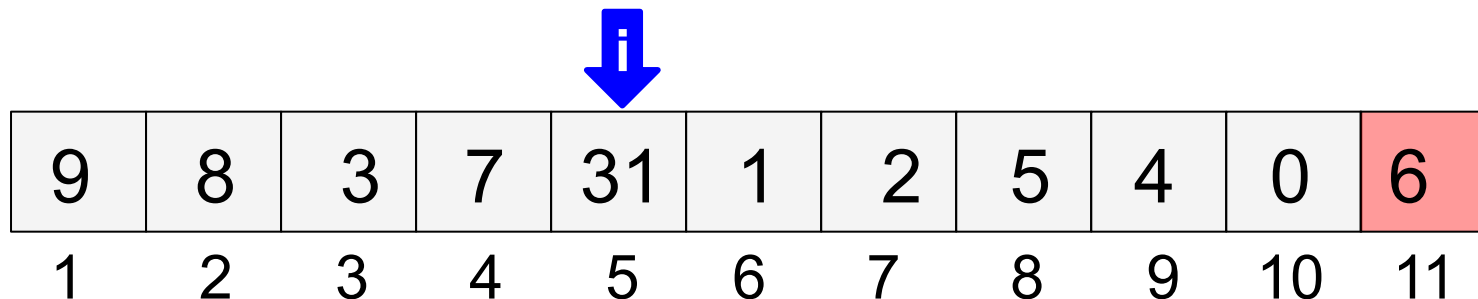
```
void construir(int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```



Exercício Resolvido (5)

- Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

```
void construir(int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```



Exercício Resolvido (5)

- Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

```
void construir(int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

true: $5 > 1 \ \&\& \ \text{array}[5] > \text{array}[2]$

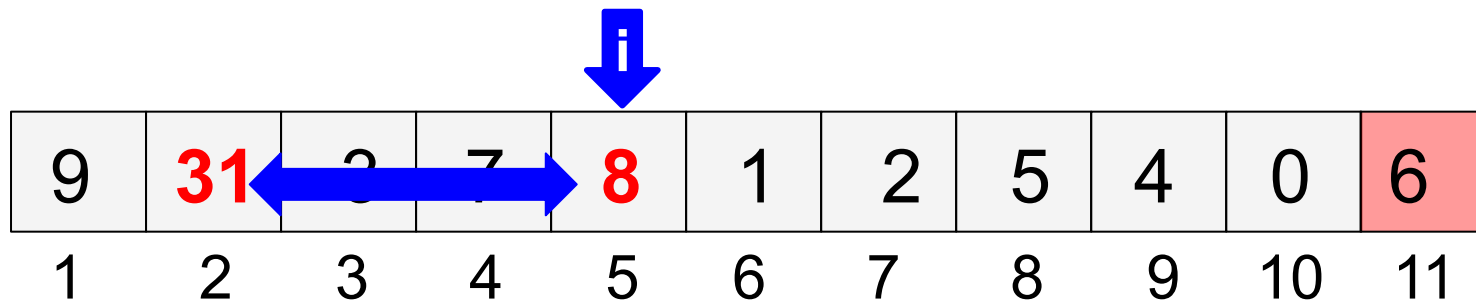


9	8	3	7	31	1	2	5	4	0	6
1	2	3	4	5	6	7	8	9	10	11

Exercício Resolvido (5)

- Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

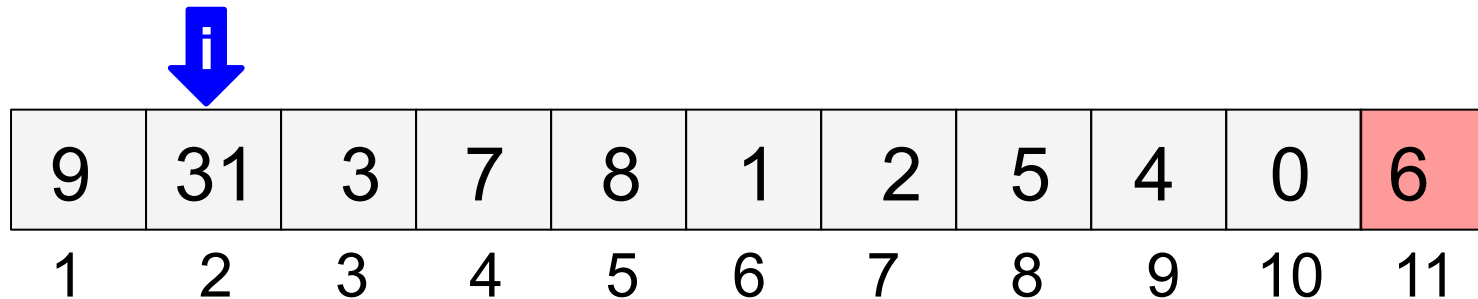
```
void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```



Exercício Resolvido (5)

- Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

```
void construir(int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```



Exercício Resolvido (5)

- Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

```
void construir(int tam){
  for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
    swap(i, i/2);
  }
}
```

true: $2 > 1 \ \&\& \ \text{array}[2] > \text{array}[1]$

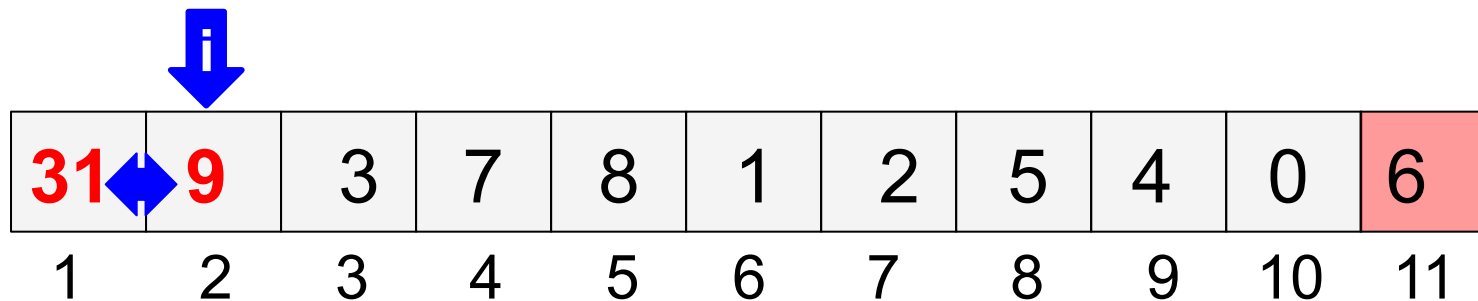


9	31	3	7	8	1	2	5	4	0	6
1	2	3	4	5	6	7	8	9	10	11

Exercício Resolvido (5)

- Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

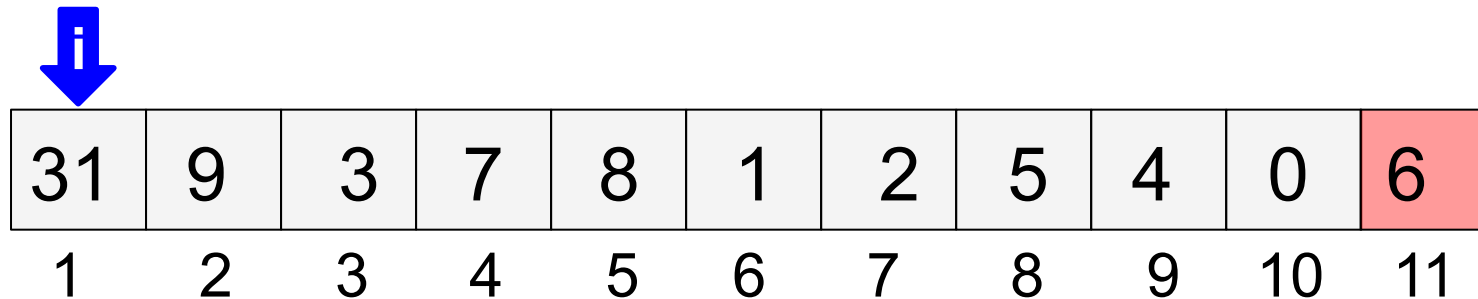
```
void construir(int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```



Exercício Resolvido (5)

- Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

```
void construir(int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```



Exercício Resolvido (5)

- Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

```
void construir(int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

false: $1 > 1 \ \&\& \dots$



31	9	3	7	8	1	2	5	4	0	6
1	2	3	4	5	6	7	8	9	10	11

Operações Possíveis com um Heap

- Construção
- Inserção de um novo elemento
- Remoção do elemento com a maior chave
- Remoção de um elemento qualquer
- Unificar dois ou mais heaps

Exercício (1)

- Faça um método que receba dois heaps e retorne um terceiro heap unificando as duas estruturas recebidas

- Definição de Heap
- **Funcionamento básico** ←
- Algoritmo em *C like*
- Análise dos número de comparações e movimentações

Funcionamento Básico

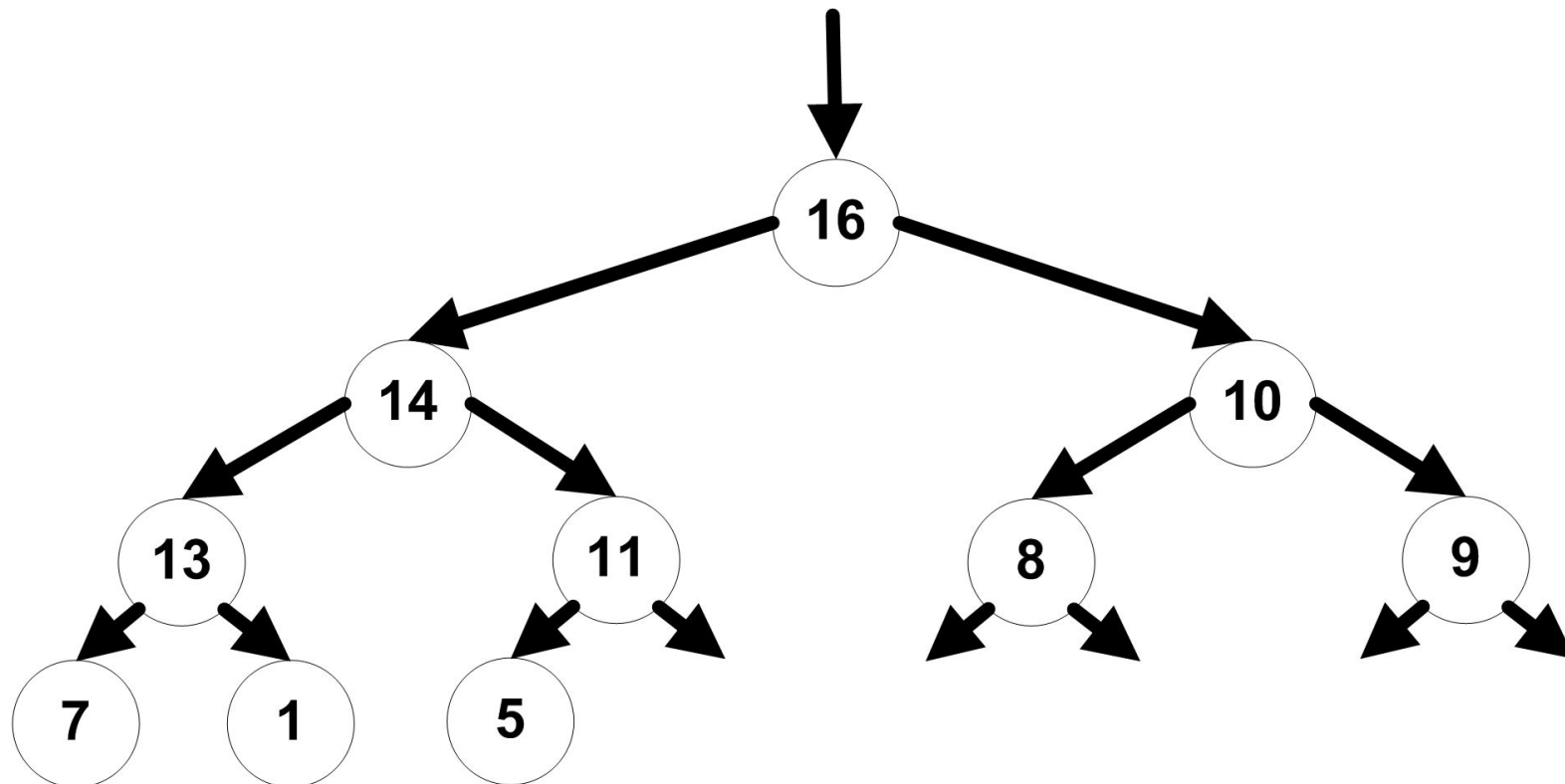
- Construa o heap inserindo sistematicamente cada um dos elementos do *array*
- Remova sistematicamente cada elemento do heap, reconstrua o heap e insira o elemento removido na posição do *array* imediatamente seguinte ao tamanho corrente do heap
- A seguir, veja os princípios de inserção e remoção no heap

Princípio de Inserção

- Crie uma nova folha (contendo o novo elemento) no último nível do heap.
Se esse estiver completo, recomece um novo nível
- Se o novo elemento for maior que seu pai, troque-os e realize o mesmo processo para o pai, avô, bisavô e, assim, sucessivamente, até que todos os pais sejam maiores que seus filhos

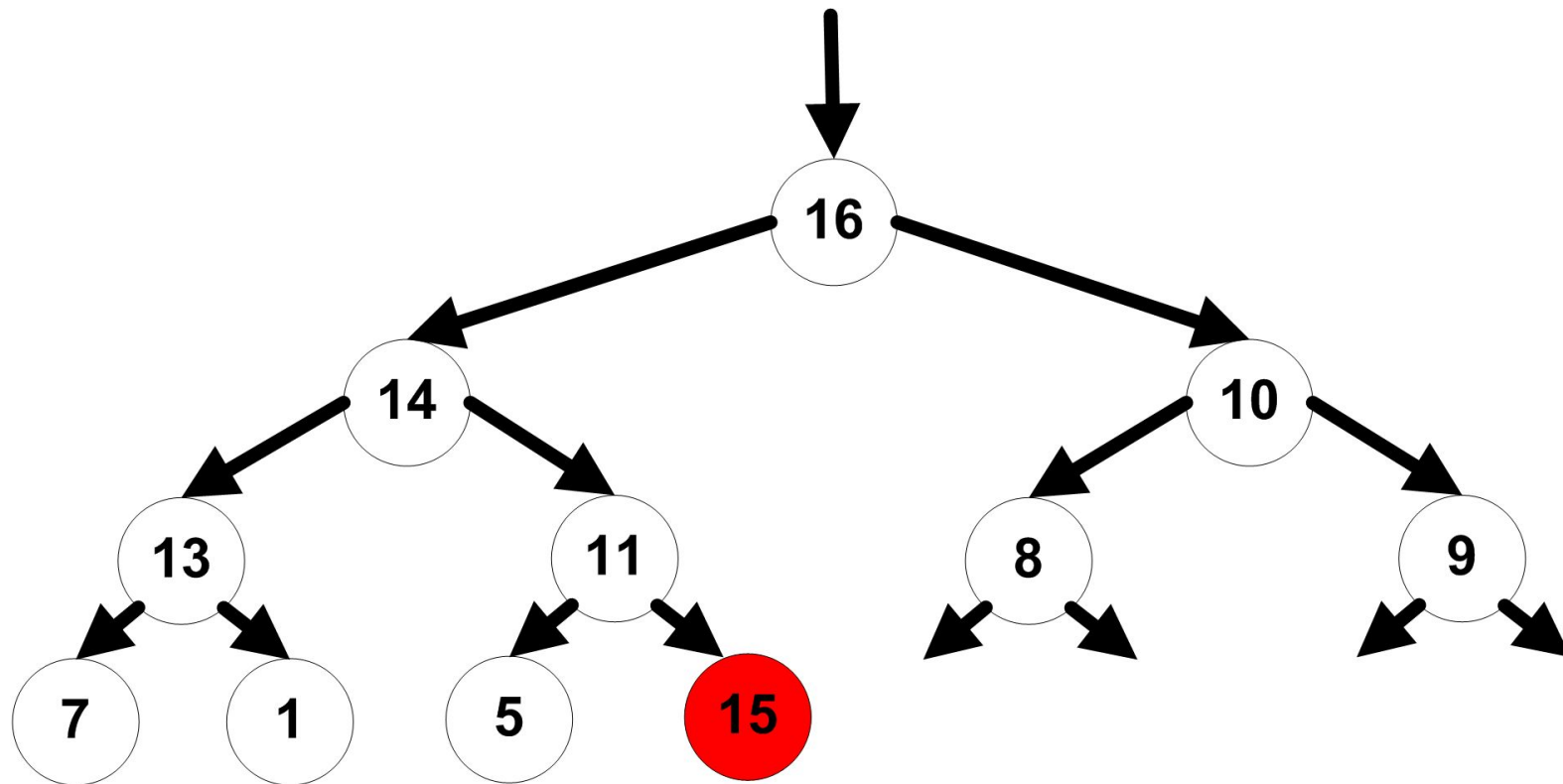
Exercício Resolvido (6)

- Inserir o 15 no heap abaixo



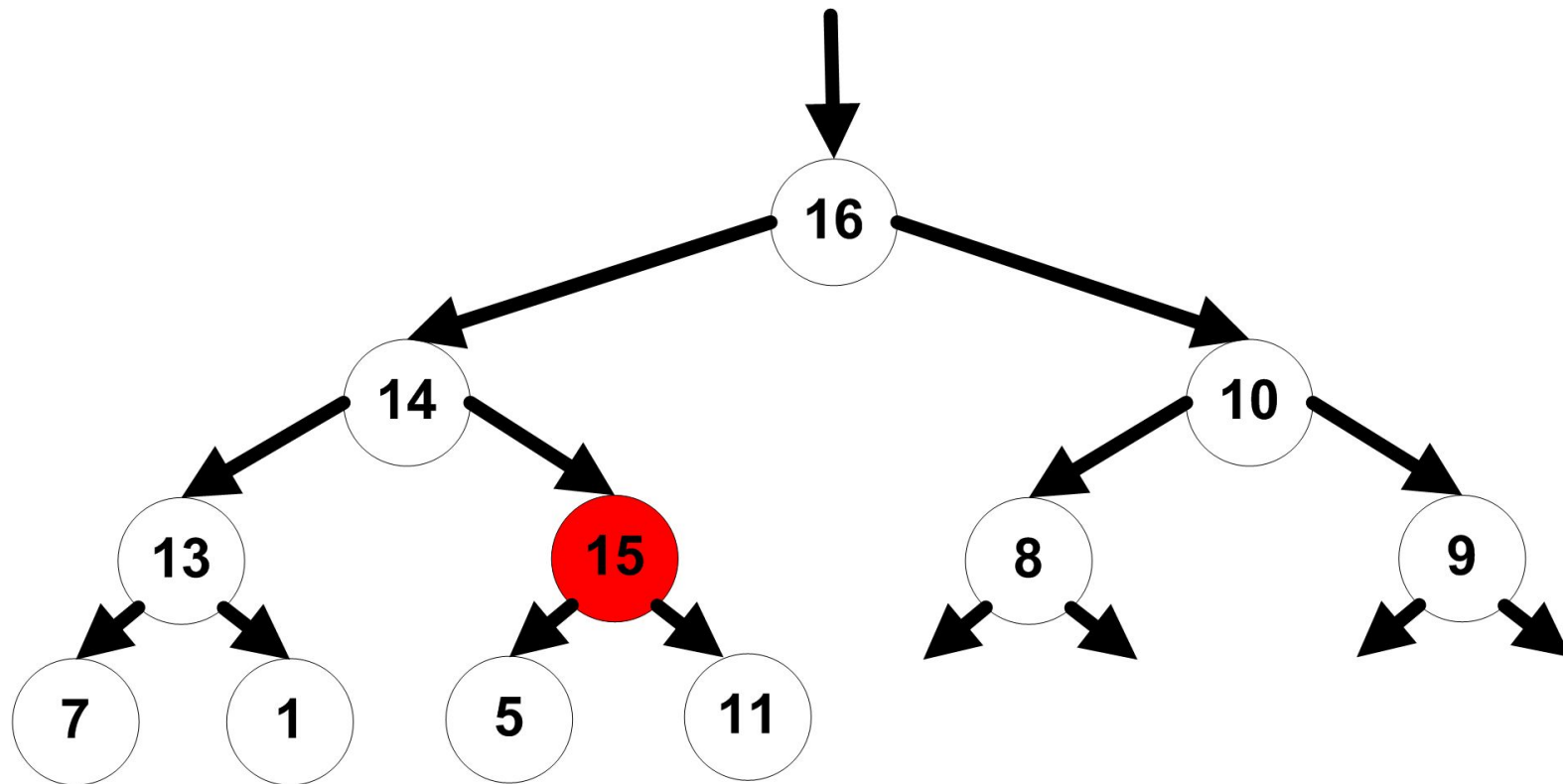
Exercício Resolvido (6)

- Inserir o 15 no heap abaixo



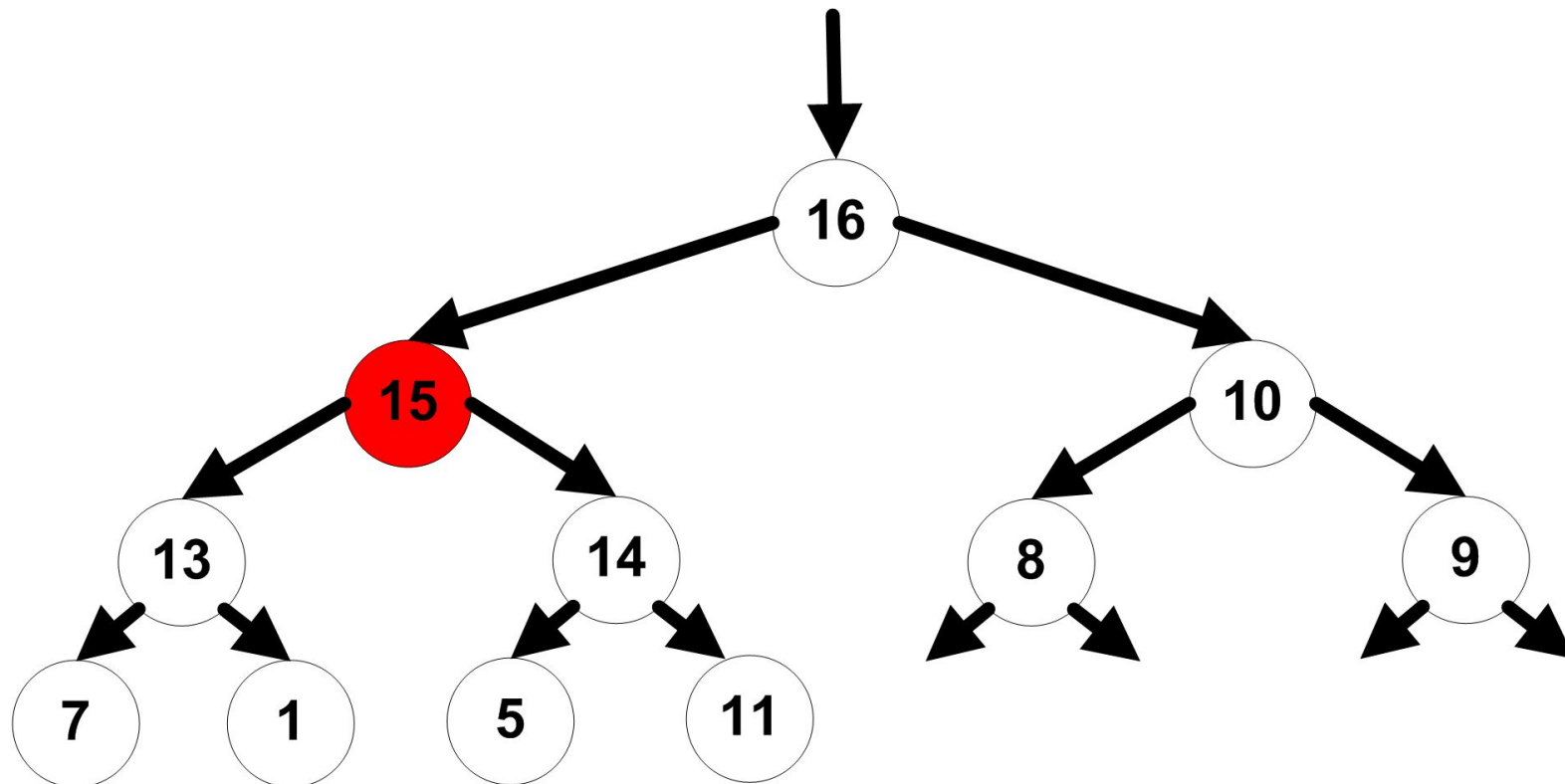
Exercício Resolvido (6)

- Inserir o 15 no heap abaixo



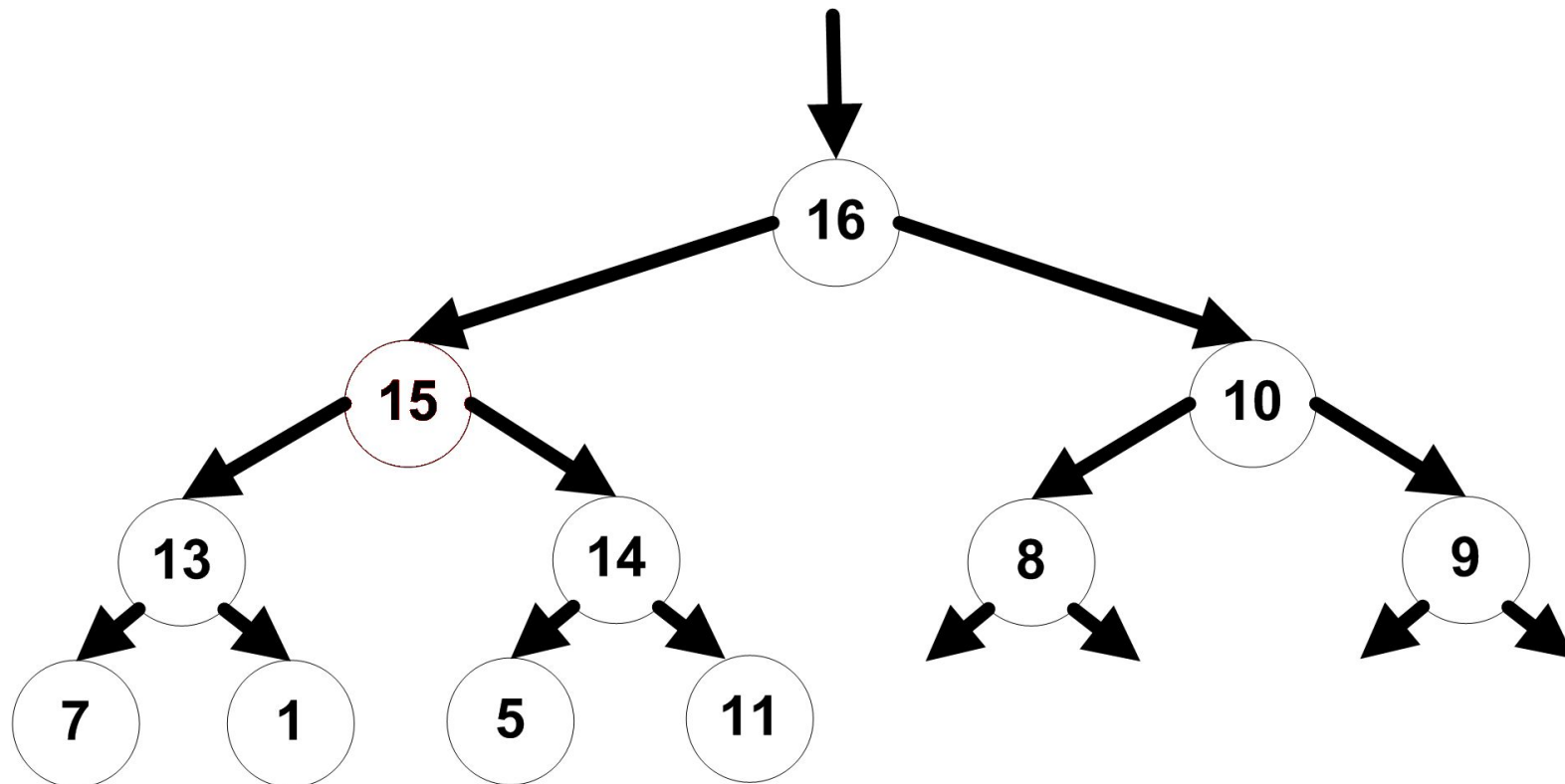
Exercício Resolvido (6)

- Inserir o 15 no heap abaixo



Exercício (2)

- Insira os números 11 e 9 na heap abaixo, respectivamente

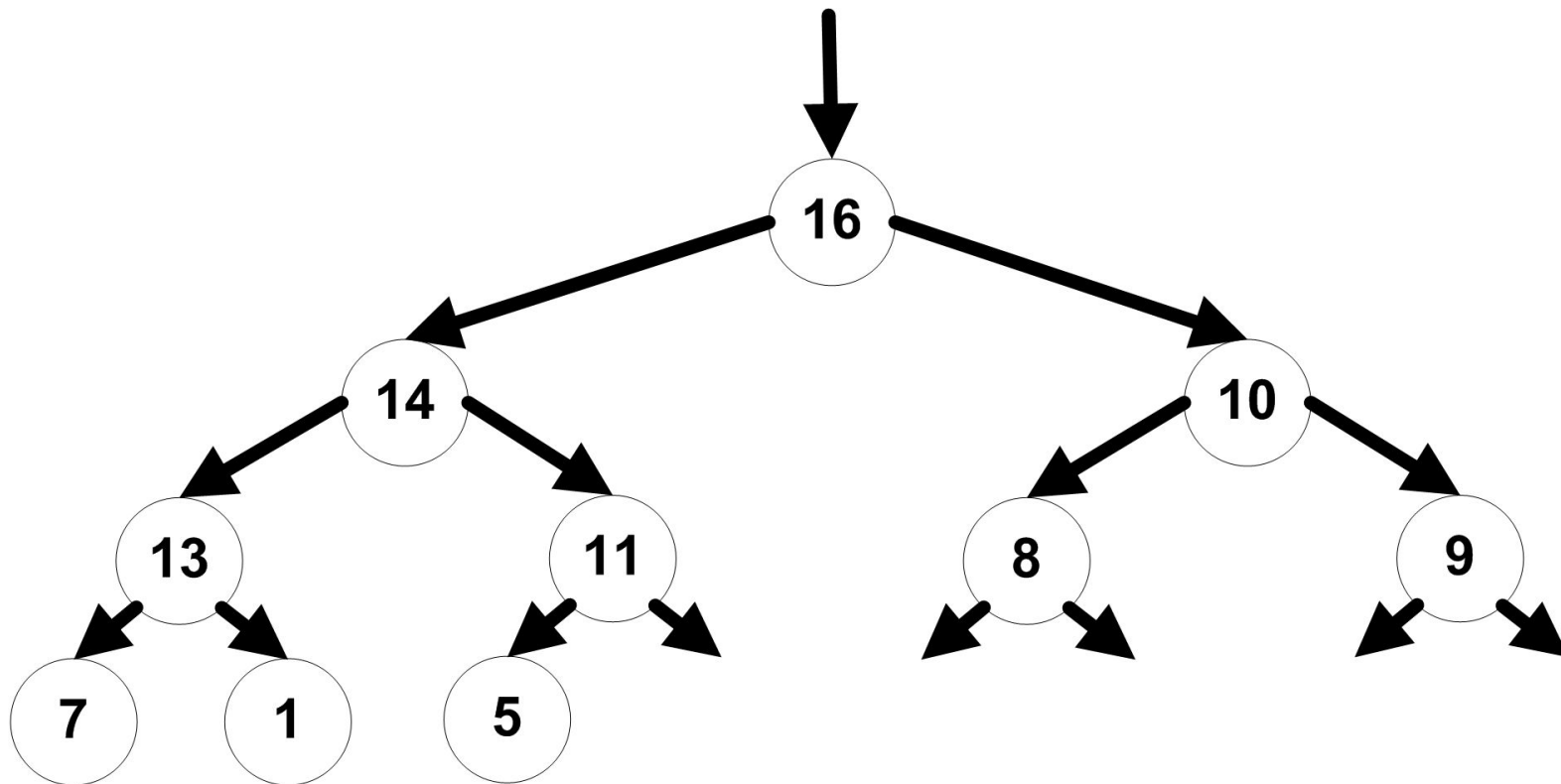


Princípio de Remoção

- Armazene o elemento da raiz em uma variável temporária
- Substitua o elemento da raiz pelo da última folha do último nível
- Remova a última folha do último nível
- Troque o elemento da raiz com o de seu maior filho
- Repita o passo anterior para o filho com elemento trocado até que todos os pais sejam maiores que seus filhos

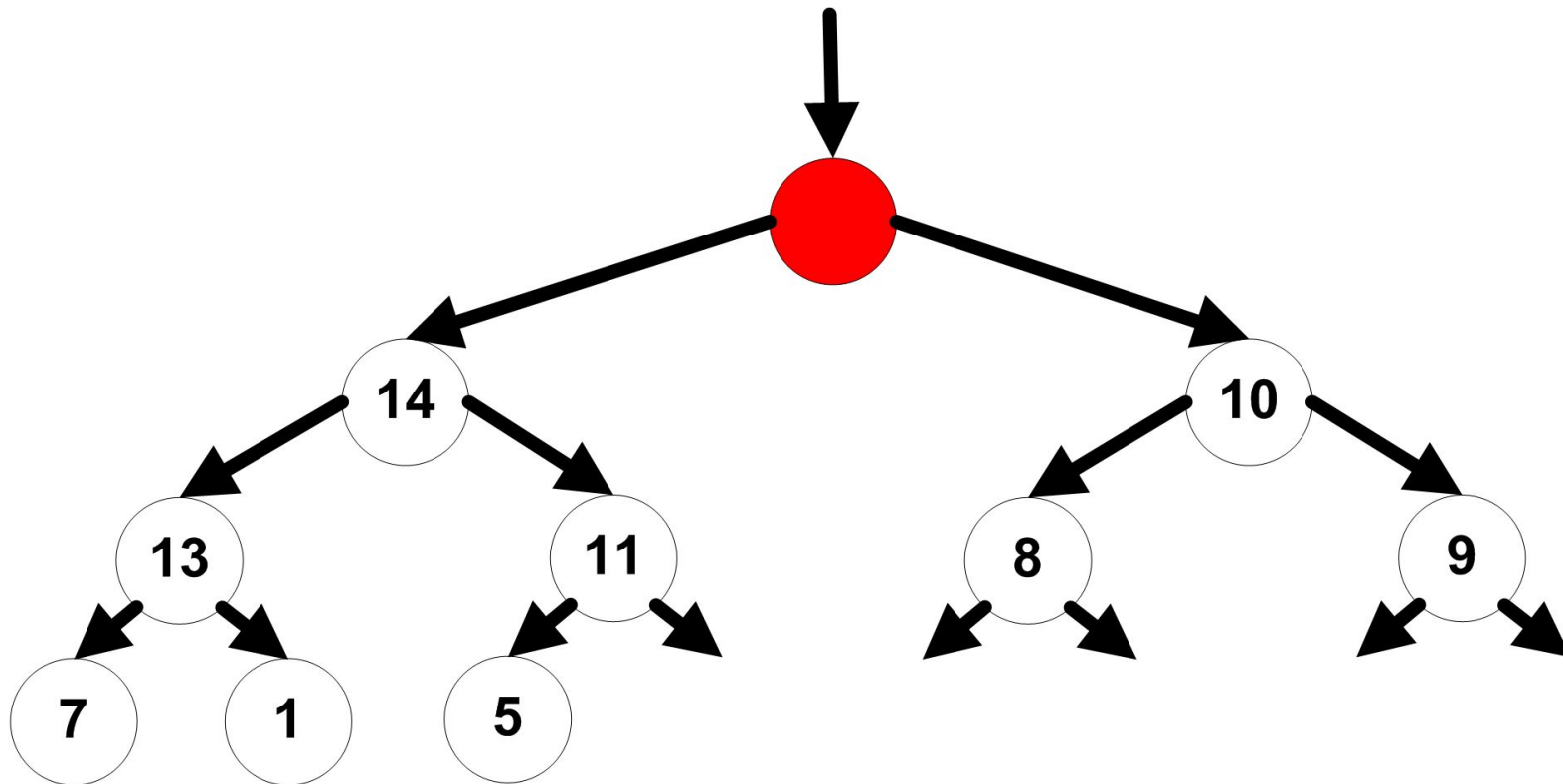
Exercício Resolvido (7)

- Remova a raiz do heap abaixo



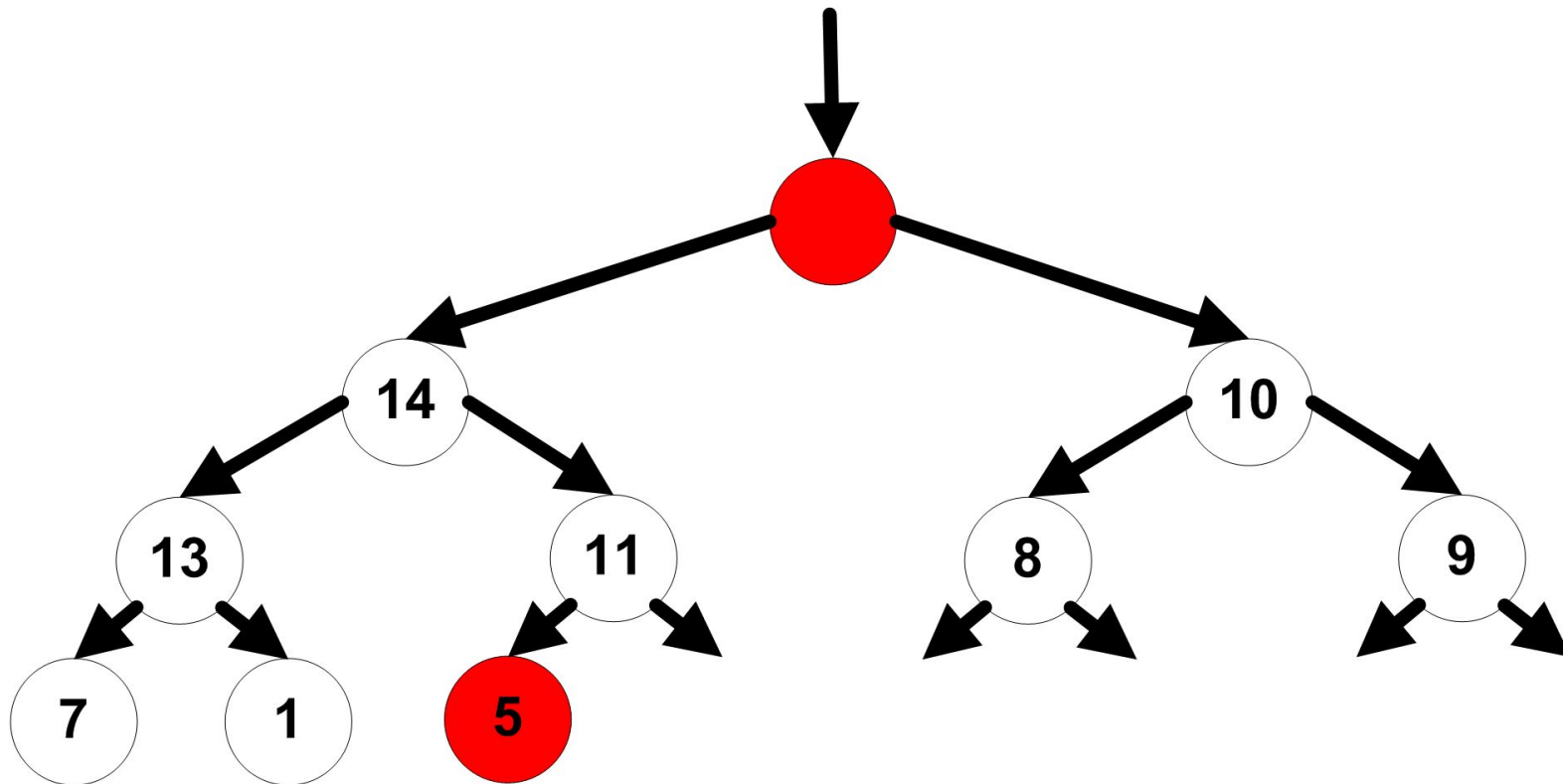
Exercício Resolvido (7)

- Remova a raiz do heap abaixo



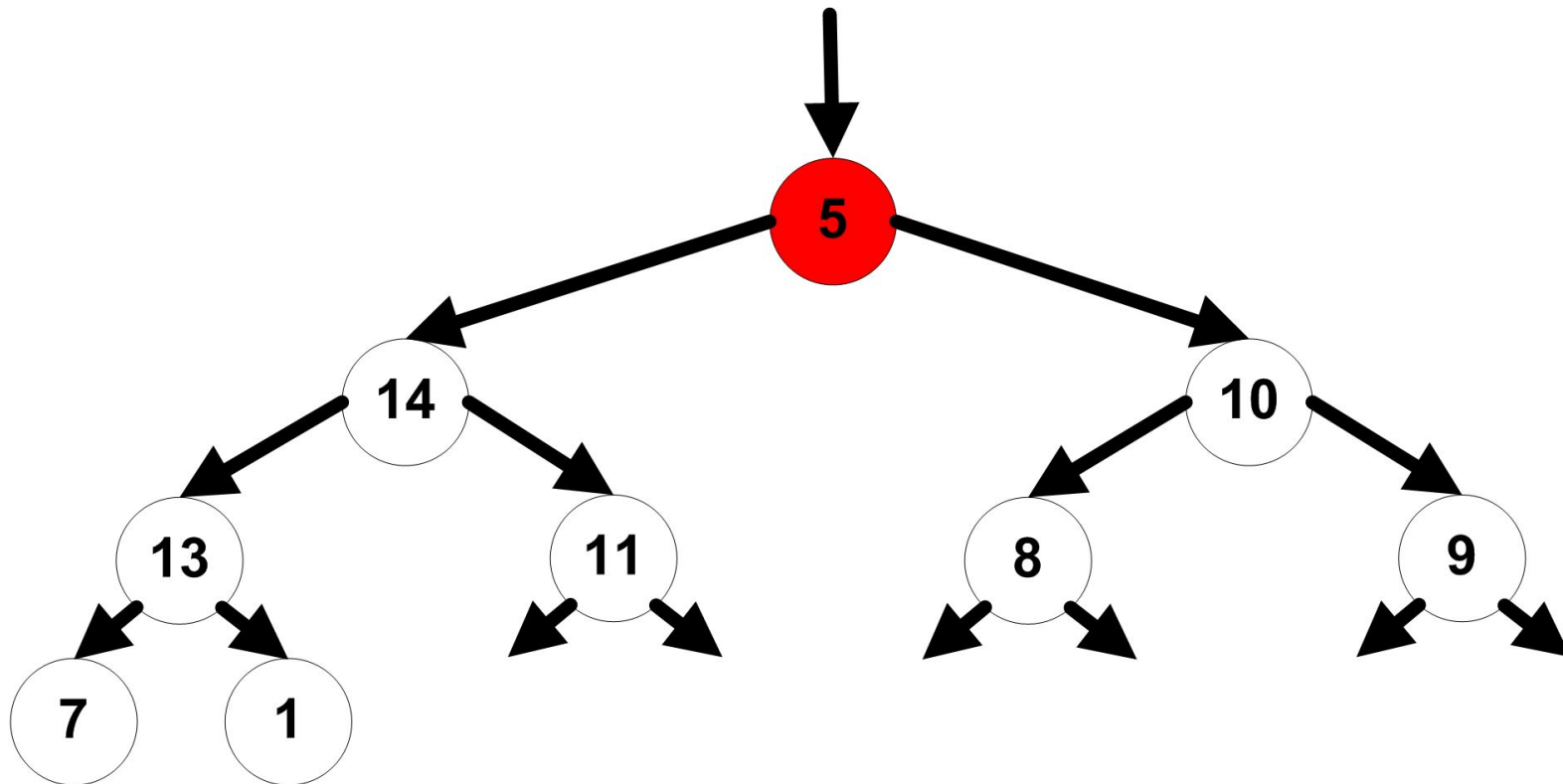
Exercício Resolvido (7)

- Remova a raiz do heap abaixo



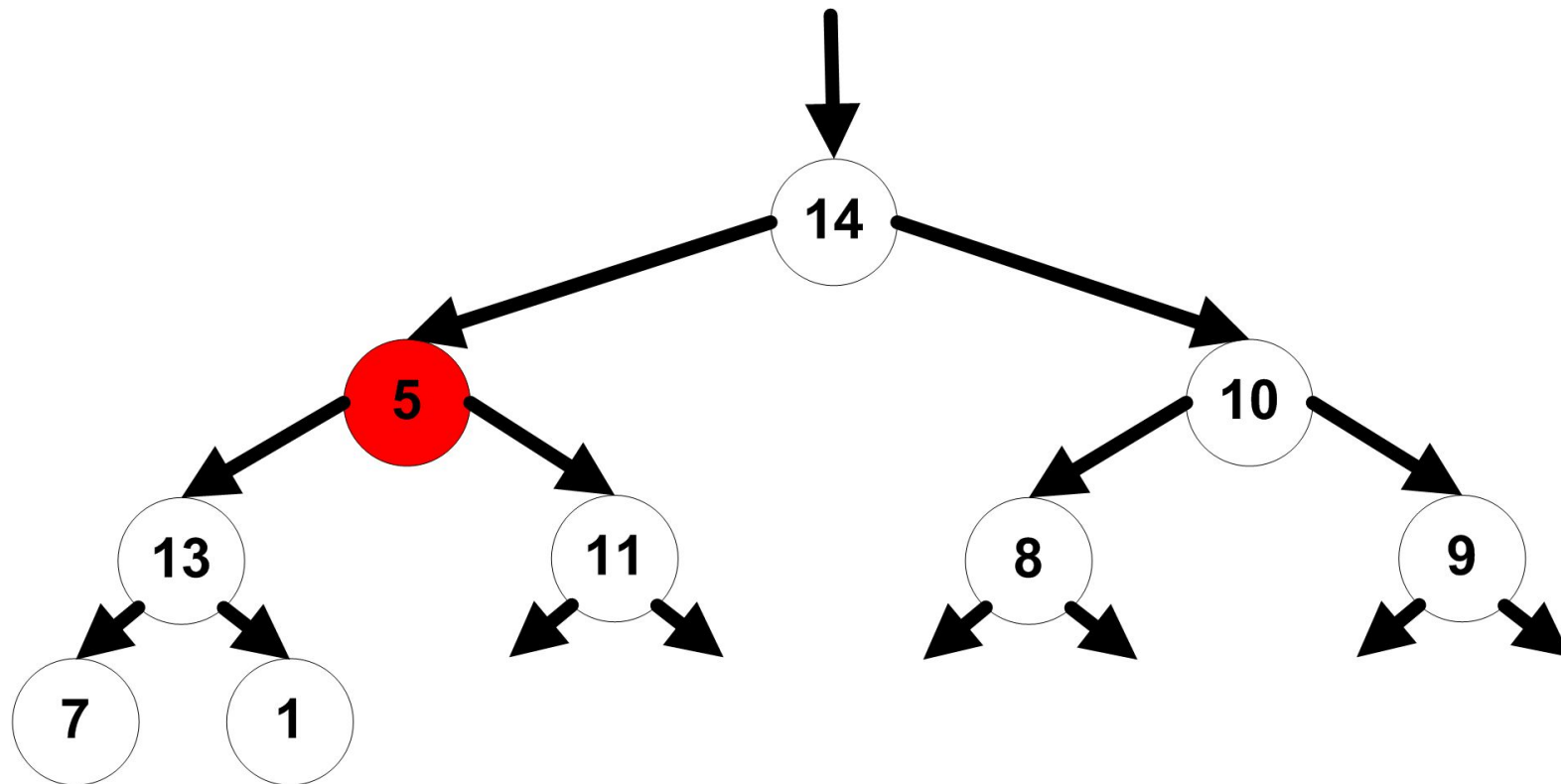
Exercício Resolvido (7)

- Remova a raiz do heap abaixo



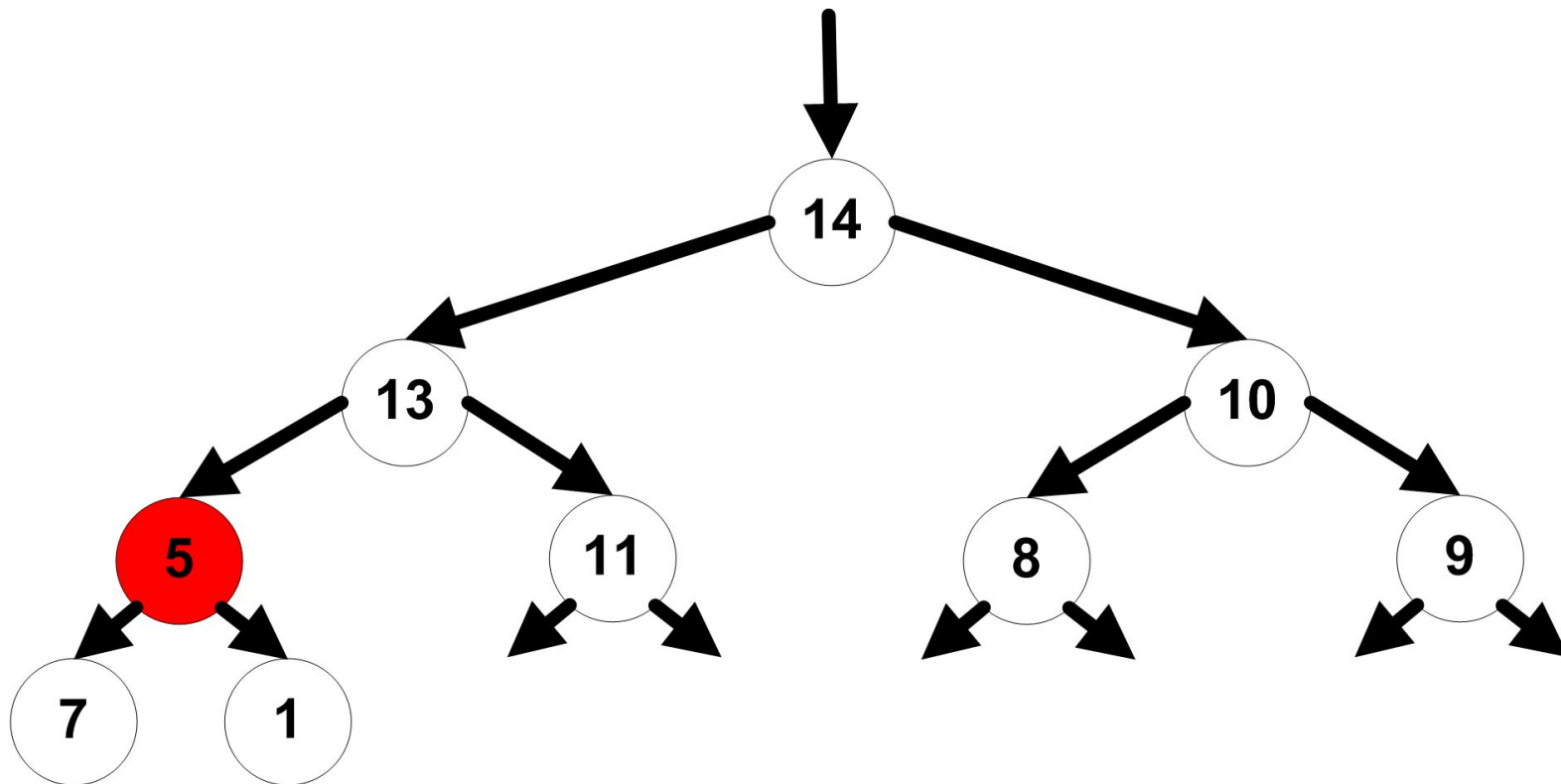
Exercício Resolvido (7)

- Remova a raiz do heap abaixo



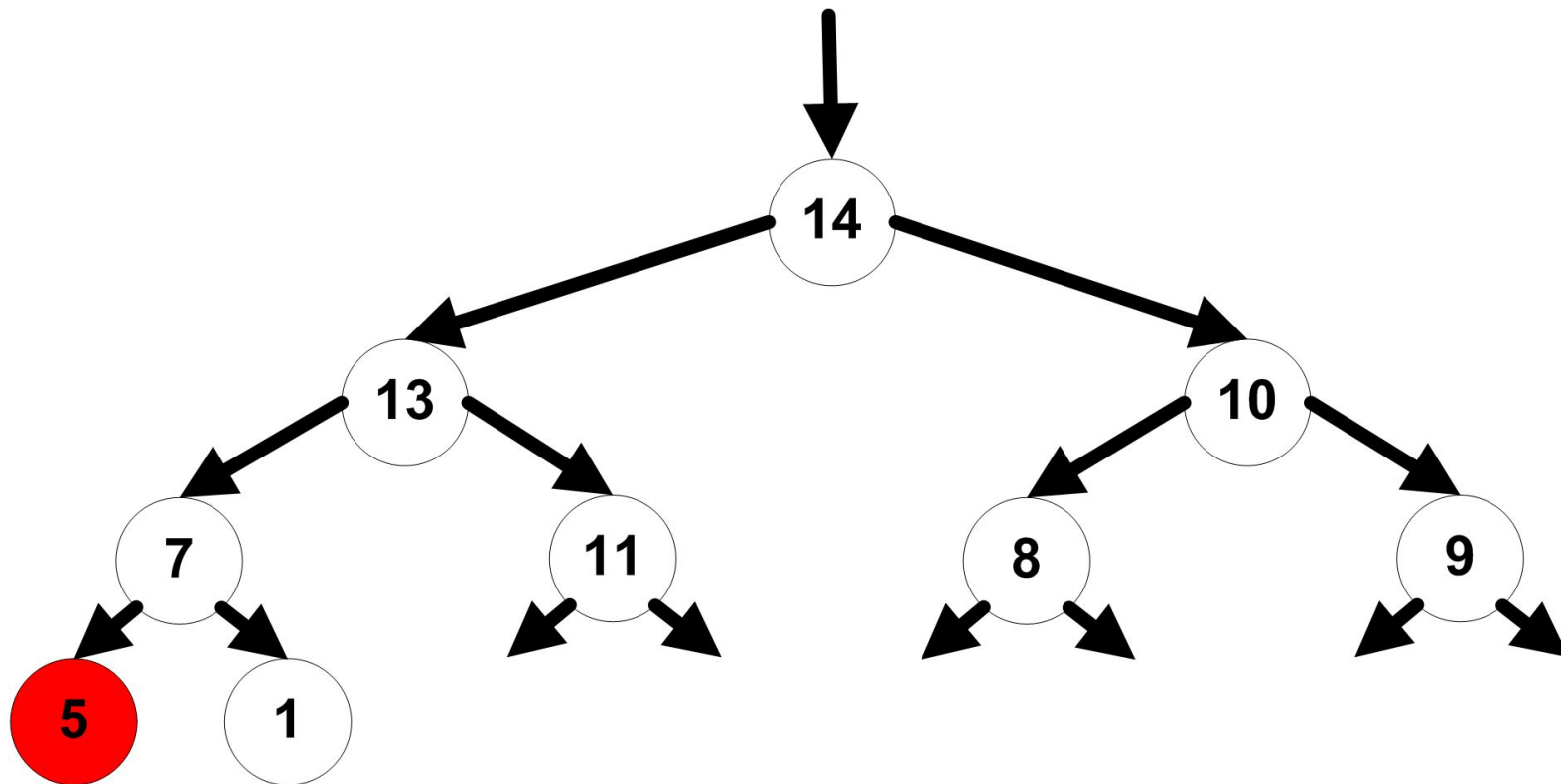
Exercício Resolvido (7)

- Remova a raiz do heap abaixo



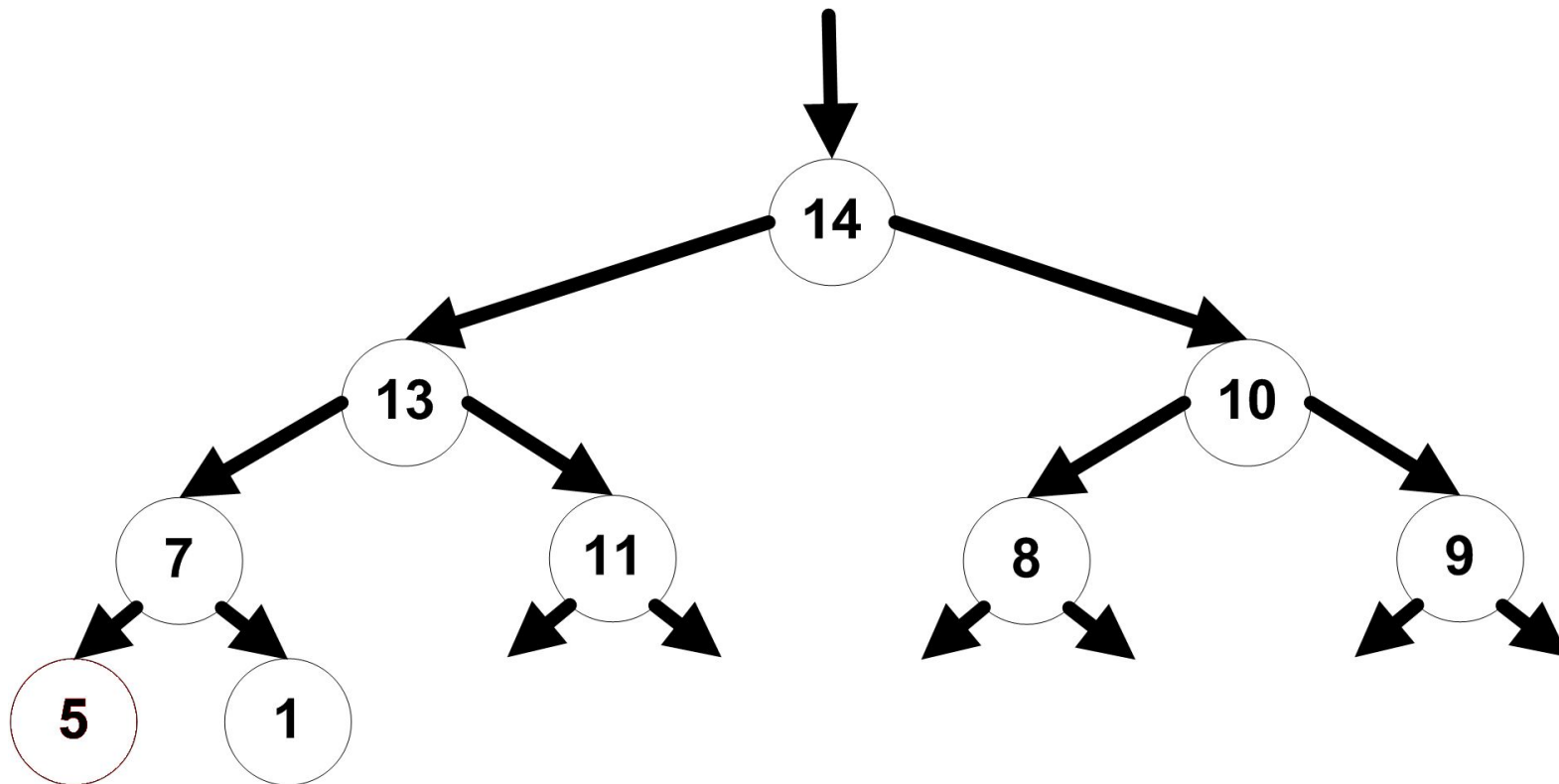
Exercício Resolvido (7)


- Remova a raiz do heap abaixo



Exercício (3)

- Remova a raiz no heap abaixo. Em seguida, remova novamente a raiz



- Definição de Heap
- Funcionamento básico
- **Algoritmo em C *like*** ← 
- Análise dos número de comparações e movimentações

Algoritmo em C *like*

```
void heapsort() {  
  
    //Construção do heap  
    for (int tam = 2; tam <= n; tam++){  
        construir(tam);  
    }  
  
    //Ordenação propriamente dita  
    int tam = n;  
    while (tam > 1){  
        swap(1, tam--);  
        reconstruir(tam);  
    }  
}
```

```
void construir(int tam){  
    . . .  
}
```

```
void reconstruir(int tam){  
    . . .  
}
```

Algoritmo em C *like*

```
void heapsort() {  
  
    //Construção do heap  
    for (int tam = 2; tam <= n; tam++){  
        construir(tam);  
    }  
    ...  
}
```

```
void construir(int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

101	115	30	63	47	20
1	2	3	4	5	6

Algoritmo em C *like*

```
void heapsort() {
```

```
    //Construção do heap
```

```
    for (int tam = 2; tam <= n; tam++){
```

```
        construir(tam);
```

```
    }
```

```
        . . .
```

```
void construir(int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
```

```
        swap(i, i/2);
```

```
    }
```

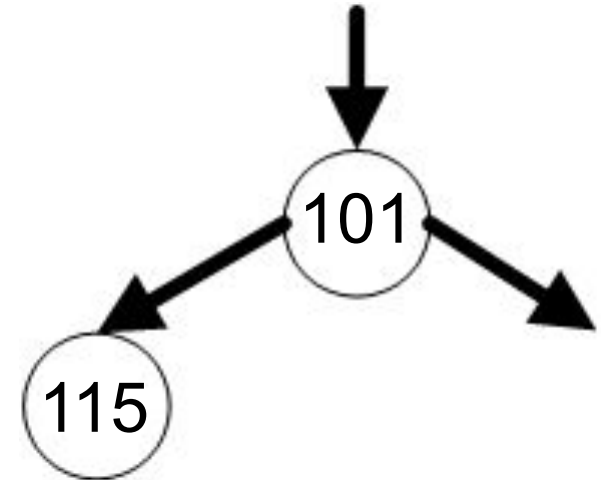
```
}
```

101	115	30	63	47	20
1	2	3	4	5	6

Algoritmo em C *like*

```
void heapsort() {
    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
}
```

```
void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```



101	115	30	63	47	20
1	2	3	4	5	6

Algoritmo em C *like*

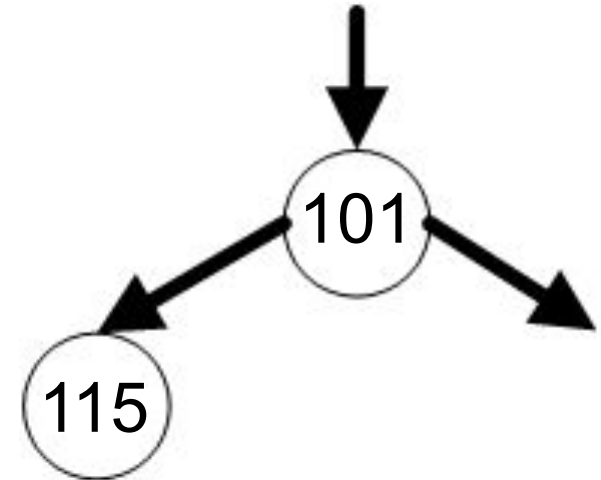
```
void heapsort() {           true: 2 <= 6
```

```
    //Construção do heap
```

```
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
```

```
void construir(int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```

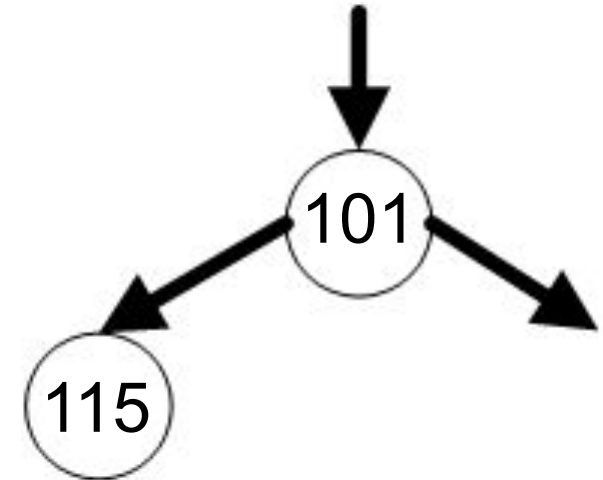


101	115	30	63	47	20
1	2	3	4	5	6

Algoritmo em C *like*

```
void heapsort() {
    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
}
```

```
void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```

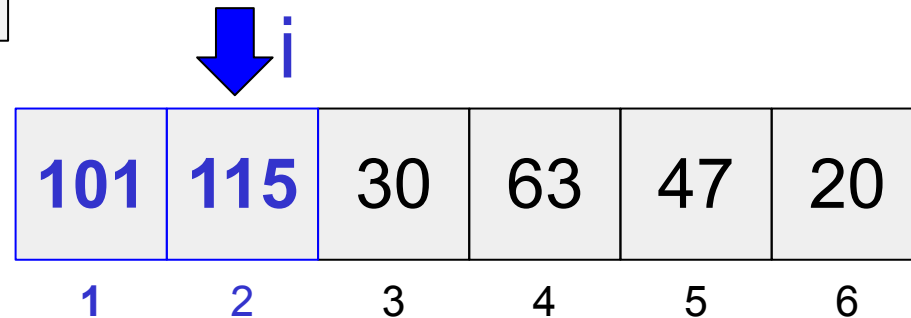
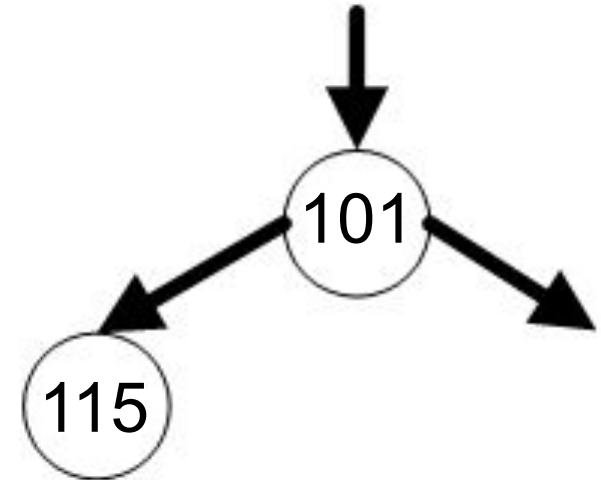


101	115	30	63	47	20
1	2	3	4	5	6

Algoritmo em C *like*

```
void heapsort() {
    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
}
```

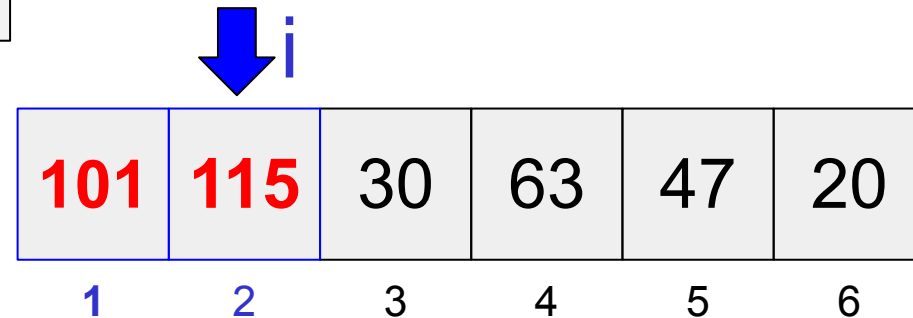
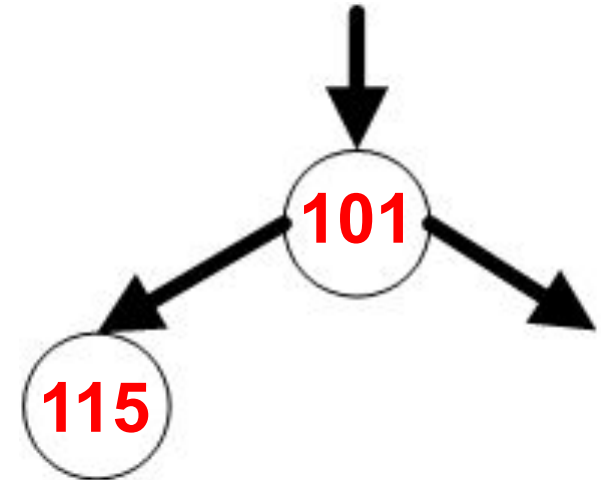
```
void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```



Algoritmo em C *like*

```
void heapsort() {
    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
}
```

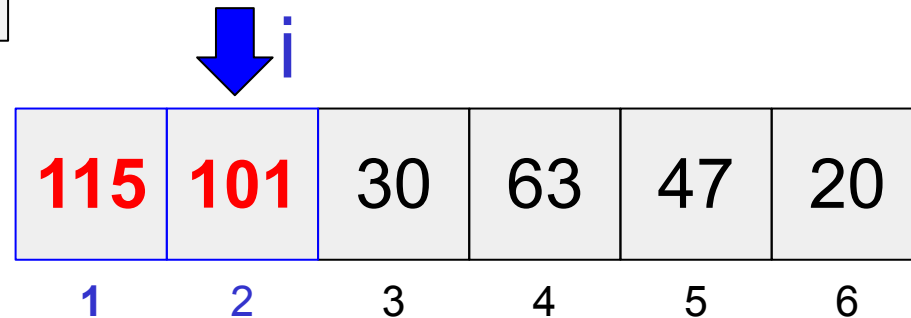
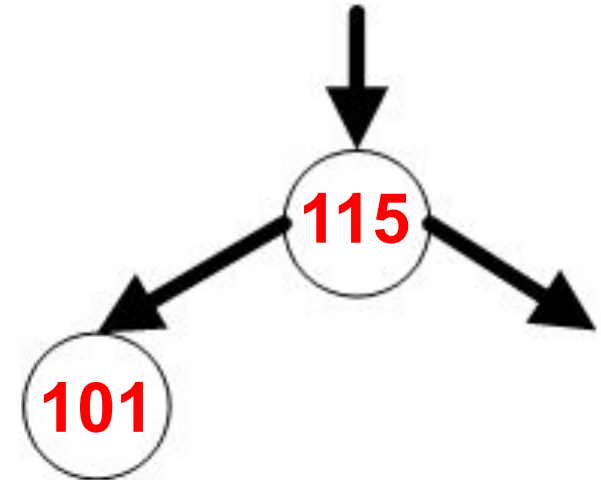
```
void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
    true: 2 > 1 && 115 > 101
}
```



Algoritmo em C *like*

```
void heapsort() {
    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
}
```

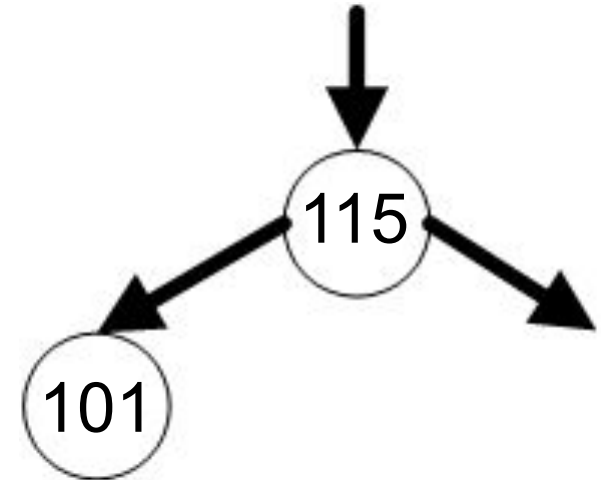
```
void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```



Algoritmo em C *like*

```
void heapsort() {
    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
}
```

```
void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```

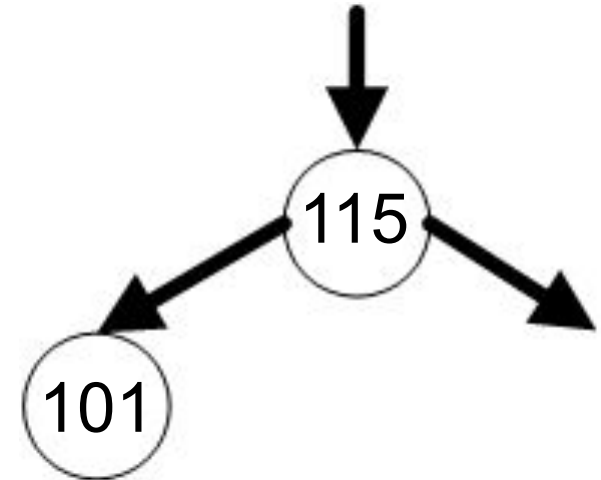


115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo em C *like*

```
void heapsort() {
    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
}
```

```
void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
    // false: 1 > 1 && ...
}
```

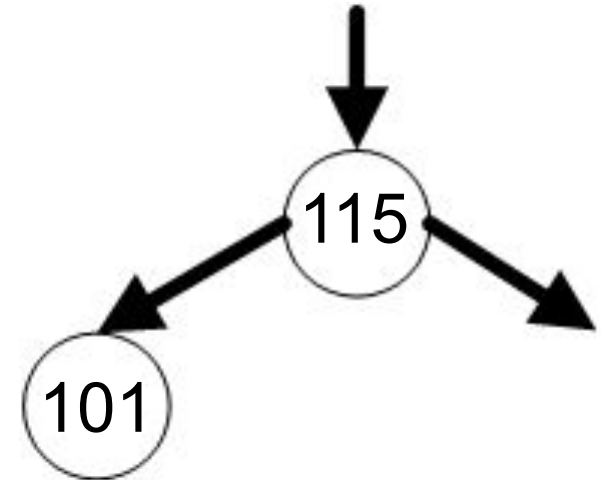


115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo em C *like*

```
void heapsort() {
    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
}
```

```
void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```

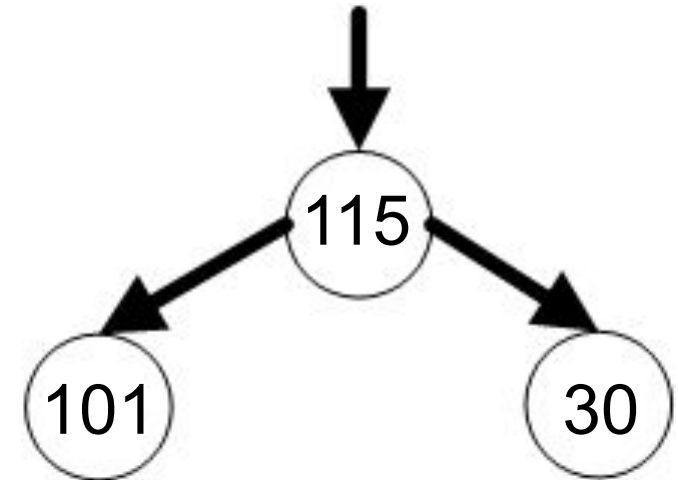


115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo em C *like*

```
void heapsort() {
    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
}
```

```
void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo em C *like*

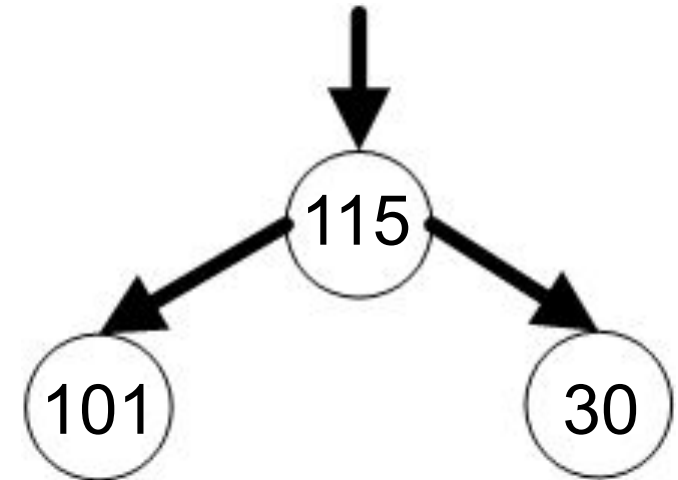
```
void heapsort() {           true: 3 <= 6
```

```
    //Construção do heap
```

```
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
```

```
void construir(int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo em C *like*

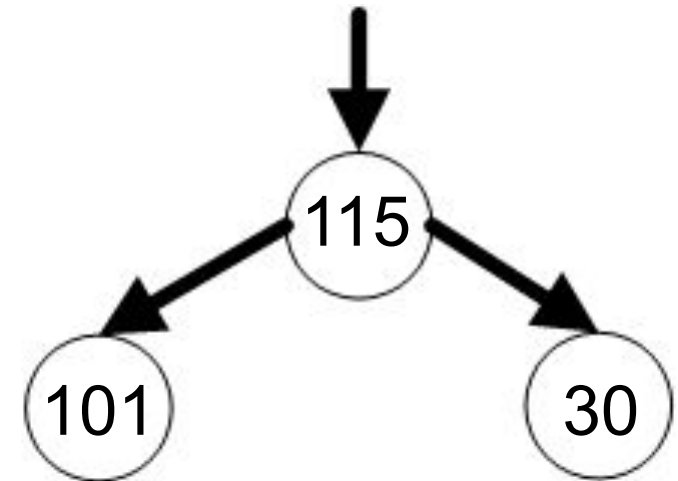
```

void heapsort() {

    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
}
    
```

```

void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
    
```

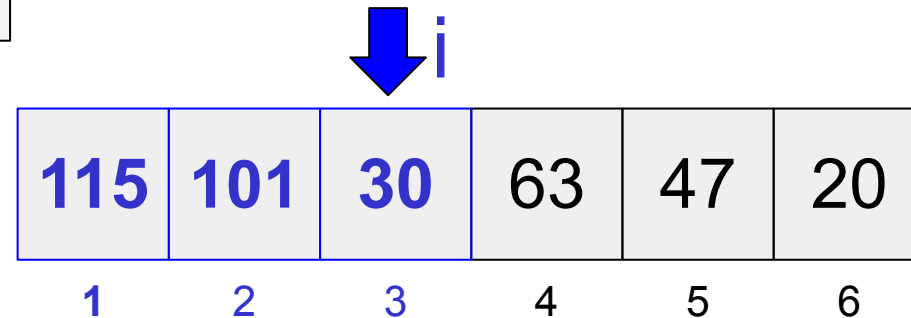
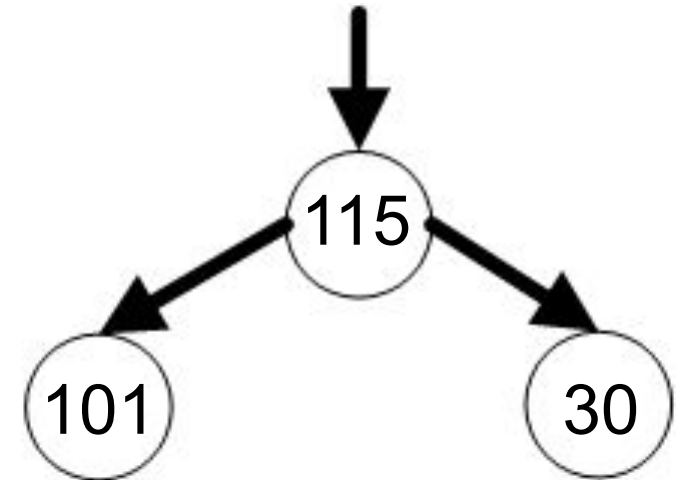


115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo em C *like*

```
void heapsort() {
    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
}
```

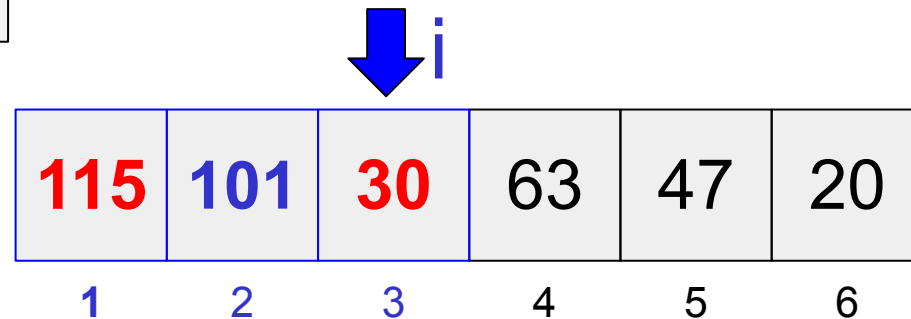
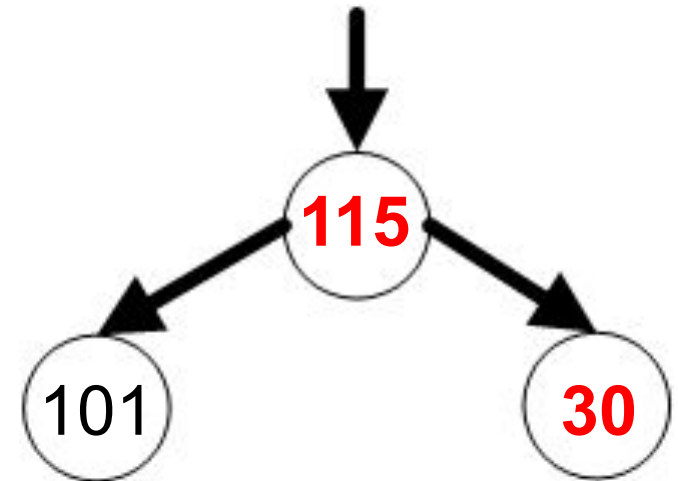
```
void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```



Algoritmo em C *like*

```
void heapsort() {
    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
}
```

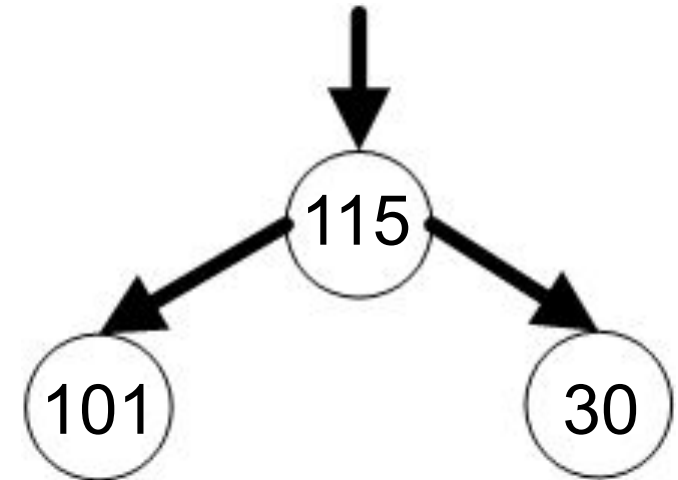
```
void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
false: 3 > 1 && 30 > 115
```



Algoritmo em C *like*

```
void heapsort() {
    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
}
```

```
void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```

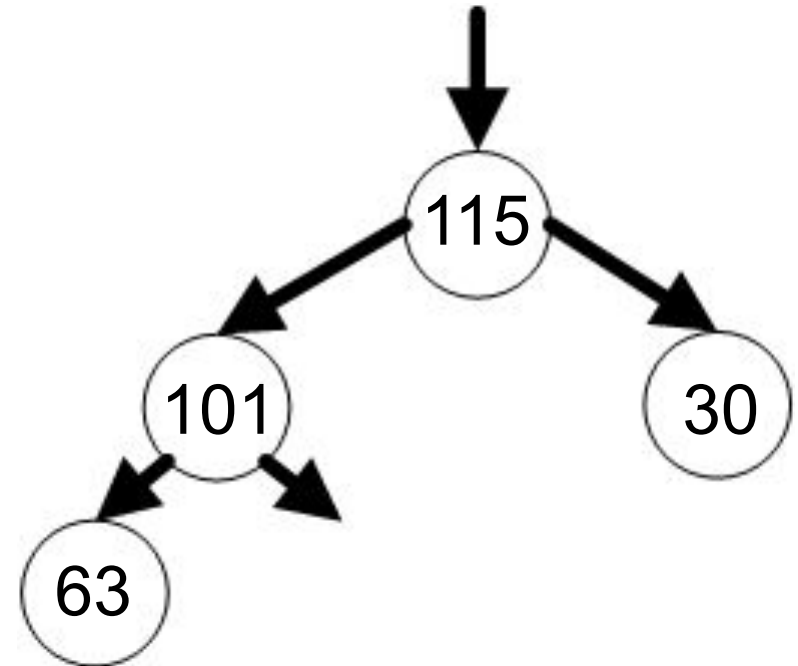


115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo em C *like*

```
void heapsort() {
    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
}
```

```
void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo em C *like*

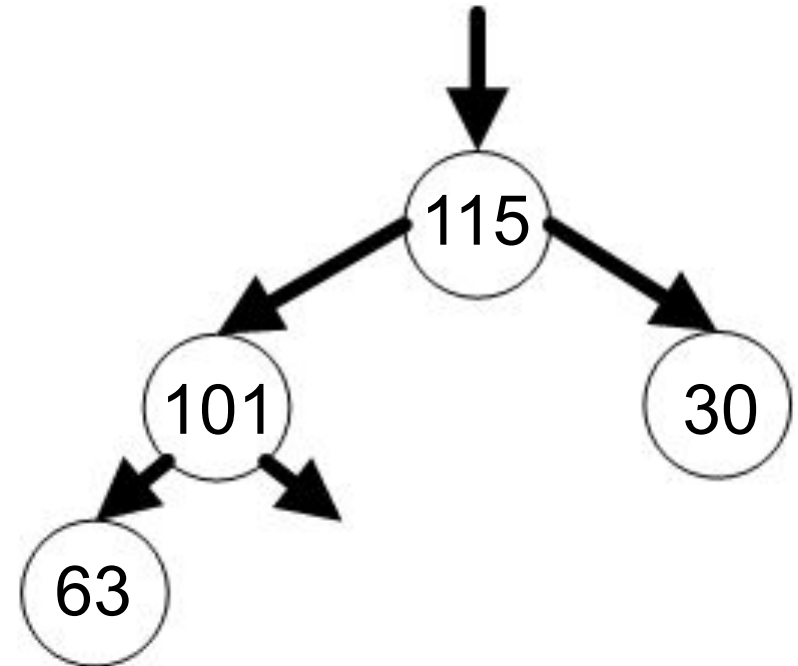
```
void heapsort() {           true: 4 <= 6
```

```
    //Construção do heap
```

```
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
```

```
void construir(int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo em C *like*

```
void heapsort() {
```

```
    //Construção do heap
```

```
    for (int tam = 2; tam <= n; tam++){
```

```
        construir(tam);
```

```
    }
```

```
        ...
```

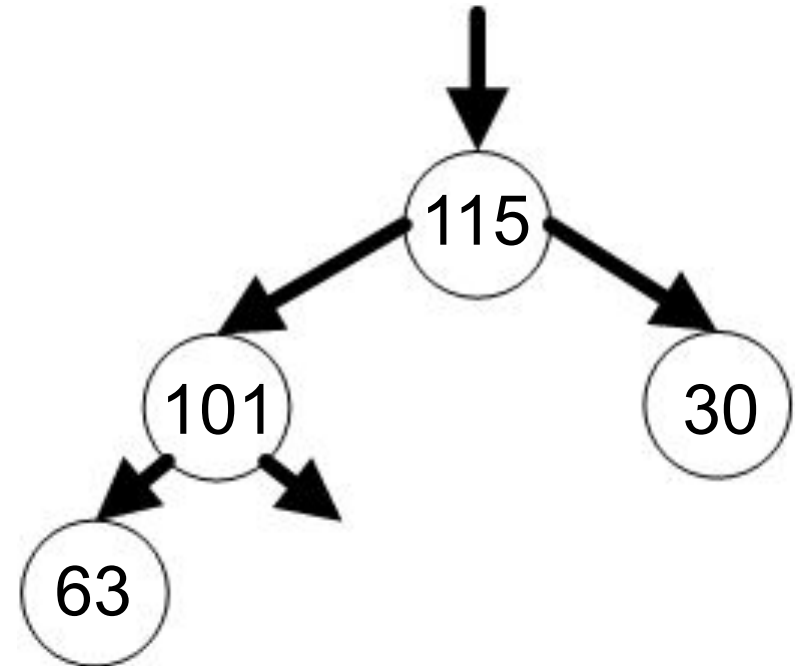
```
void construir(int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
```

```
        swap(i, i/2);
```

```
    }
```

```
}
```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo em C like

```
void heapsort() {
```

```
    //Construção do heap
```

```
    for (int tam = 2; tam <= n; tam++){
```

```
        construir(tam);
```

```
    }
```

```
        ...
```

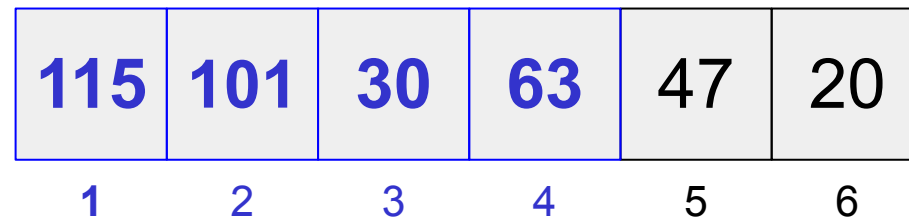
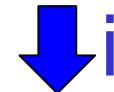
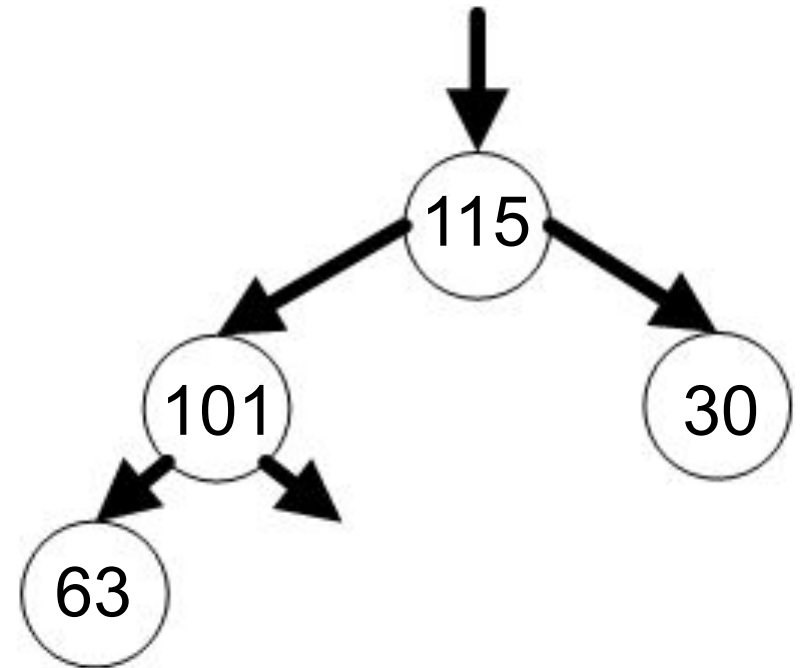
```
void construir(int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
```

```
        swap(i, i/2);
```

```
    }
```

```
}
```



Algoritmo em C like

```
void heapsort() {
```

```
    //Construção do heap
```

```
    for (int tam = 2; tam <= n; tam++){
```

```
        construir(tam);
```

```
    }
```

...

```
void construir(int tam){
```

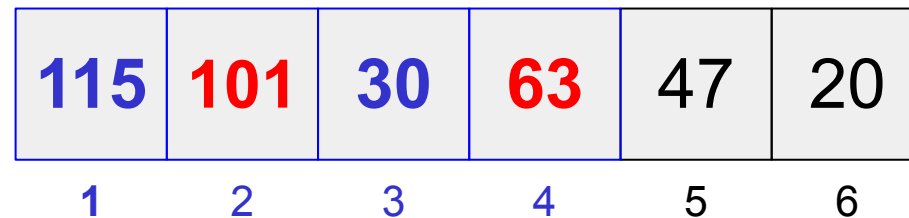
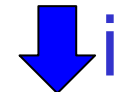
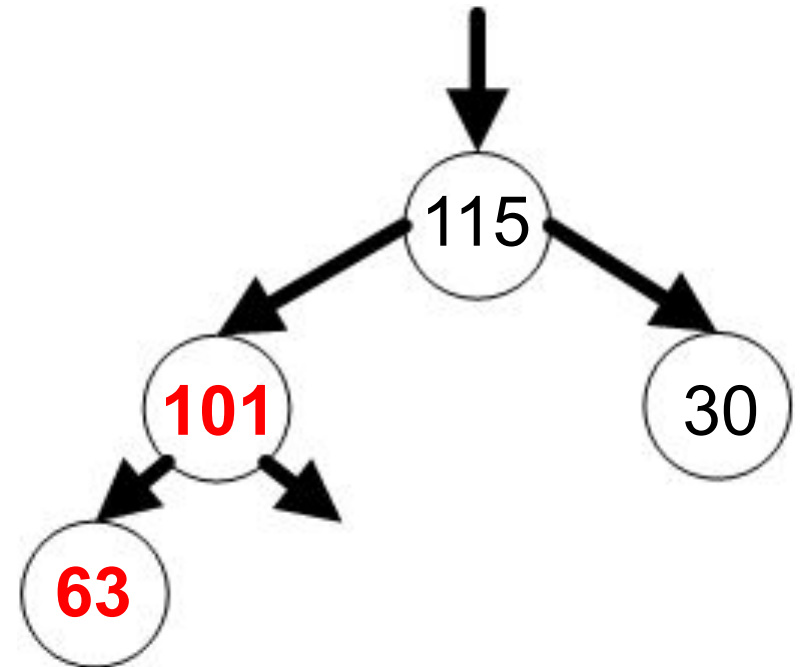
```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
```

```
        swap(i, i/2);
```

```
    }
```

```
}
```

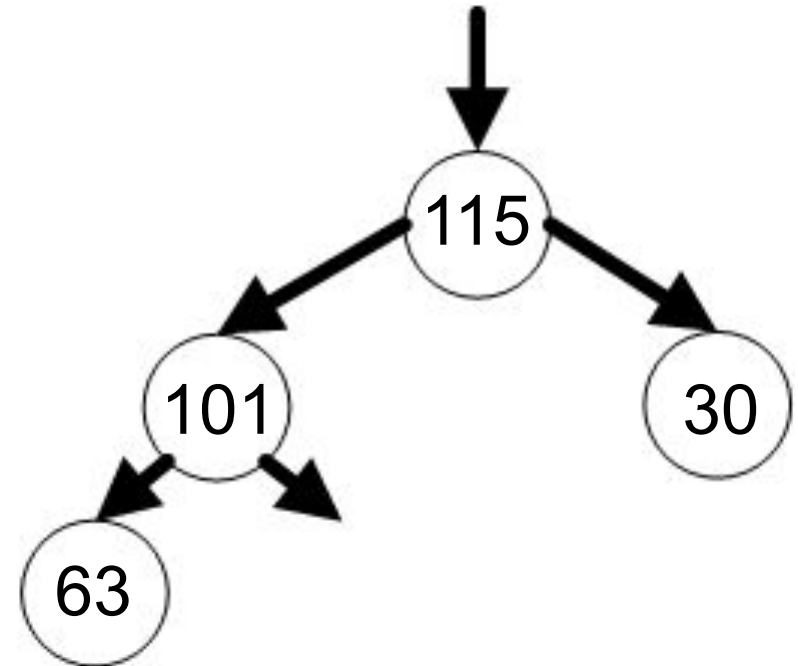
false: $4 > 1 \ \&\& \ 63 > 101$



Algoritmo em C *like*

```
void heapsort() {
    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
}
```

```
void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo em C *like*

```

void heapsort() {

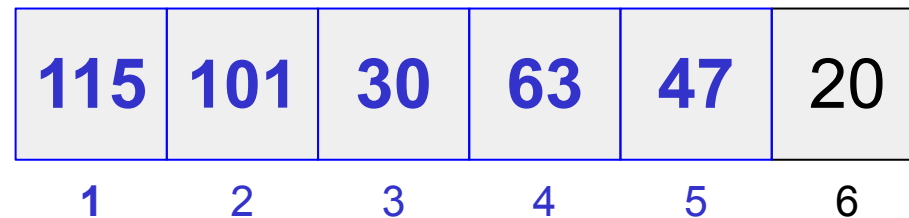
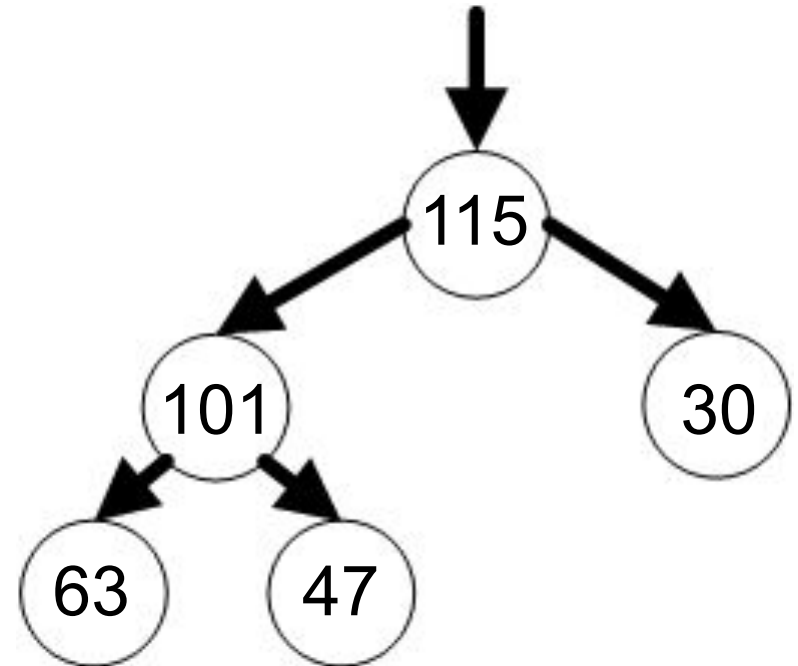
    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
}

```

```

void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}

```



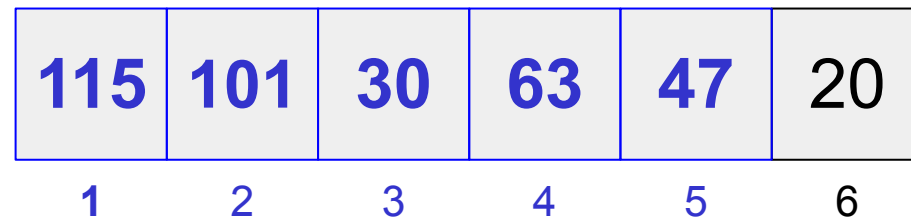
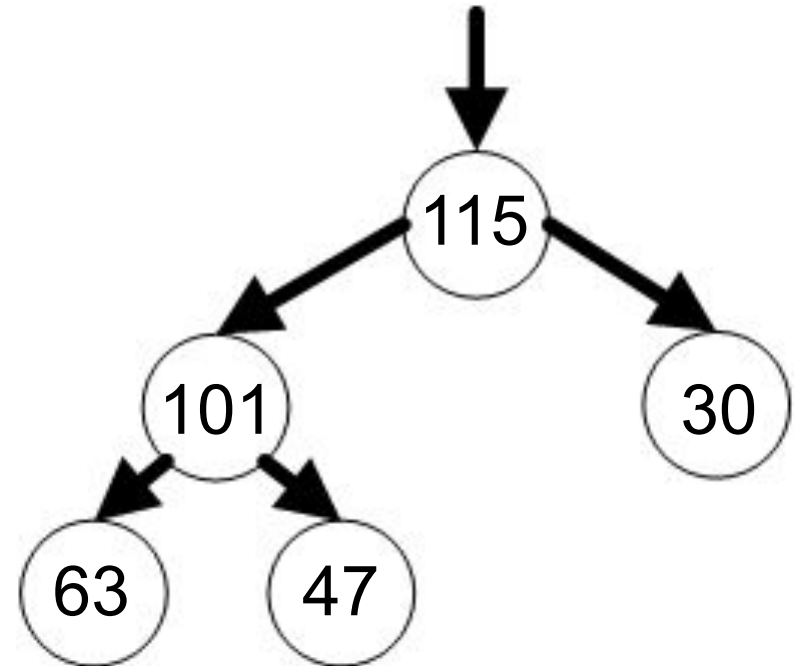
Algoritmo em C like

```
void heapsort() {           true: 5 <= 6
```

```
    //Construção do heap
```

```
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
```

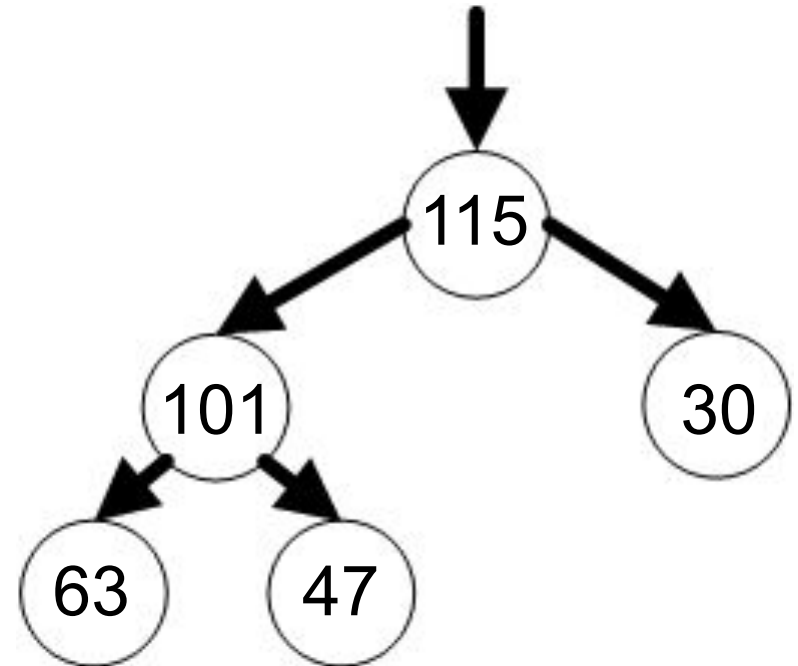
```
void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```



Algoritmo em C *like*

```
void heapsort() {
    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
}
```

```
void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```

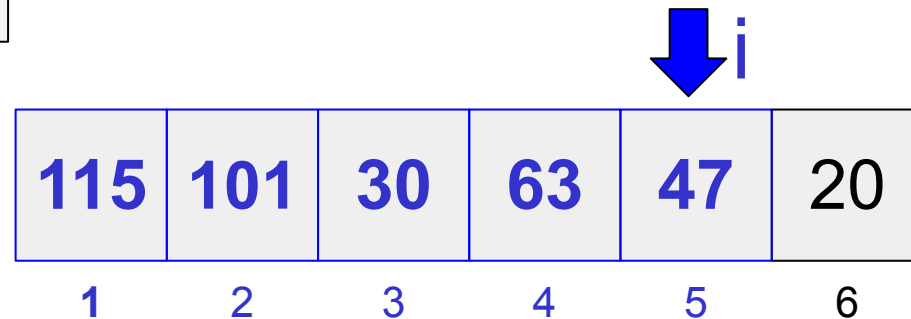
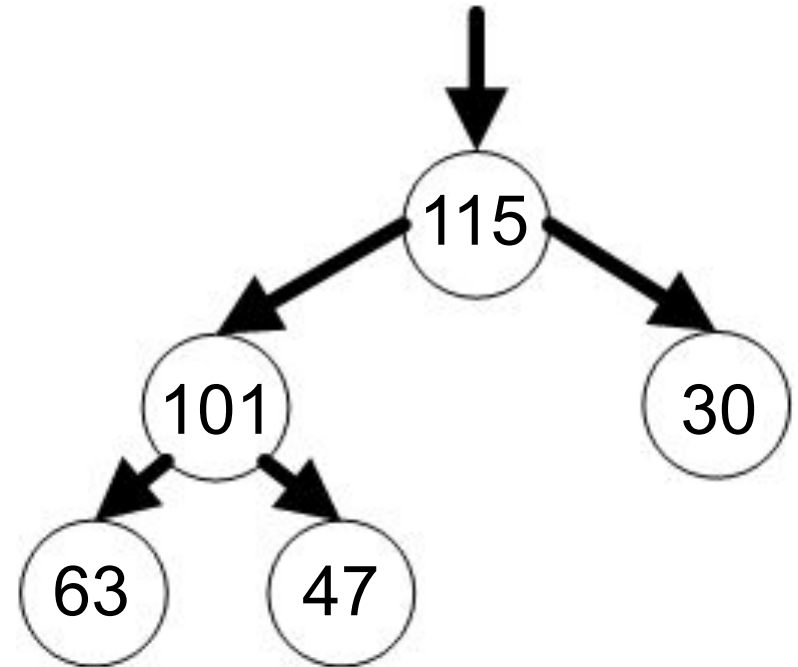


115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo em C like

```
void heapsort() {
    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
}
```

```
void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```



Algoritmo em C *like*

```
void heapsort() {
```

```
    //Construção do heap
```

```
    for (int tam = 2; tam <= n; tam++){
```

```
        construir(tam);
```

```
    }
```

```
        ...
```

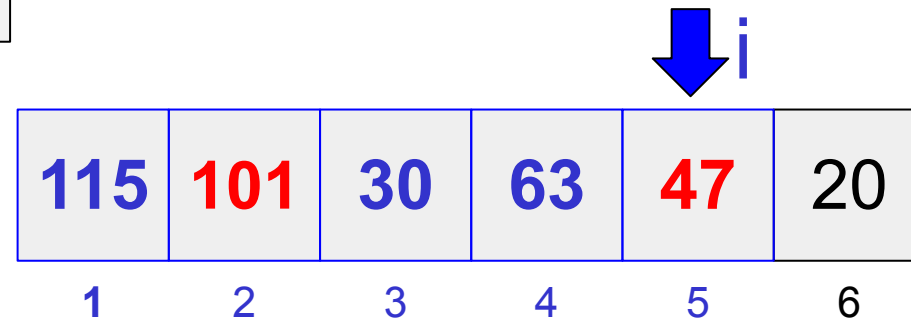
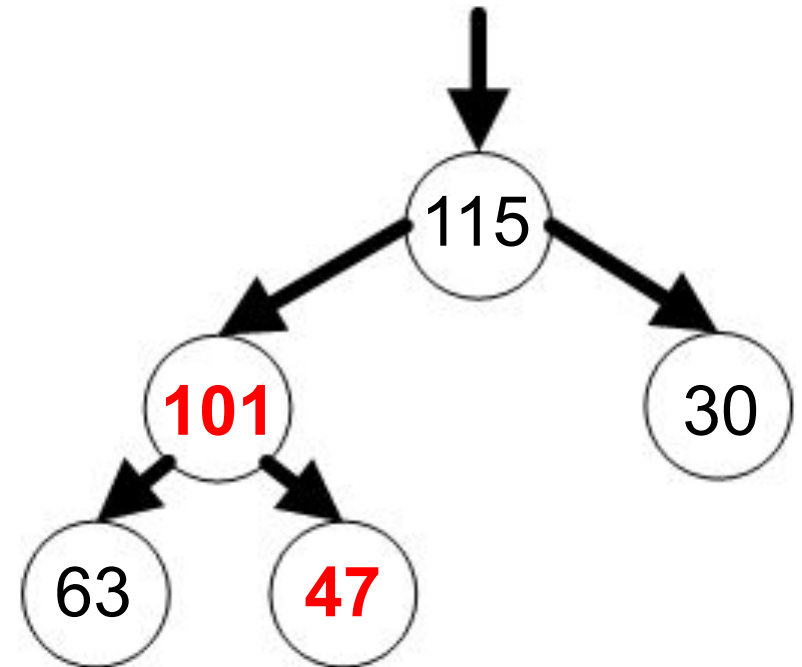
```
void construir(int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
```

```
        swap(i, i/2);
```

```
    }
```

```
    }
    false: 5 > 1 && 47 > 101
```



Algoritmo em C like

```
void heapsort() {
```

```
    //Construção do heap
```

```
    for (int tam = 2; tam <= n; tam++){
```

```
        construir(tam);
```

```
    }
```

```
        ...
```

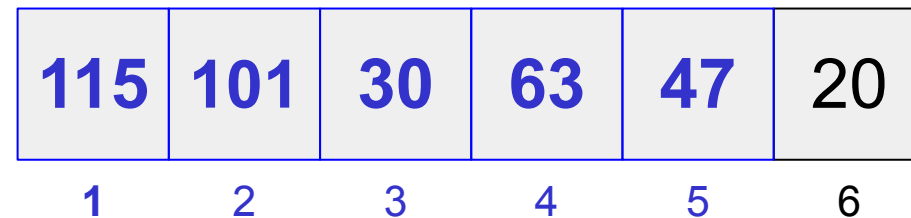
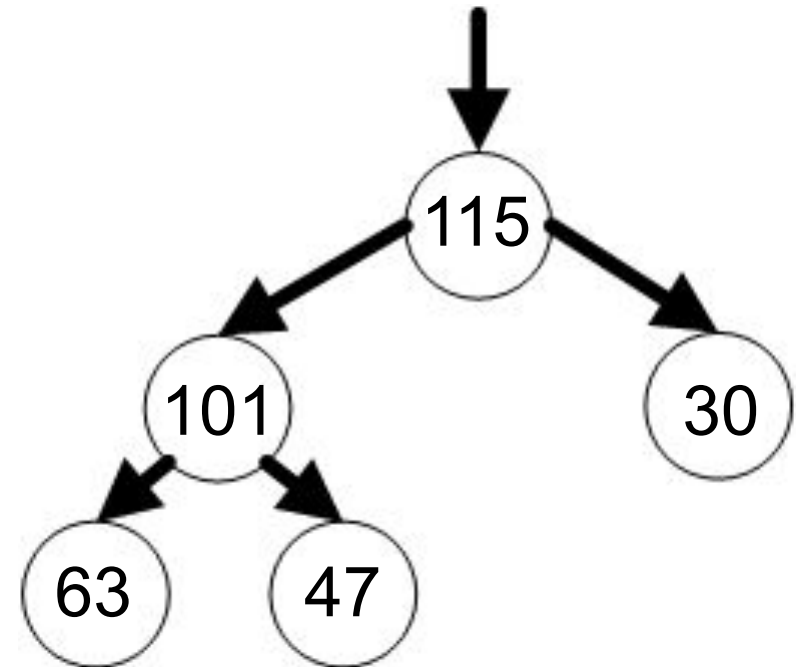
```
void construir(int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
```

```
        swap(i, i/2);
```

```
    }
```

```
}
```



Algoritmo em C *like*

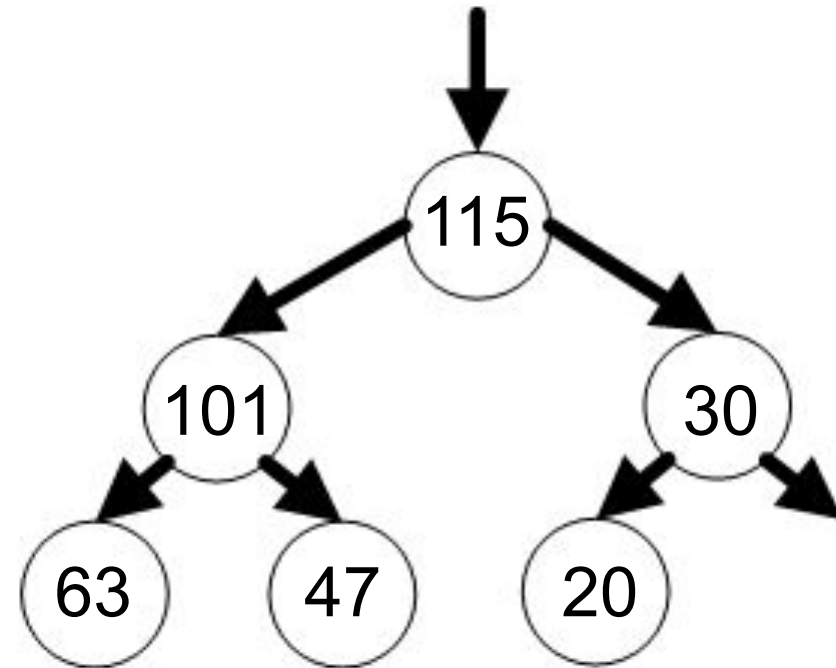
```

void heapsort() {

    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
}
    
```

```

void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
    
```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo em C *like*

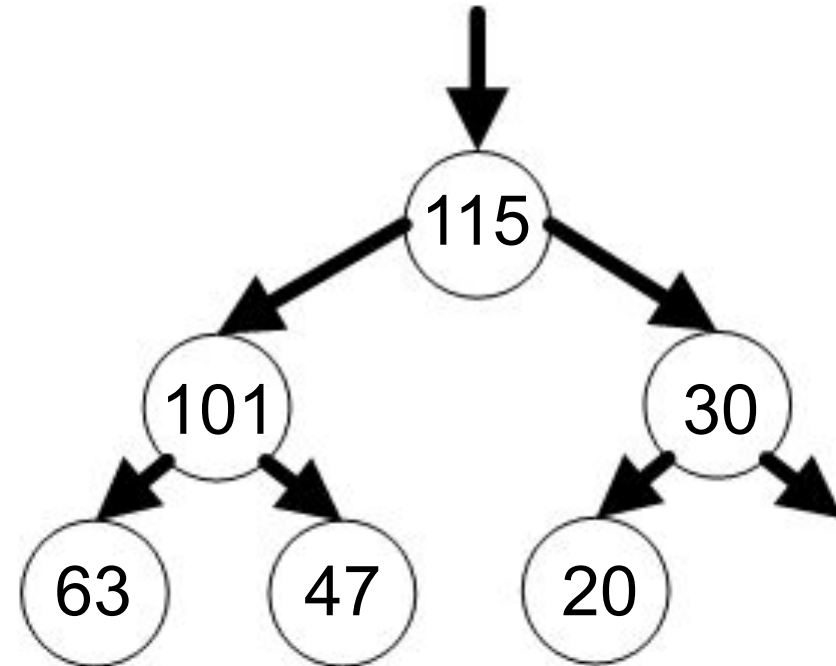
```
void heapsort() {           true: 6 <= 6
```

```
    //Construção do heap
```

```
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
```

```
void construir(int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo em C *like*

```
void heapsort() {
```

```
    //Construção do heap
```

```
    for (int tam = 2; tam <= n; tam++){
```

```
        construir(tam);
```

```
    }
```

```
        ...
```

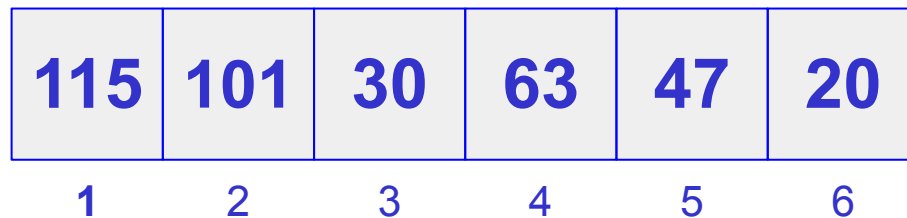
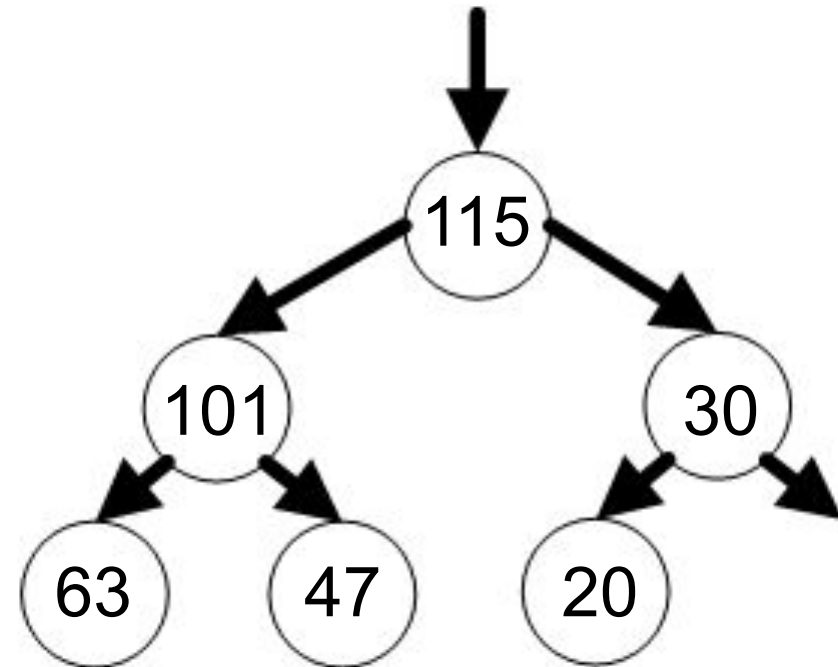
```
void construir(int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
```

```
        swap(i, i/2);
```

```
    }
```

```
}
```



Algoritmo em C like

```
void heapsort() {
```

```
    //Construção do heap
```

```
    for (int tam = 2; tam <= n; tam++){
```

```
        construir(tam);
```

```
    }
```

```
        ...
```

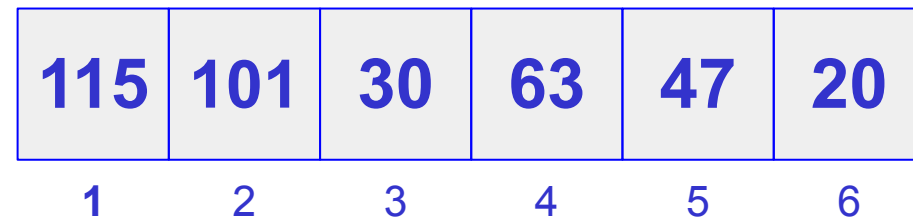
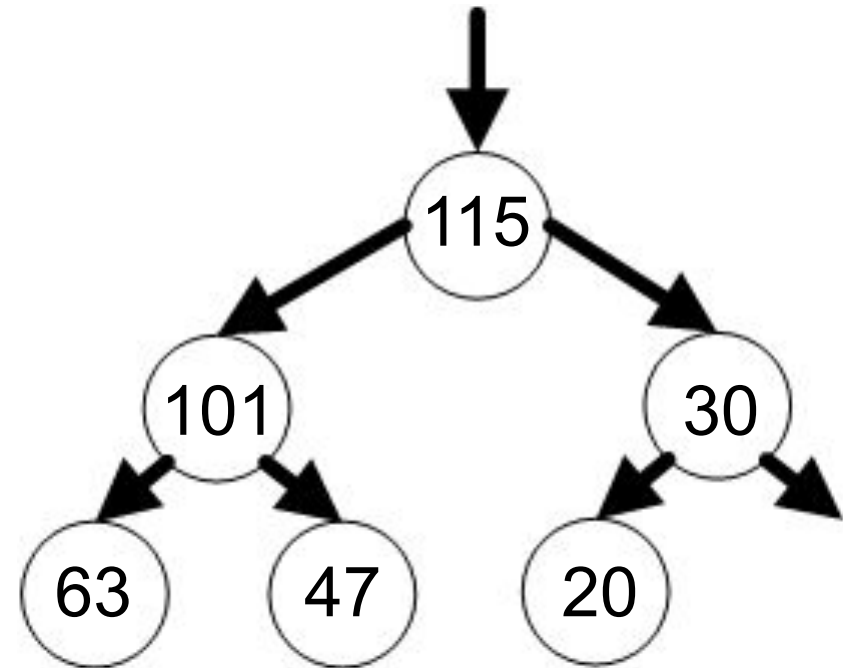
```
void construir(int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
```

```
        swap(i, i/2);
```

```
    }
```

```
}
```



Algoritmo em C like

```
void heapsort() {
```

```
    //Construção do heap
```

```
    for (int tam = 2; tam <= n; tam++){
```

```
        construir(tam);
```

```
    }
```

...

```
void construir(int tam){
```

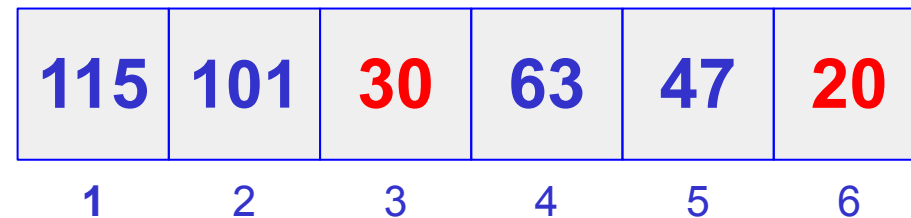
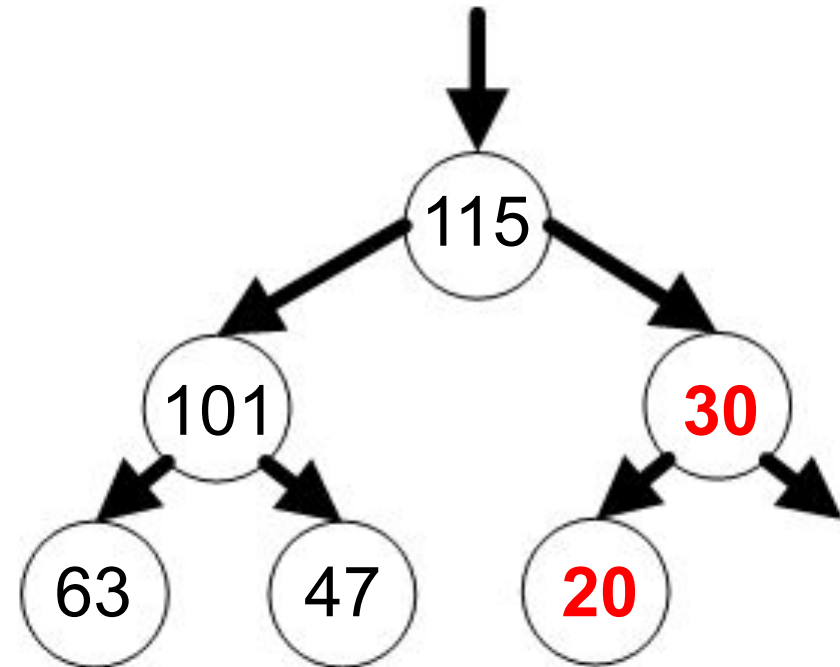
```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
```

```
        swap(i, i/2);
```

```
    }
```

```
}
```

false: $6 > 1 \ \&\& \ 20 > 30$



Algoritmo em C *like*

```
void heapsort() {
```

```
    //Construção do heap
```

```
    for (int tam = 2; tam <= n; tam++){
```

```
        construir(tam);
```

```
    }
```

```
        ...
```

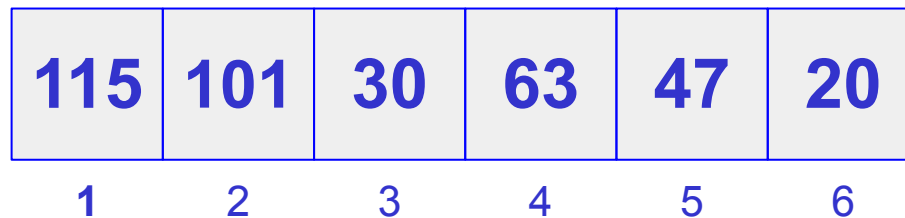
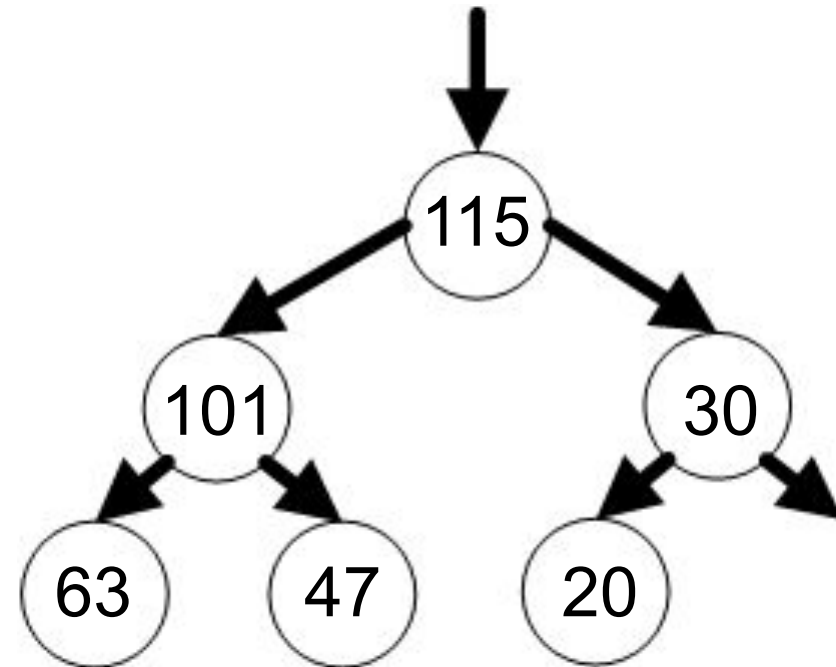
```
void construir(int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
```

```
        swap(i, i/2);
```

```
    }
```

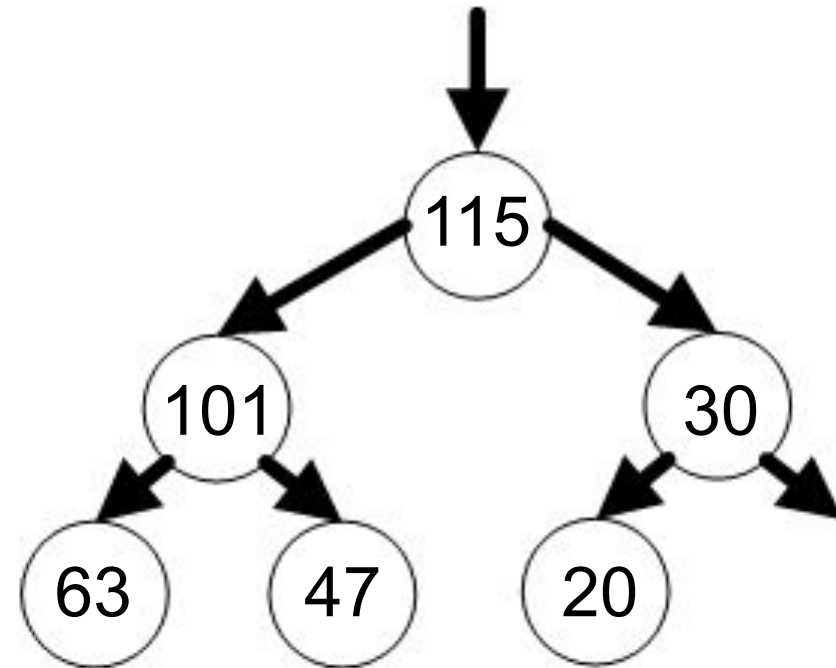
```
}
```



Algoritmo em C *like*

```
void heapsort() {
    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
}
```

```
void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo em C *like*

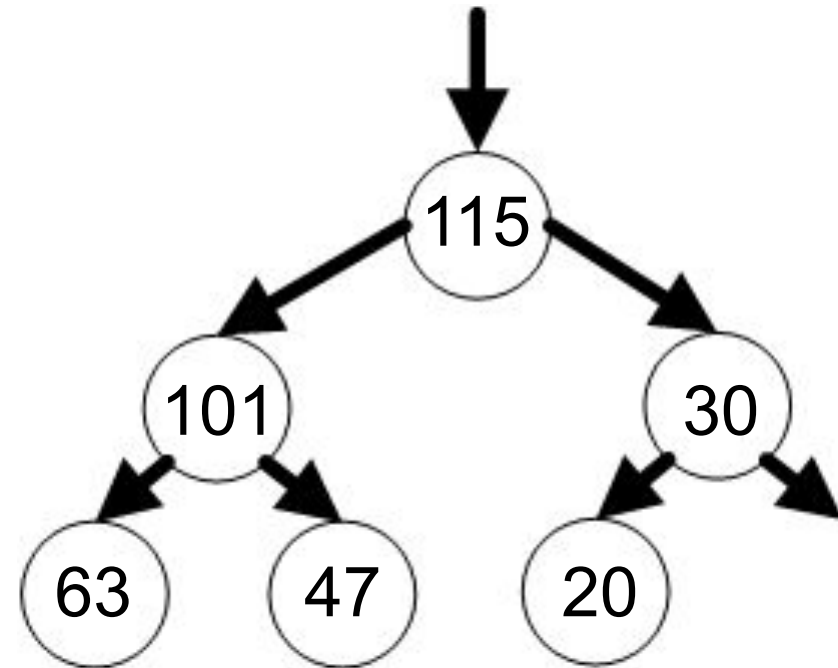
```
void heapsort() {           false: 7 <= 6
```

```
    //Construção do heap
```

```
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
```

```
void construir(int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo em C *like*

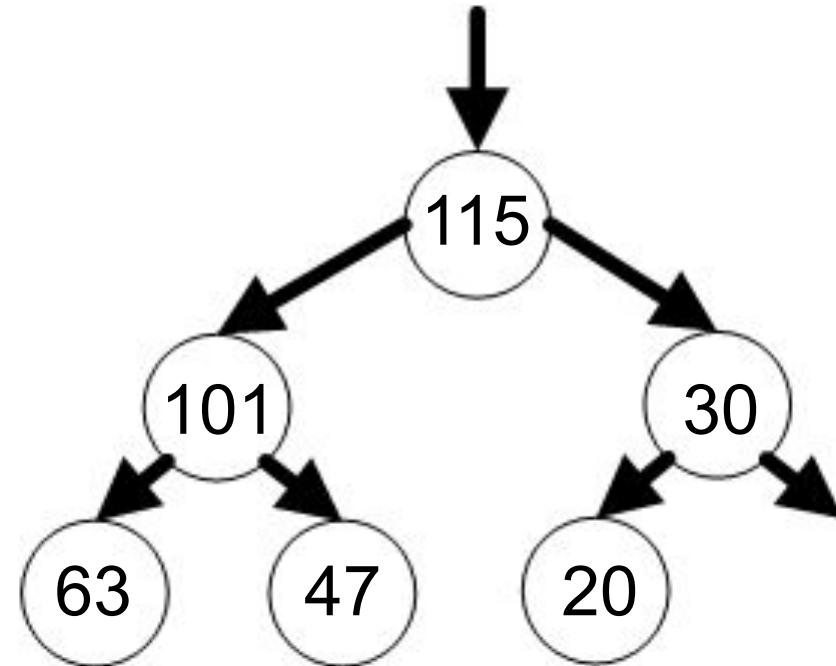
```

void heapsort() {

    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
    ...
}
    
```

```

void construir(int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
    
```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo em C *like*

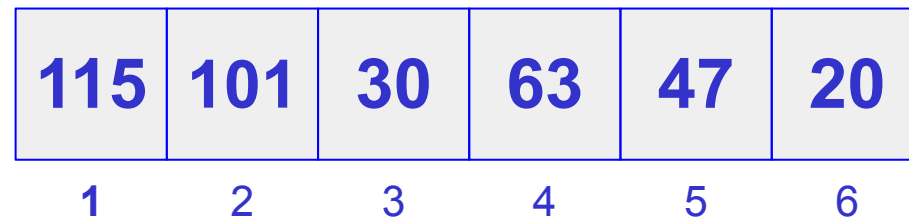
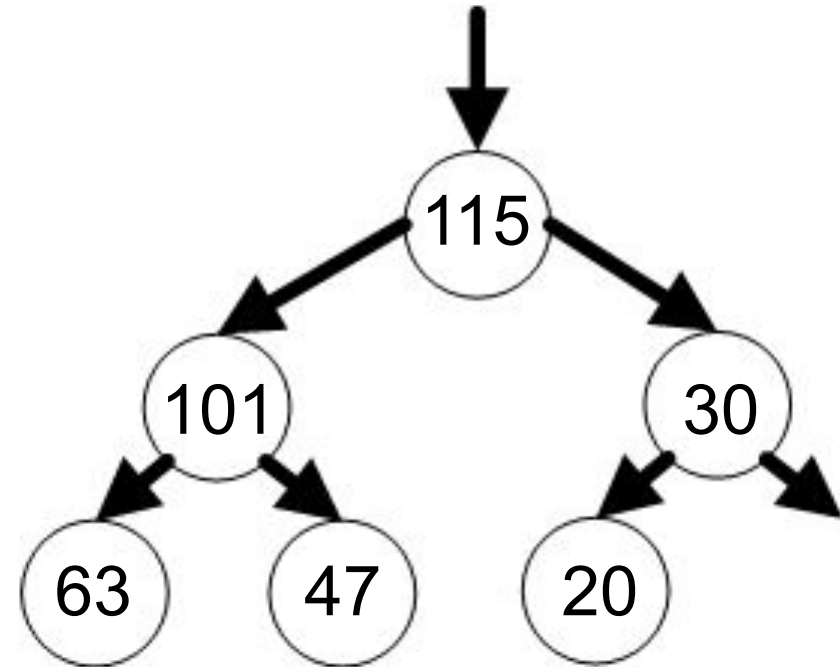
```

void heapsort() {

    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
}

```

...

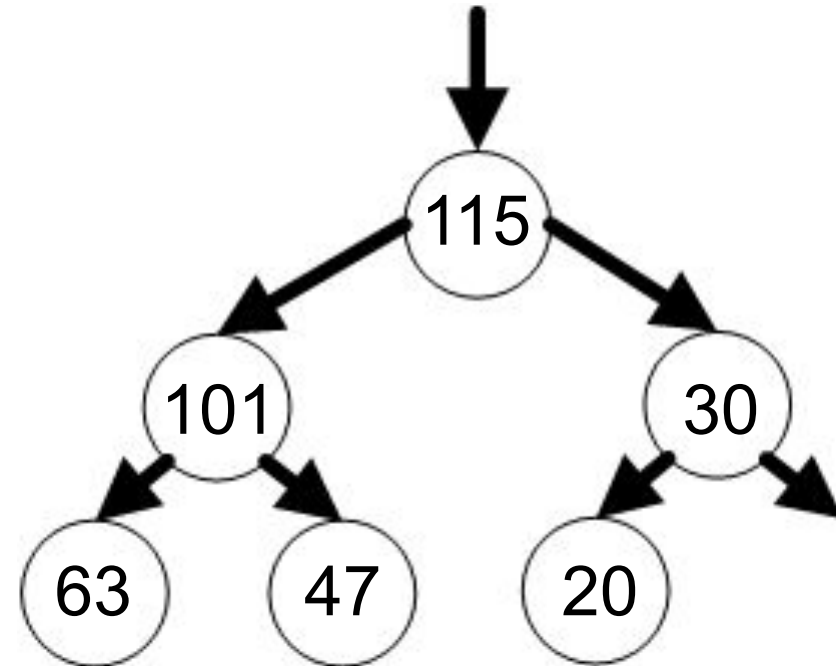


Algoritmo em C *like*

```
void heapsort() {
    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
}
```

//Ordenação propriamente dita

```
int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}
}
```



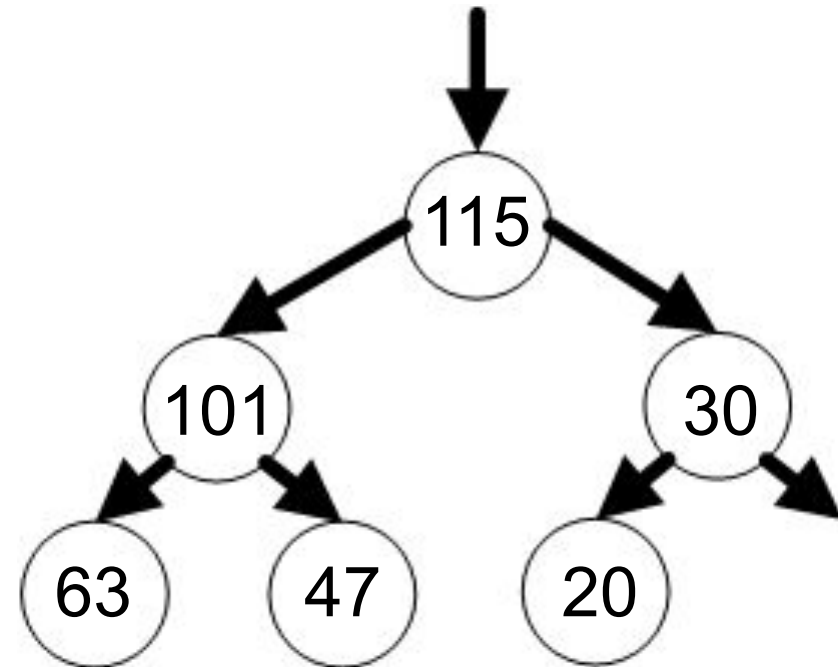
115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo em C *like*

```
void heapsort() {
    //Construção do heap
    for (int tam = 2; tam <= n; tam++){
        construir(tam);
    }
}
```

//Ordenação propriamente dita

```
int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}
}
```



115	101	30	63	47	20
1	2	3	4	5	6

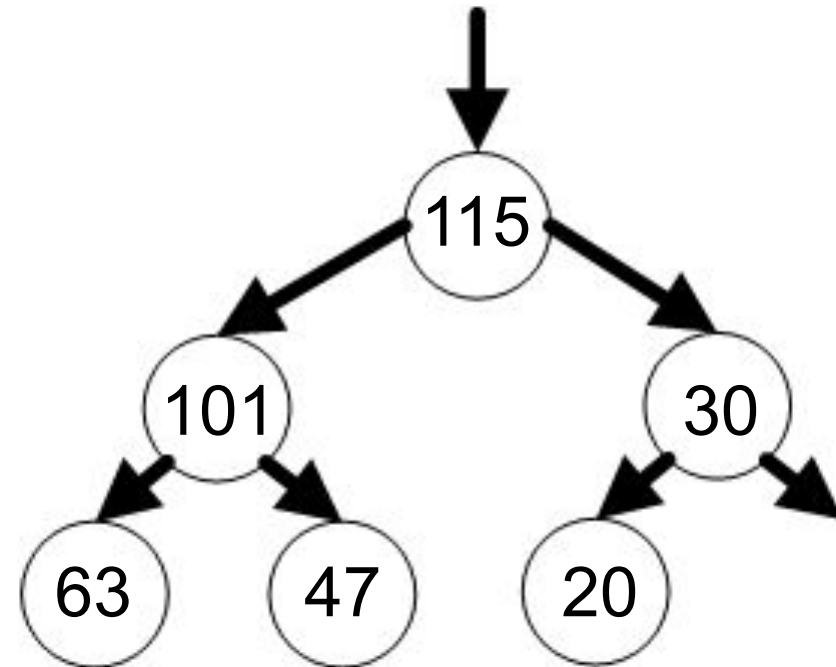
Algoritmo em C like

```

...
int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}
}
    
```

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}
    
```

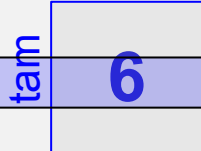


115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo em C like

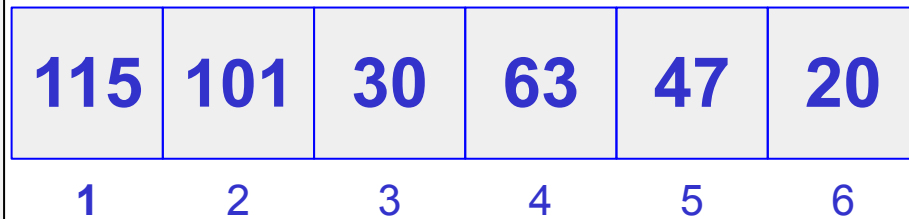
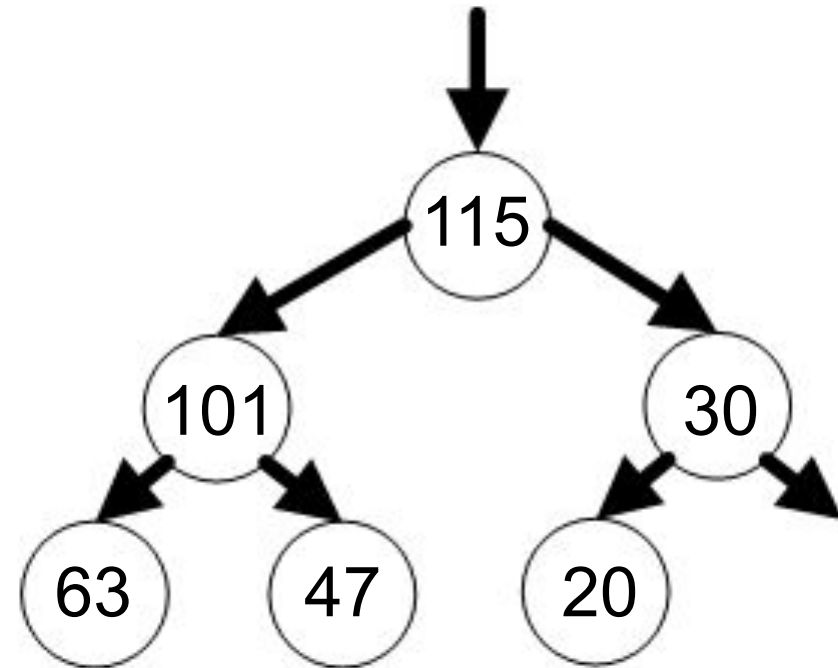
```

    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```



```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```



Algoritmo em C like

```

...
int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}

```

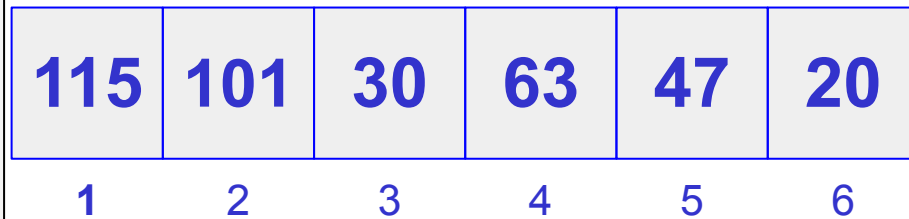
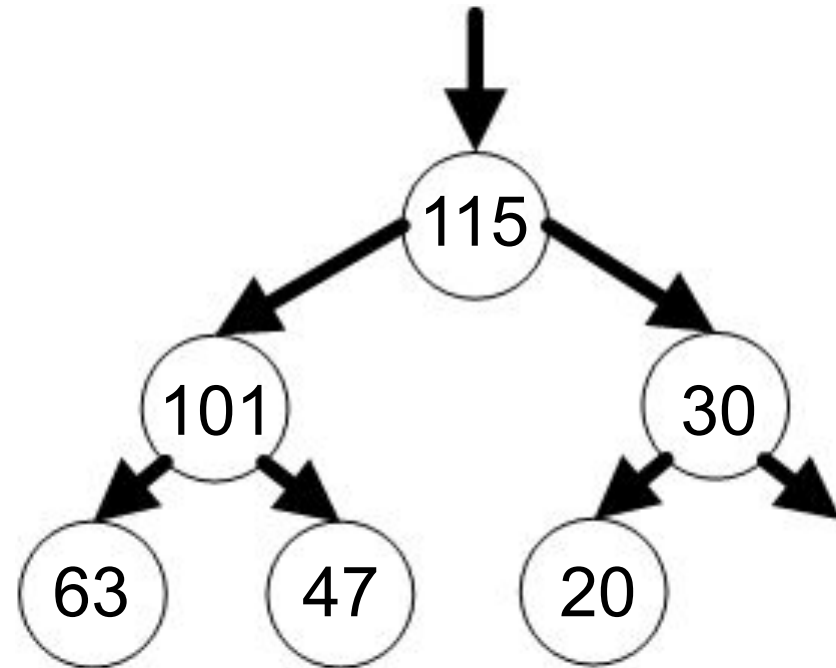
tam **6**

true: $6 > 1$

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



Algoritmo em C like

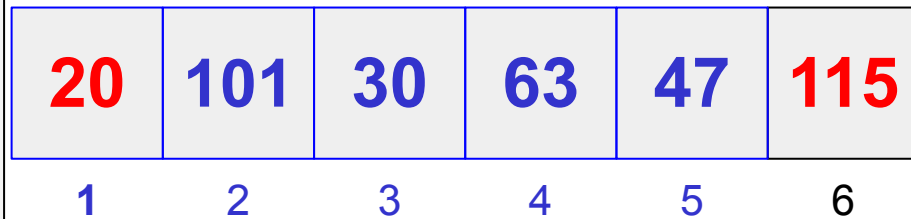
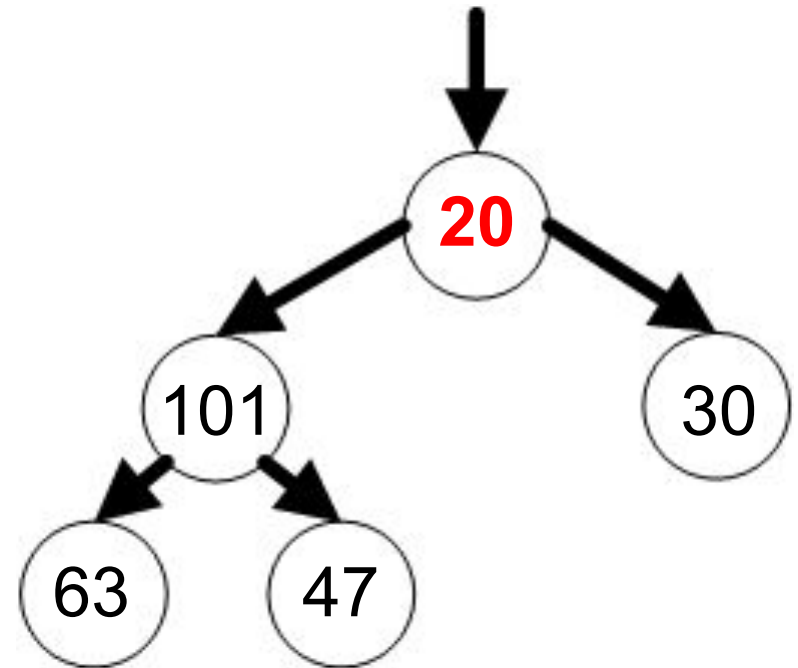
```

    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```



```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```



Algoritmo em C like

```

    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
}

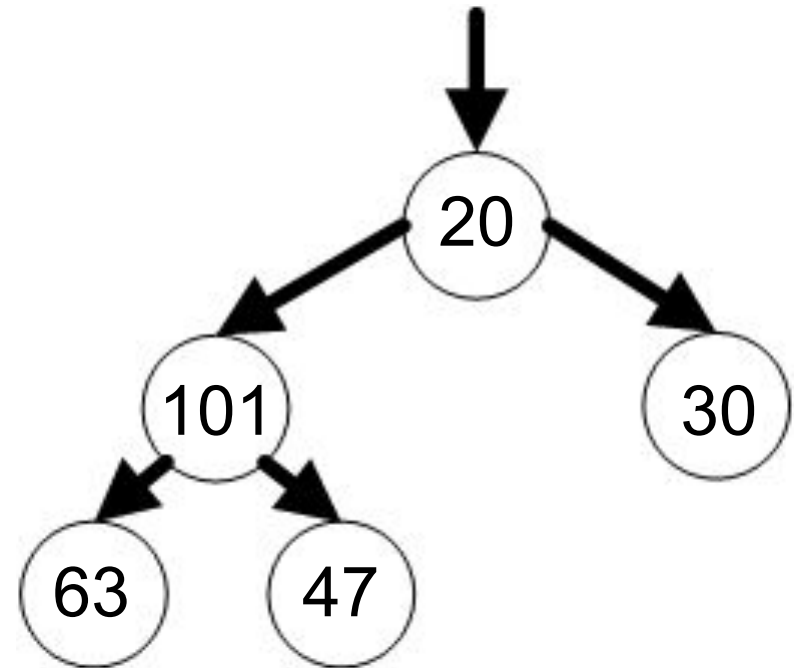
```

tam **5**

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



20	101	30	63	47	115
1	2	3	4	5	6

Algoritmo em C like

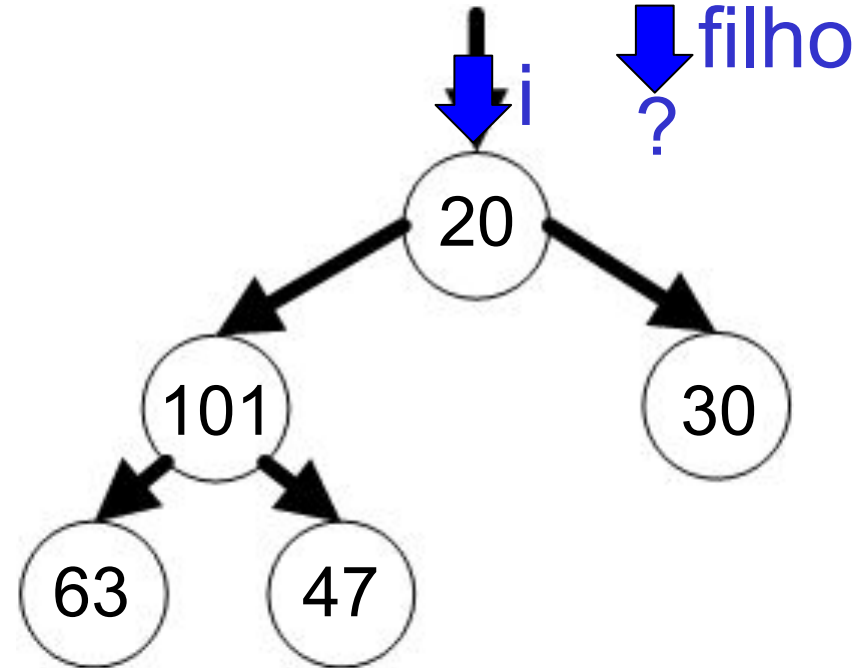
```

...
int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}
}
    
```

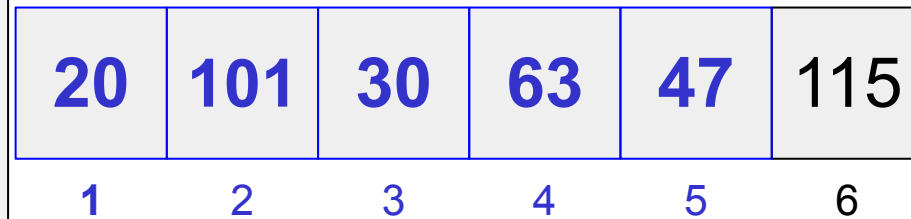
tam **5**

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}
    
```



i



filho
?

Algoritmo em C like

```

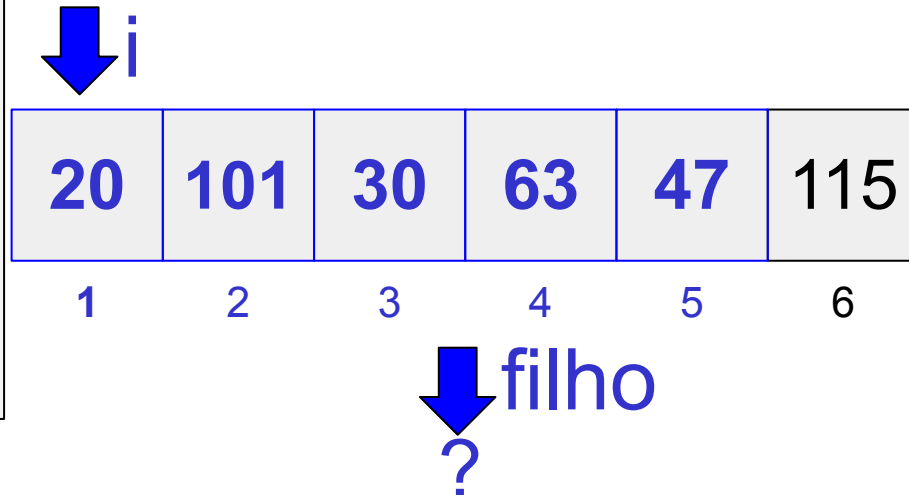
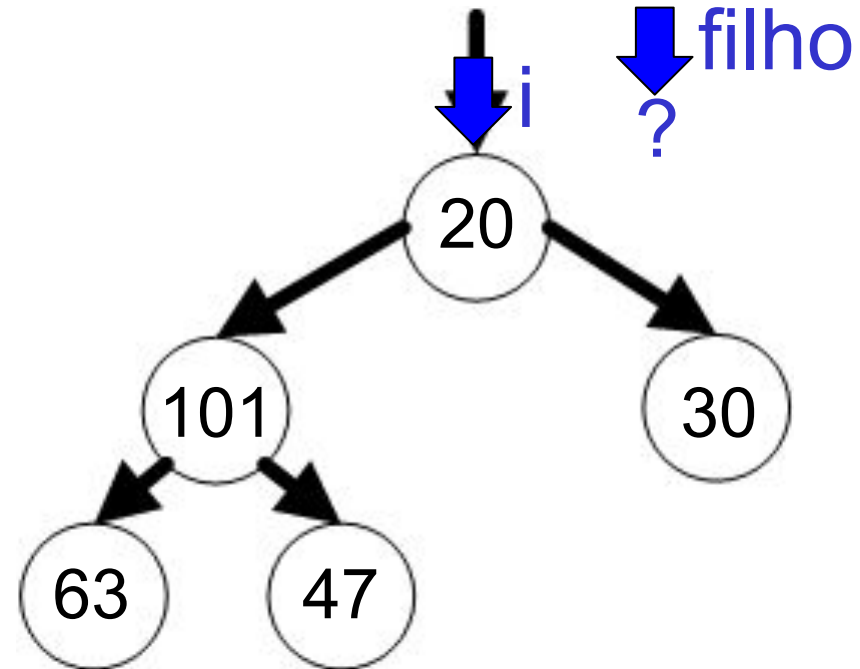
...
int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}
}
    
```

tam 5

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}
    
```

true: 1 <= 2



Algoritmo em C like

```

int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}

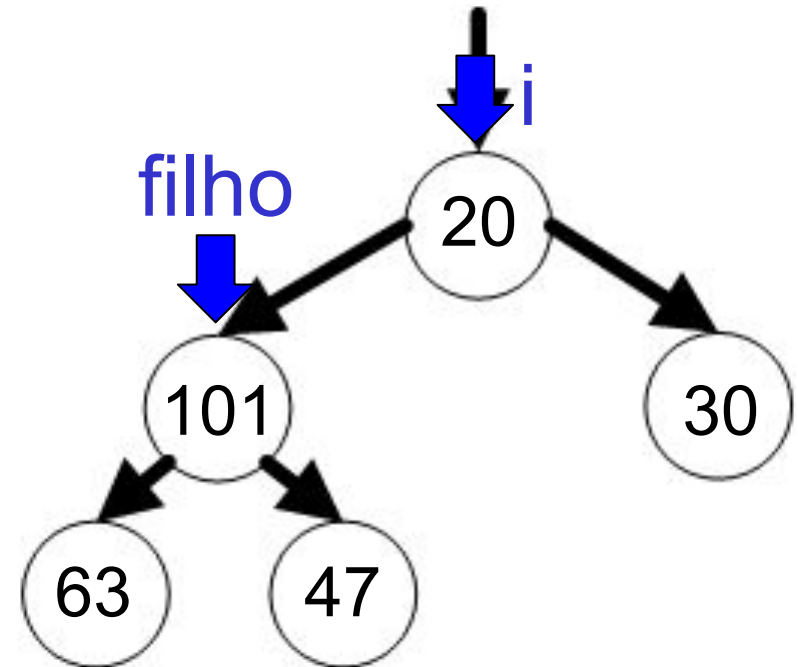
```

tam **5**

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



i **filho**

20	101	30	63	47	115
1	2	3	4	5	6

Algoritmo em C like

```

int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}

```

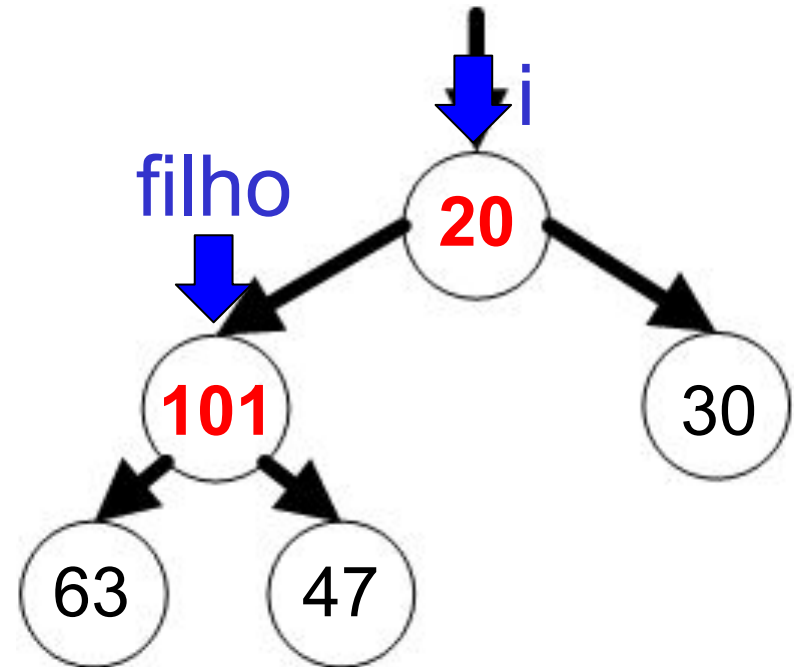
tam **5**

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```

true: 20 < 101



i **filho**

20	101	30	63	47	115
1	2	3	4	5	6

Algoritmo em C like

```

int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}

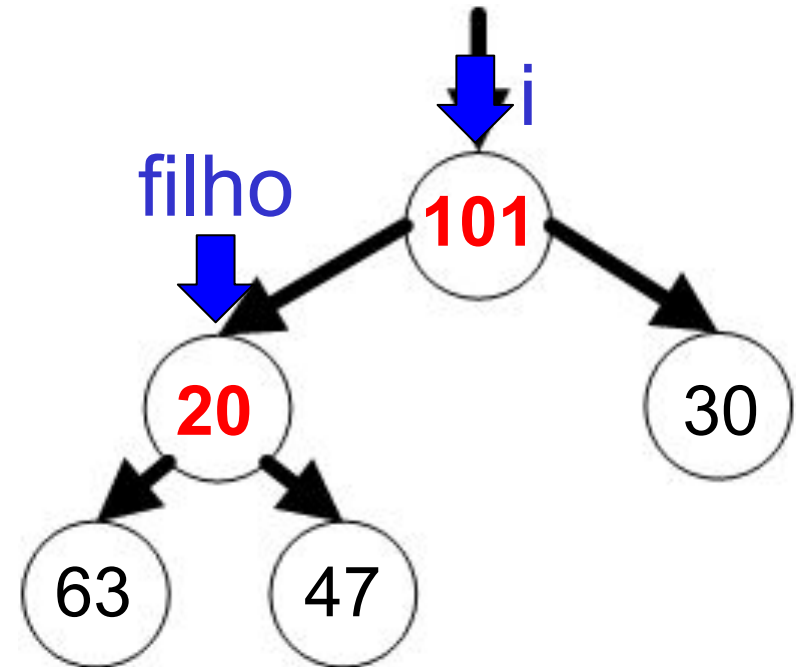
```

tam **5**

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



i **filho**

101	20	30	63	47	115
1	2	3	4	5	6

Algoritmo em C like

```

int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}

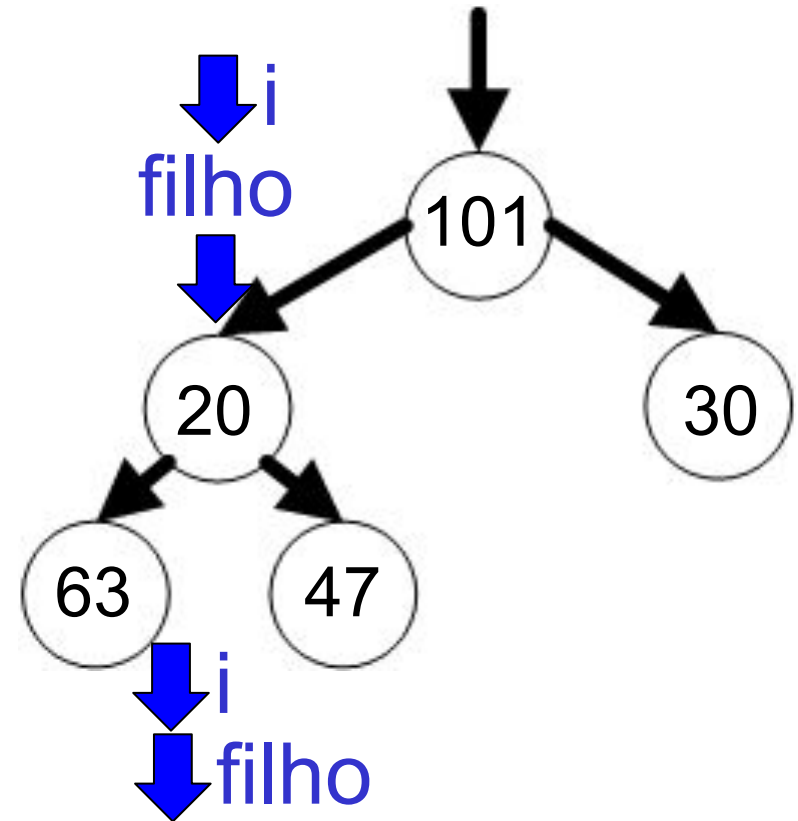
```

tam **5**

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



101	20	30	63	47	115
1	2	3	4	5	6

Algoritmo em C like

```

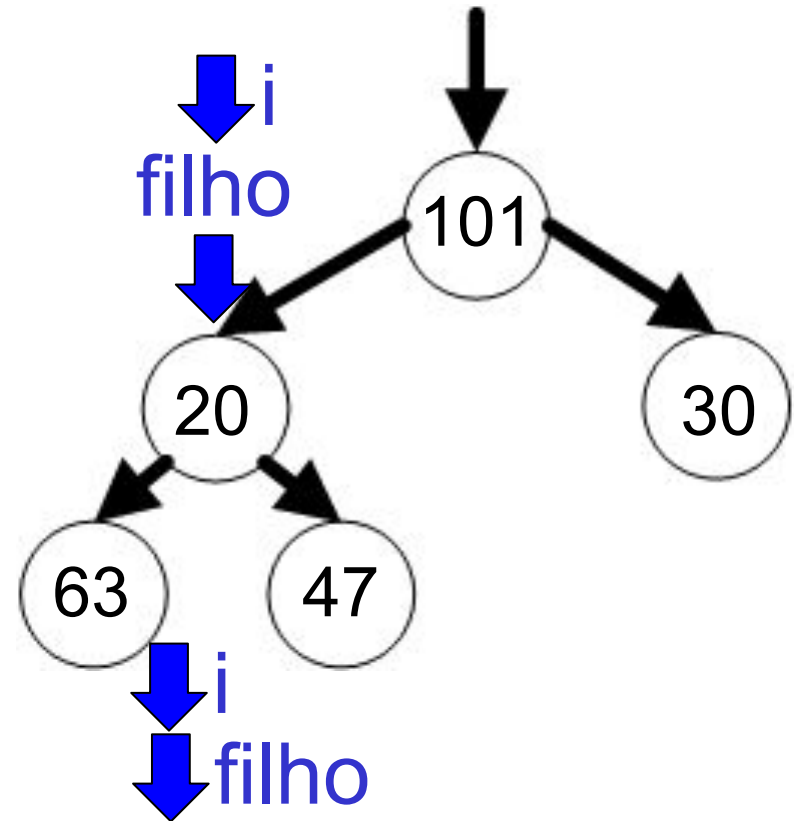
    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```

tam 5

```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```

true: $2 \leq 2$



101	20	30	63	47	115
1	2	3	4	5	6

Algoritmo em C like

```

int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}

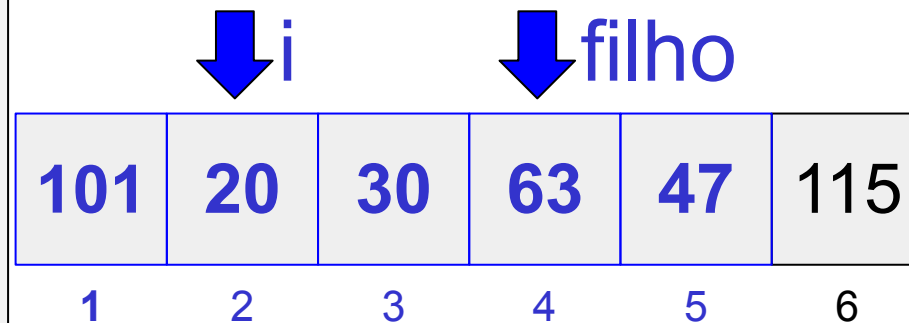
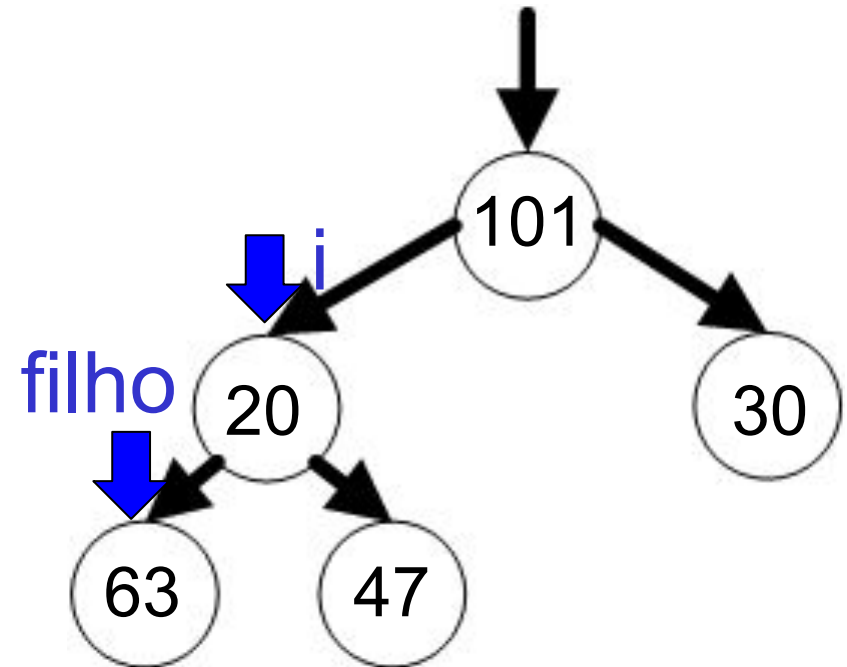
```

tam **5**

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



Algoritmo em C like

```

int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}

```

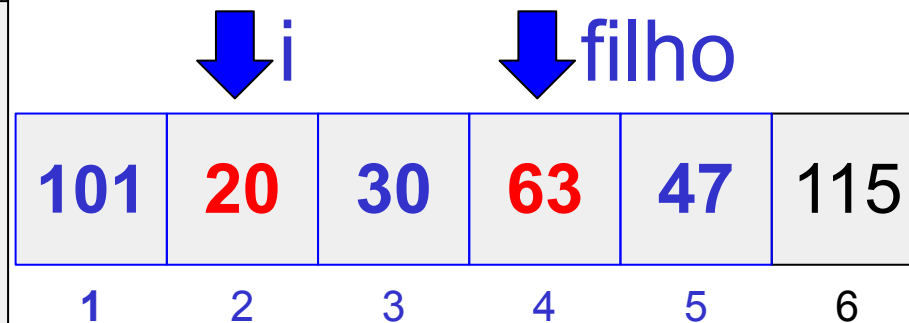
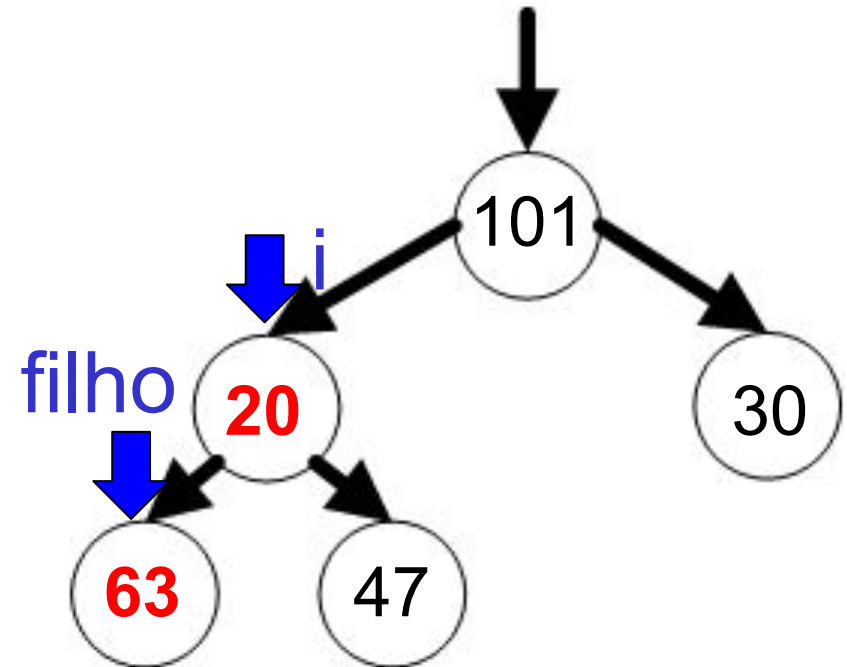
tam **5**

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```

true: 20 < 63



Algoritmo em C like

```

int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}

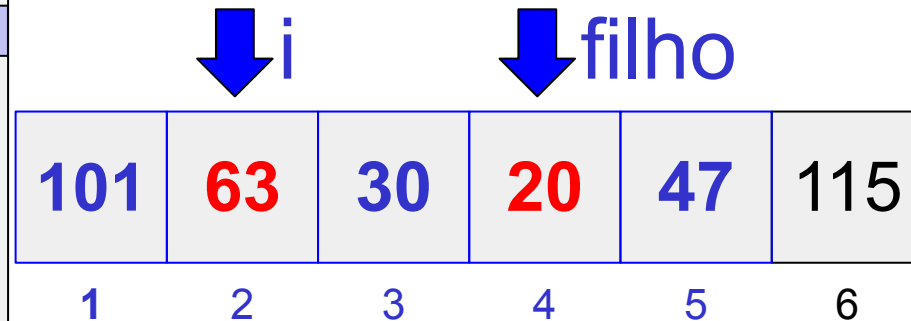
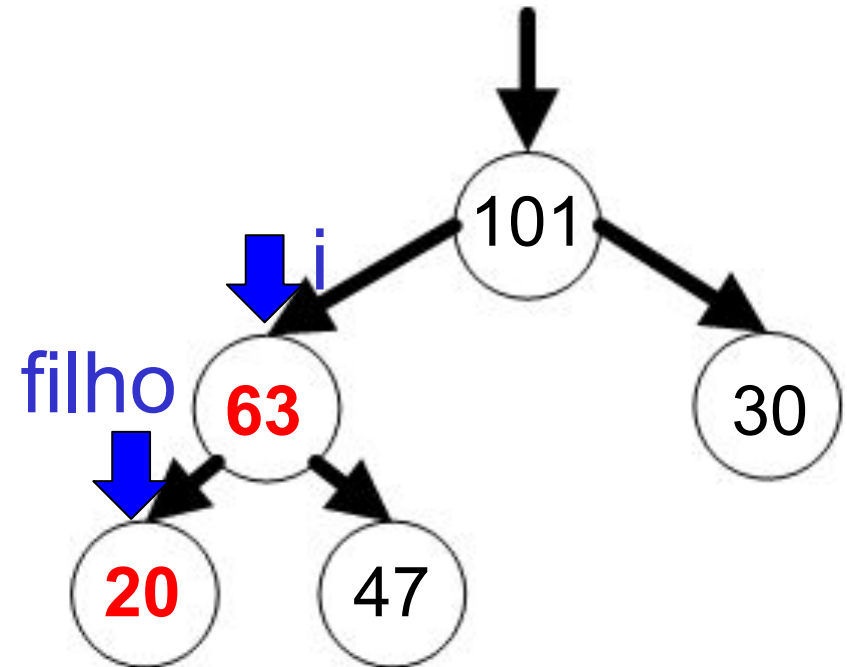
```

tam **5**

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



Algoritmo em C like

```

int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}

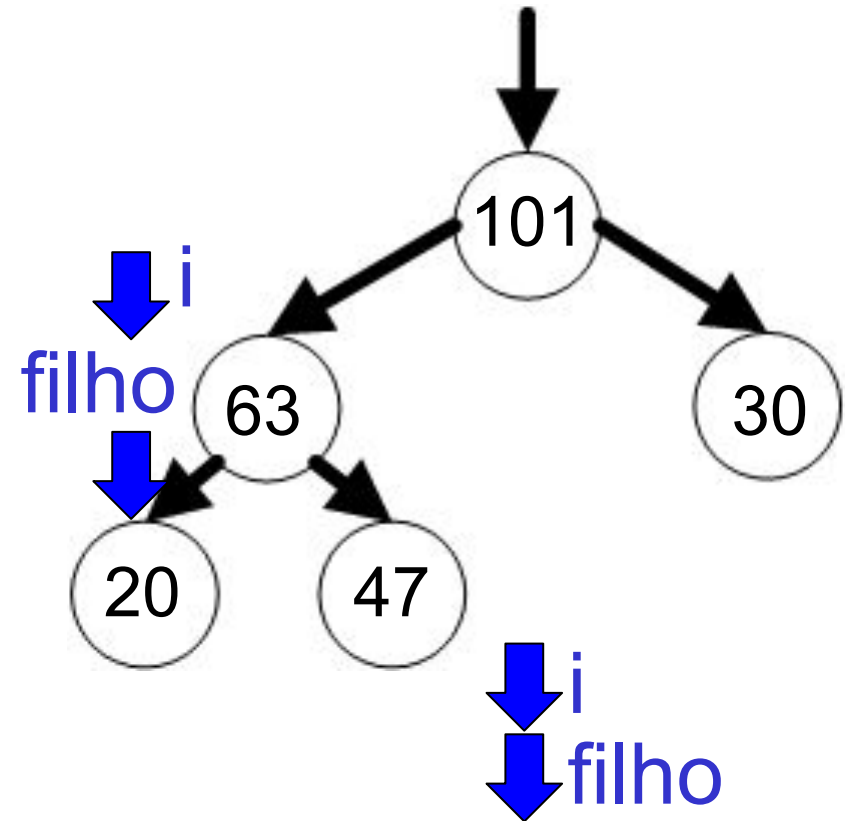
```

tam **5**

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



101	63	30	20	47	115
1	2	3	4	5	6

Algoritmo em C like

```

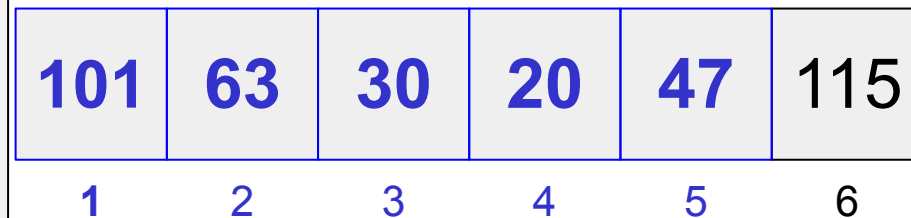
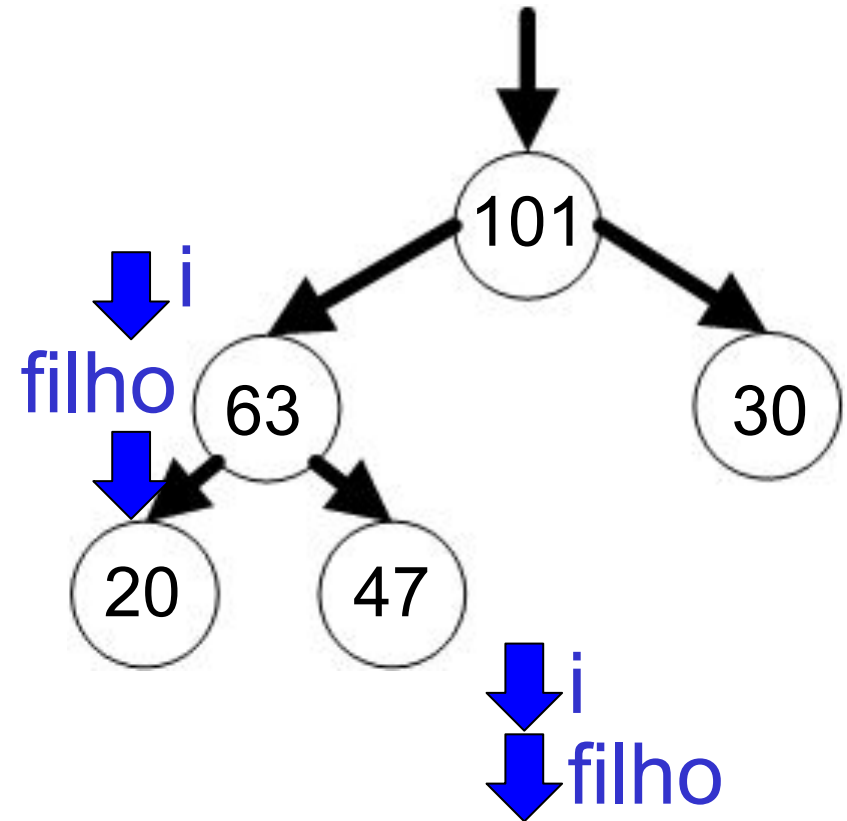
...
int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}
}
    
```

tam 5

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}
    
```

false: 4 <= 2



Algoritmo em C like

```

int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}

```

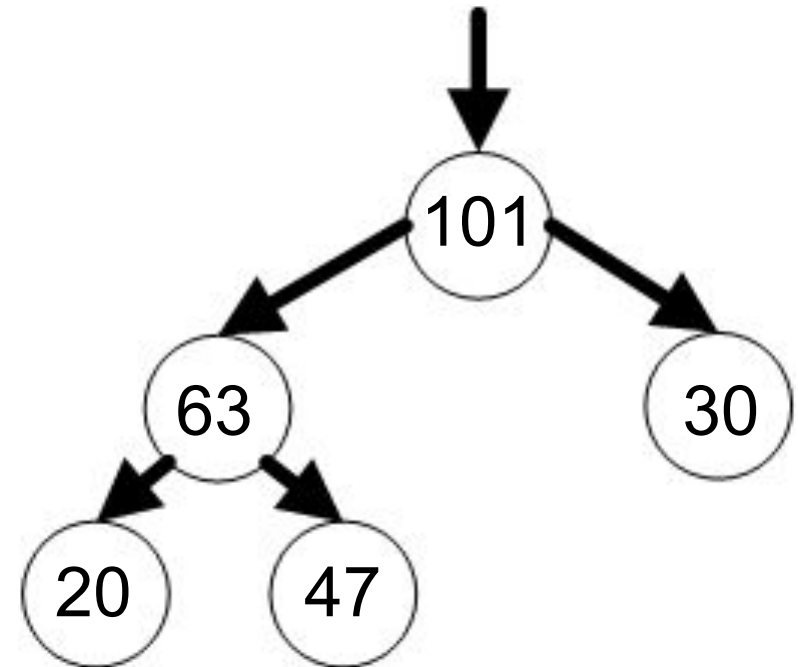
tam

5

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



101	63	30	20	47	115
1	2	3	4	5	6

Algoritmo em C like

```

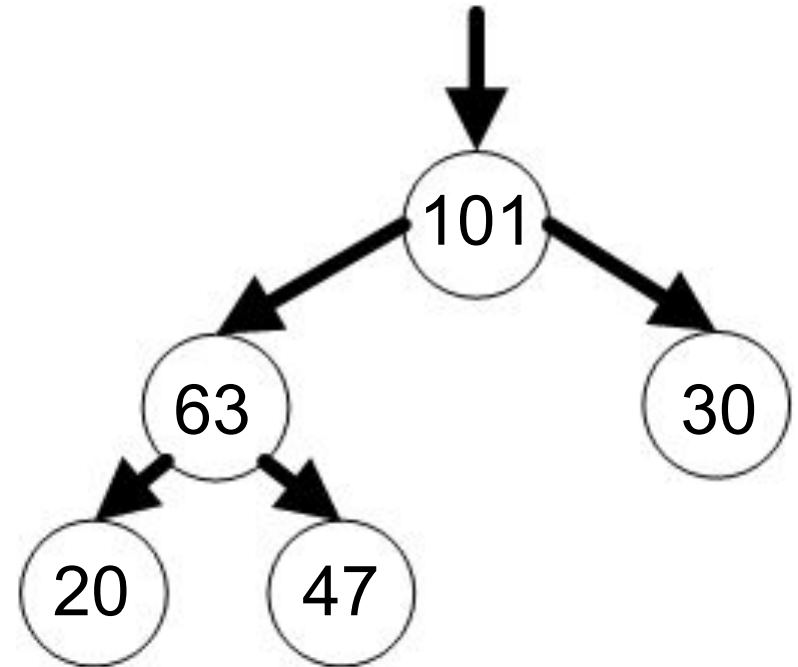
    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```

tam **5**

true: $5 > 1$

```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```



101	63	30	20	47	115
1	2	3	4	5	6

Algoritmo em C like

```

    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
}

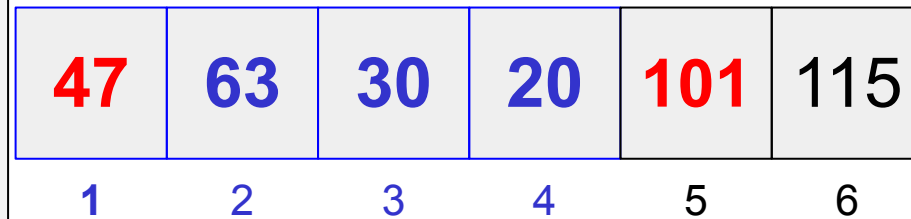
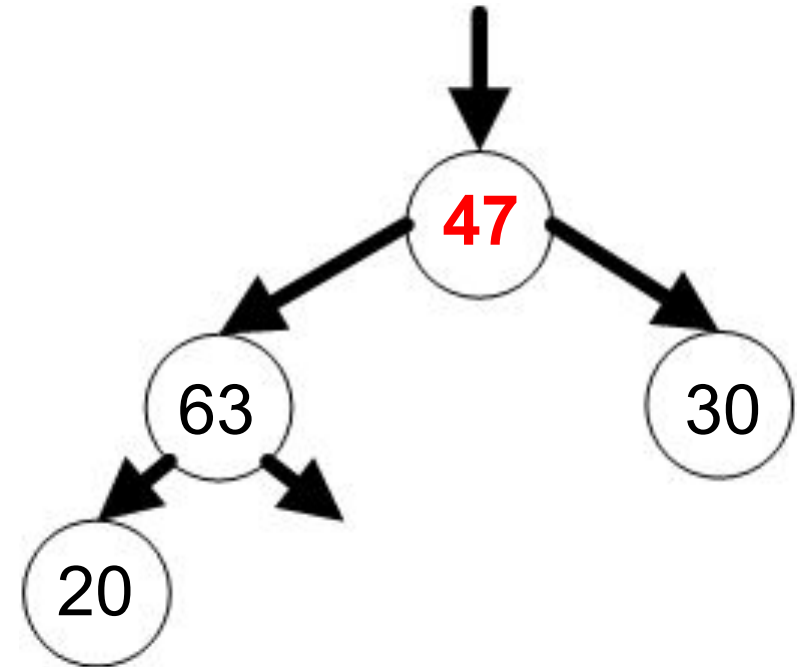
```



```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



Algoritmo em C like

```

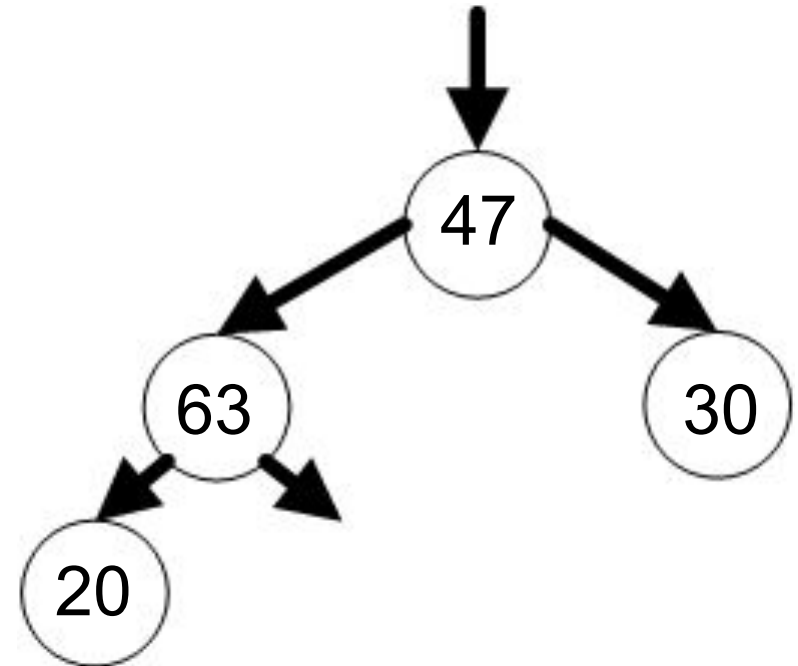
int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}
}
    
```

tam

4

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}
    
```



47	63	30	20	101	115
1	2	3	4	5	6

Algoritmo em C like

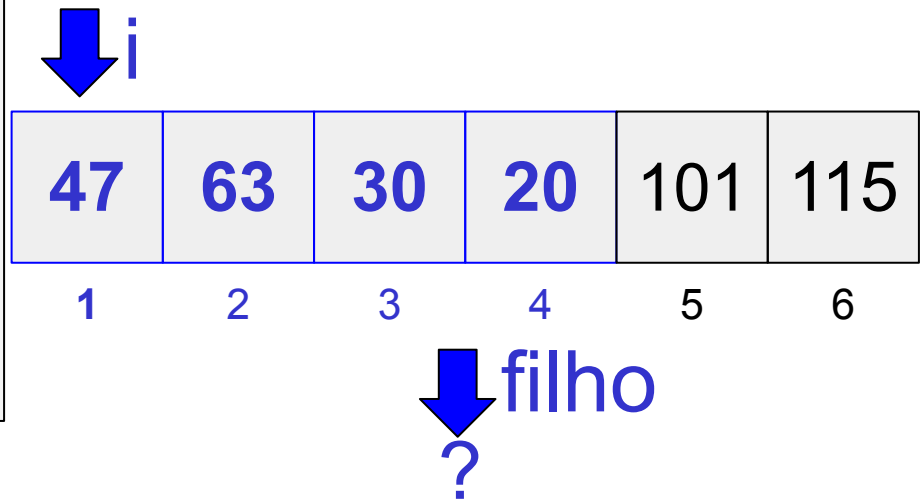
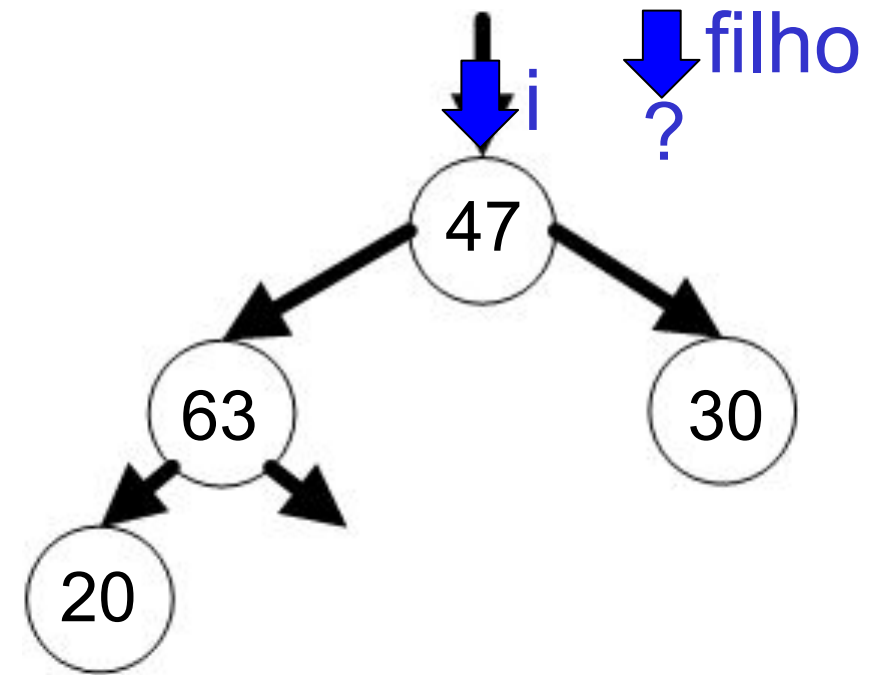
```

...
int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}
}
    
```

tam 4

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}
    
```



Algoritmo em C like

```

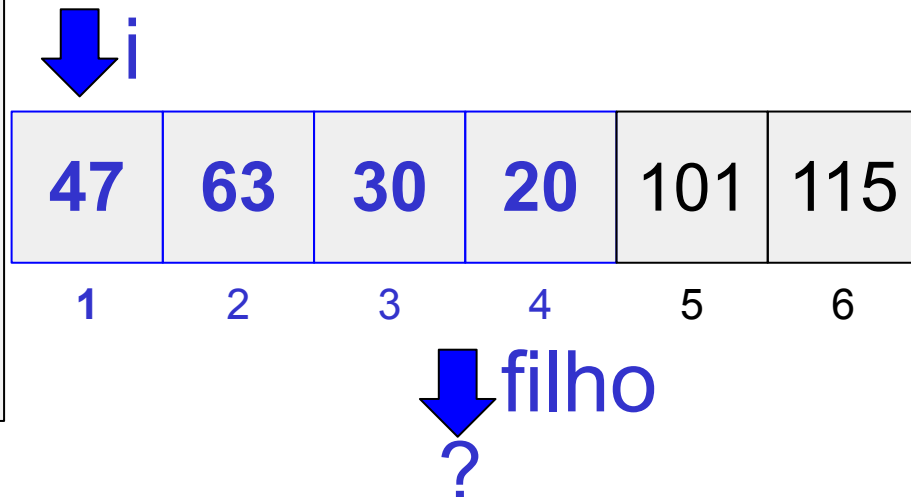
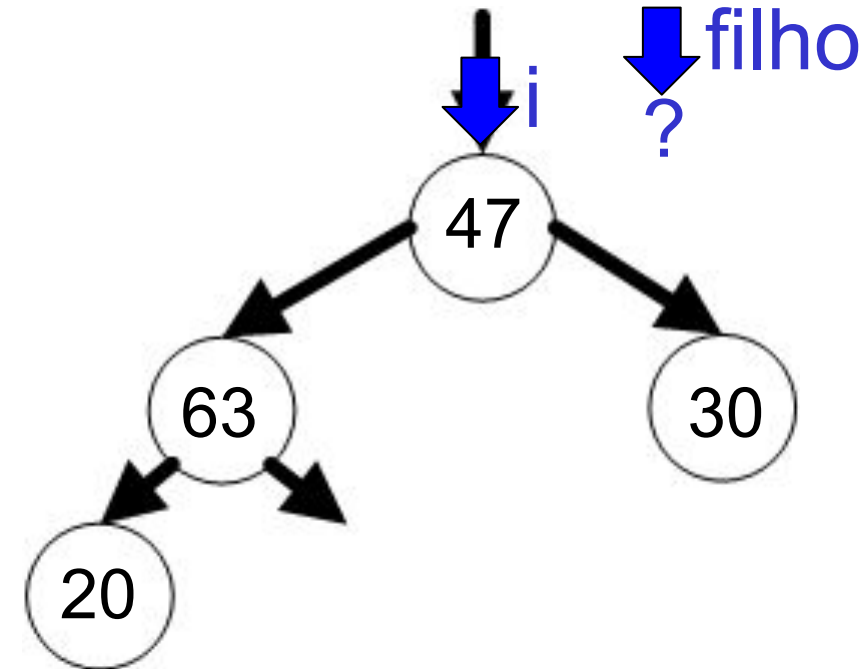
...
int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}
}
    
```

tam 4

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}
    
```

true: 1 <= 2



Algoritmo em C like

```

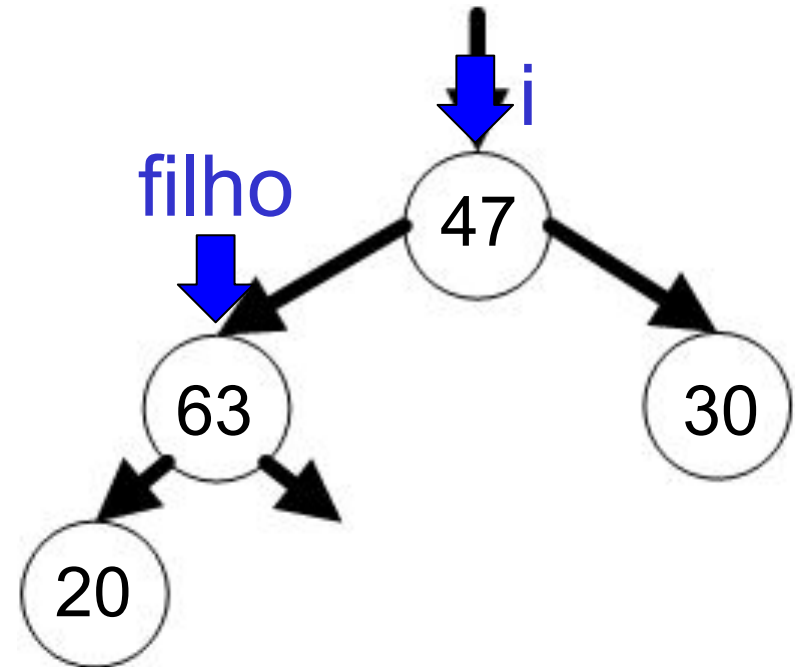
int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}
    
```

tam

4

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}
    
```



↓ i ↓ filho

47	63	30	20	101	115
1	2	3	4	5	6

Algoritmo em C like

```

int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}

```

tam

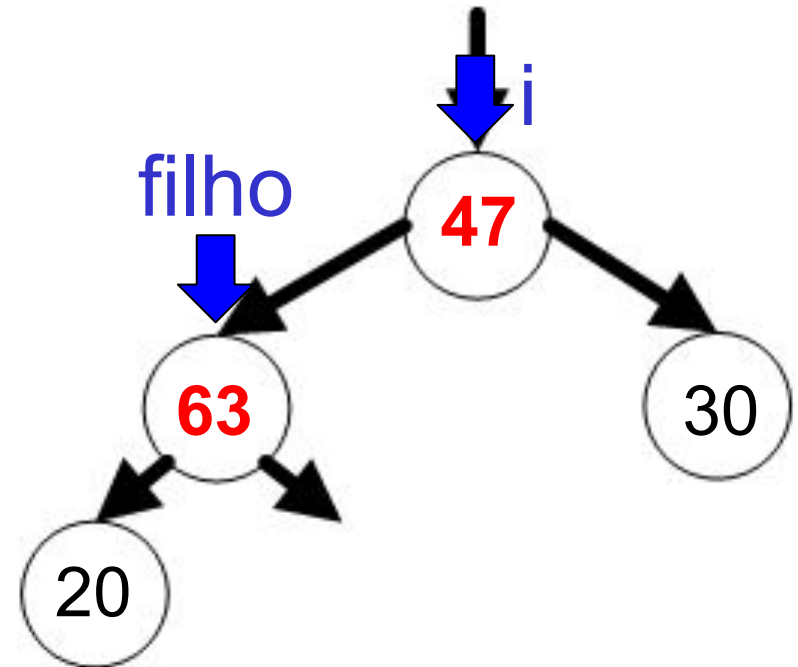
4

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```

true: 47 < 63



↓ i ↓ filho

47	63	30	20	101	115
1	2	3	4	5	6

Algoritmo em C like

```

int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}

```

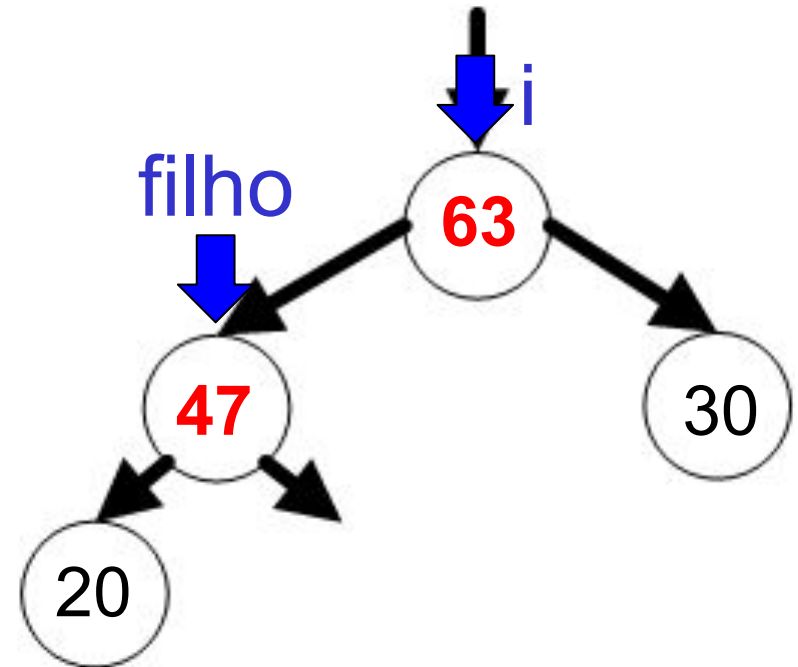
tam

4

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



↓ i ↓ filho

63	47	30	20	101	115
1	2	3	4	5	6

Algoritmo em C like

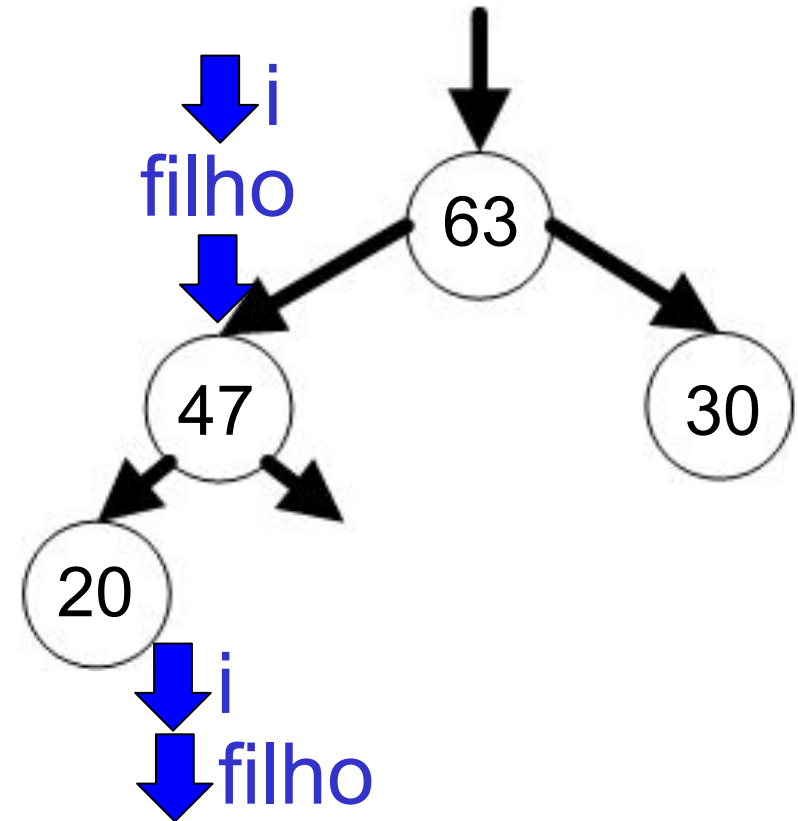
```

    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```

tam 4

```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```



63	47	30	20	101	115
1	2	3	4	5	6

Algoritmo em C like

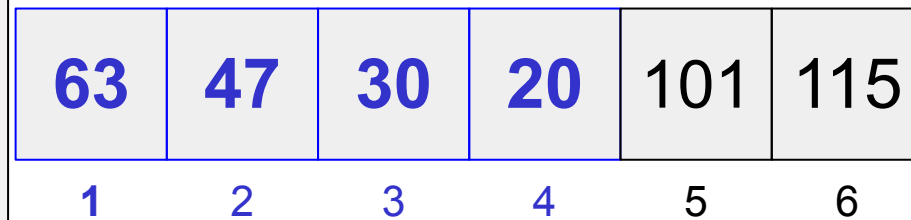
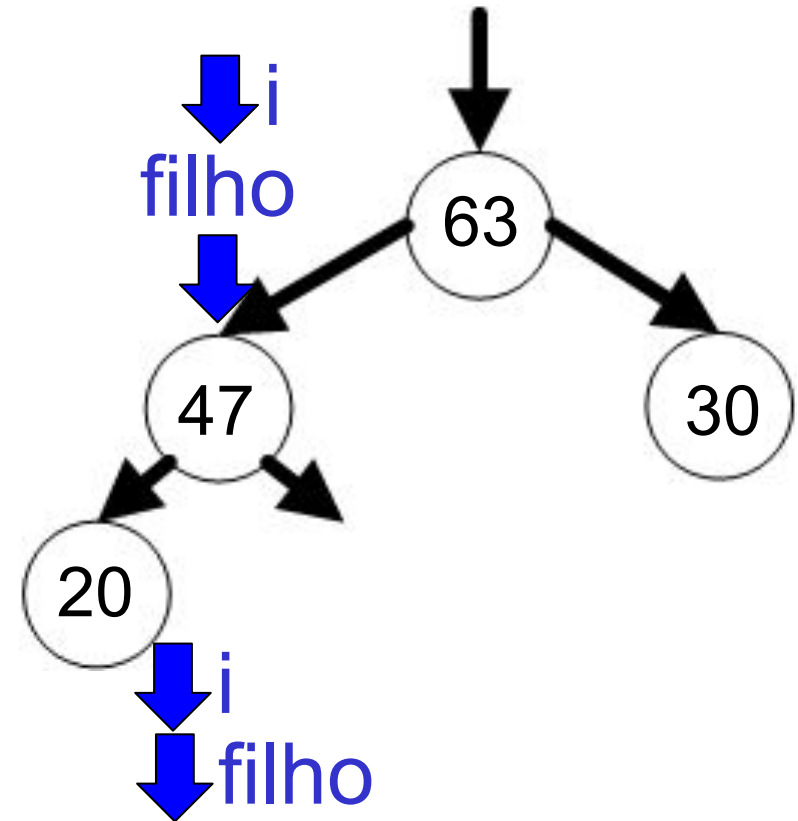
```

    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```



```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```



Algoritmo em C like

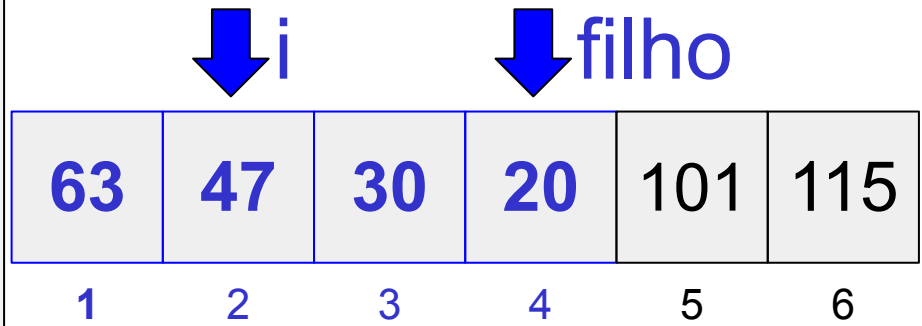
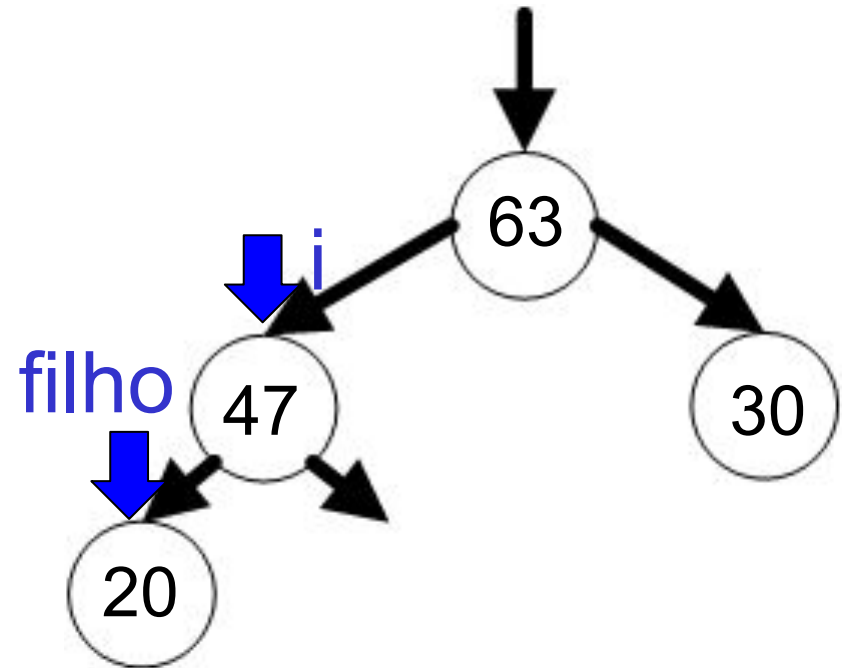
```

    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```



```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```



Algoritmo em C like

```

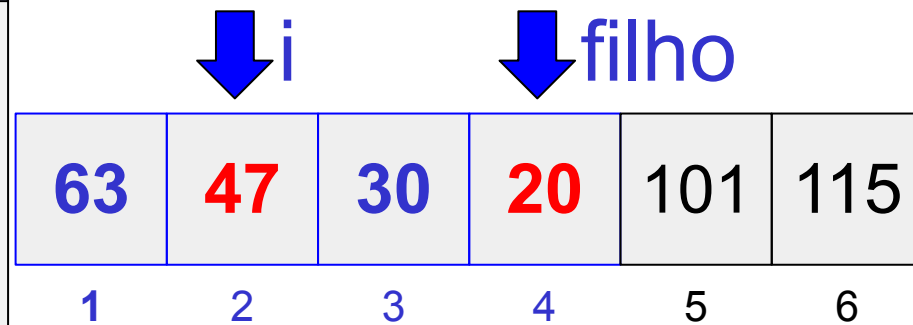
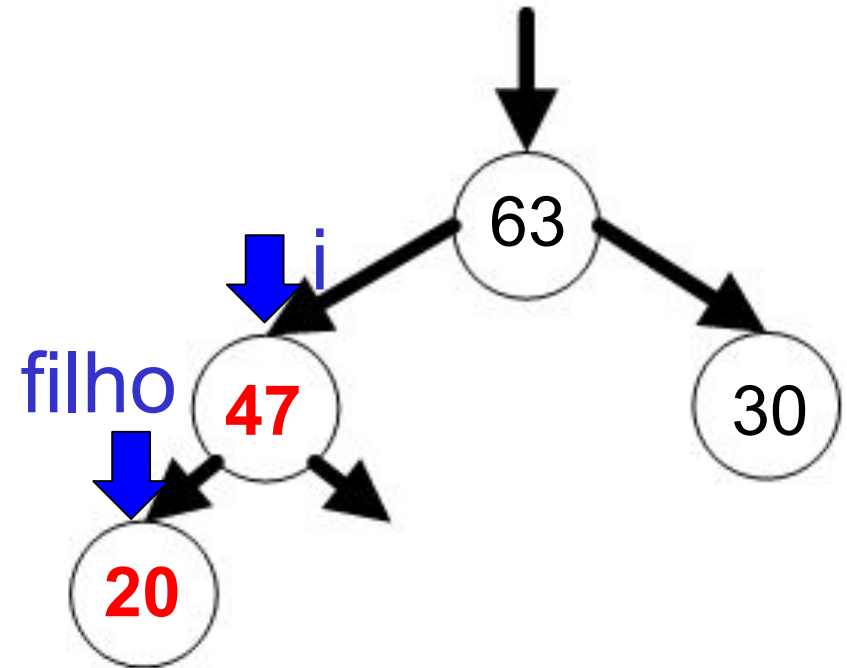
    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```



```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```

false: 47 < 20



Algoritmo em C like

```

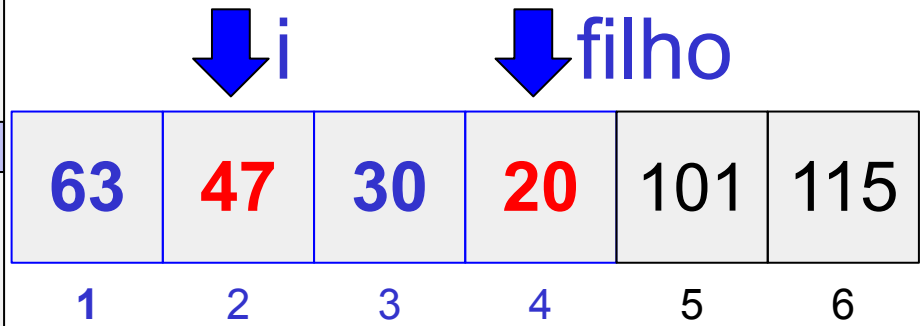
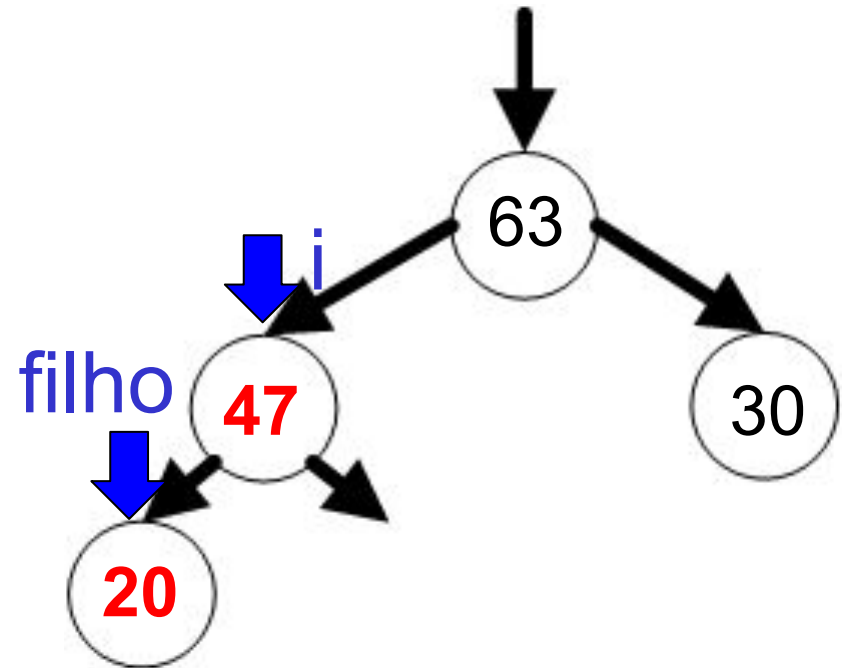
    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```



```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```

false: 47 < 20



Algoritmo em C like

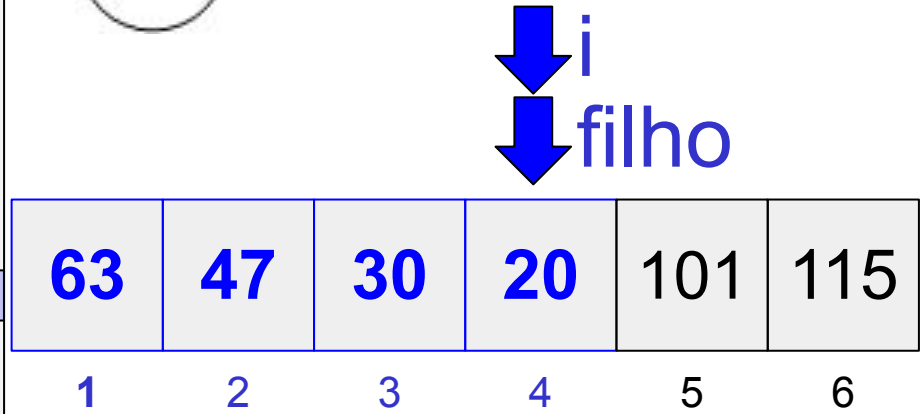
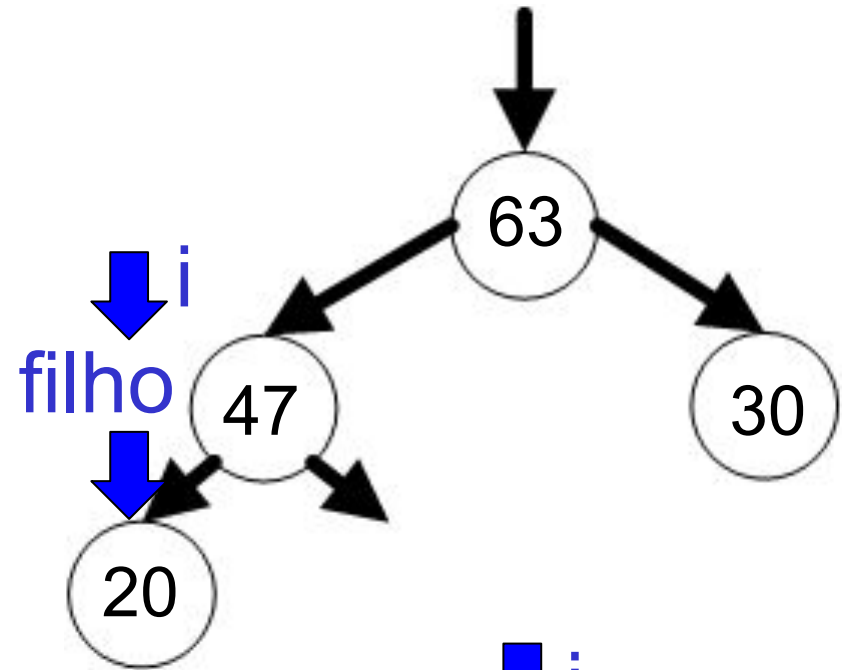
```

...
int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}
}
    
```



```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}
    
```



Algoritmo em C *like*

```

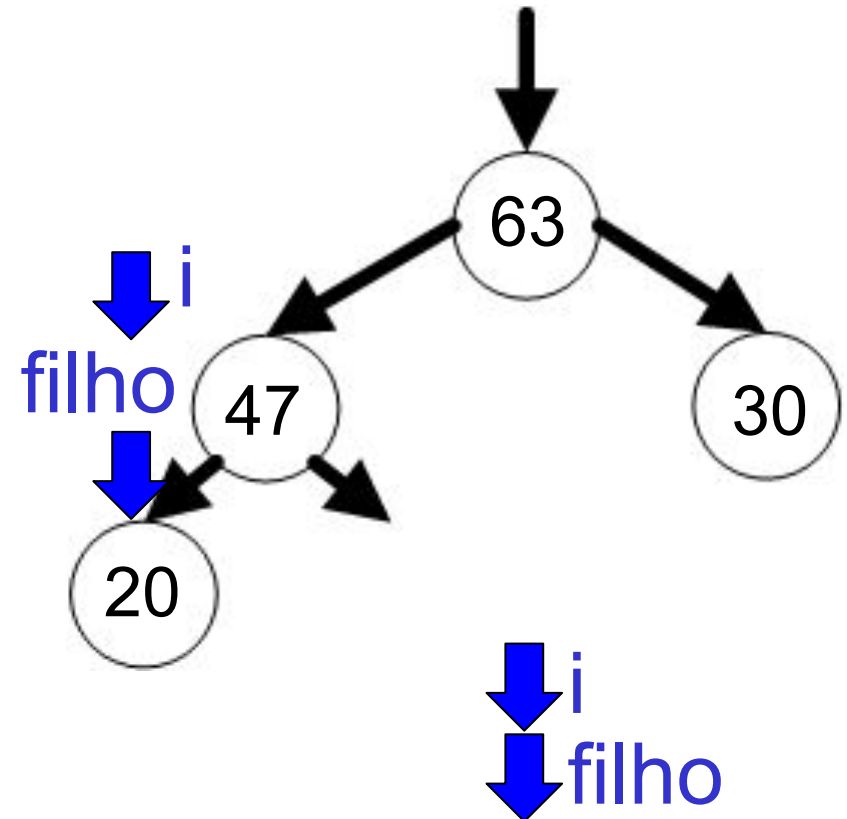
    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```

tam 4

```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```

false: 4 <= 2



63	47	30	20	101	115
1	2	3	4	5	6

Algoritmo em C *like*

```

    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
}

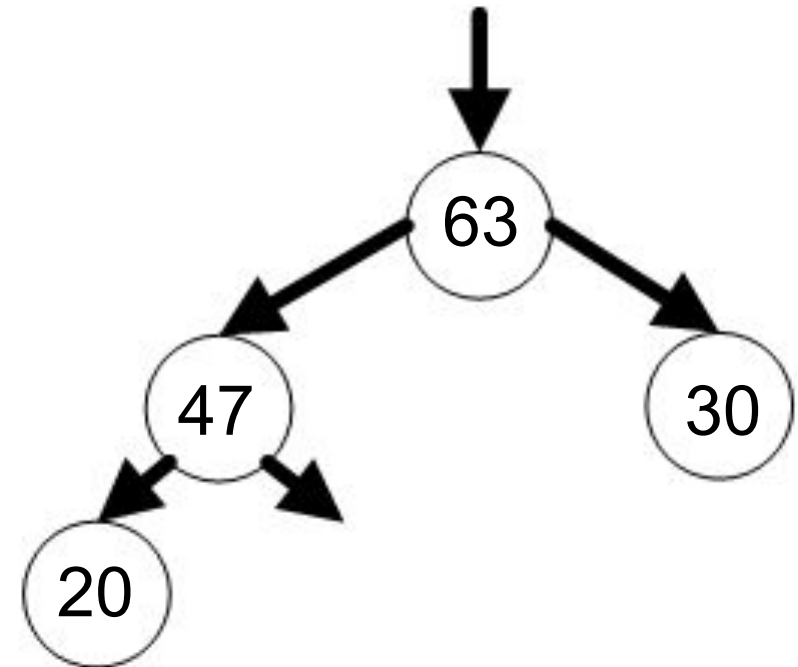
```



```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



63	47	30	20	101	115
1	2	3	4	5	6

Algoritmo em C like

```

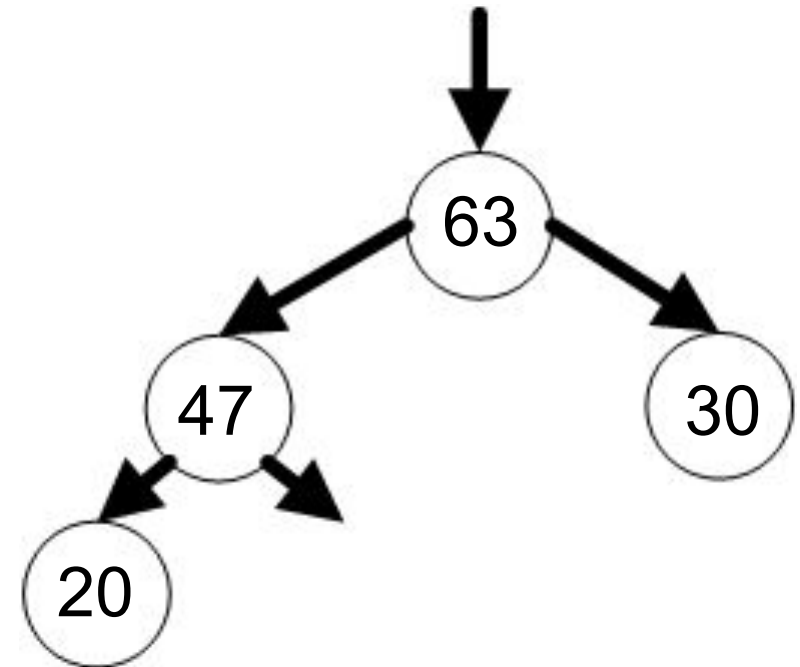
    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```

tam 4

true: 4 > 1

```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```



63	47	30	20	101	115
1	2	3	4	5	6

Algoritmo em C like

```

int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}

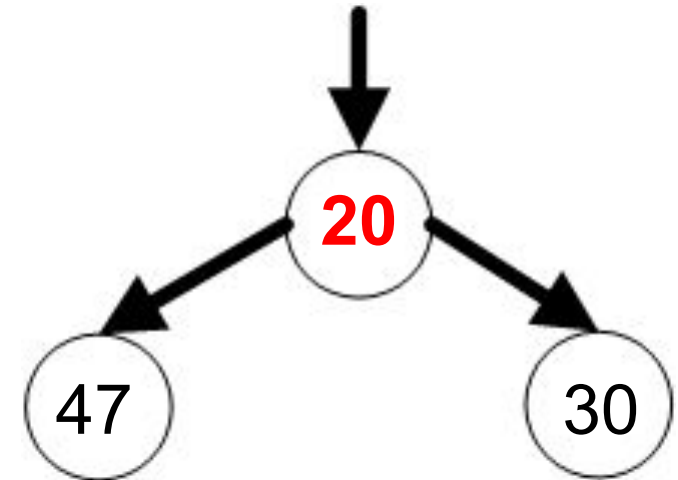
```

tam **3**

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



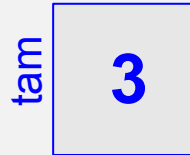
20	47	30	63	101	115
1	2	3	4	5	6

Algoritmo em C like

```

    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
}

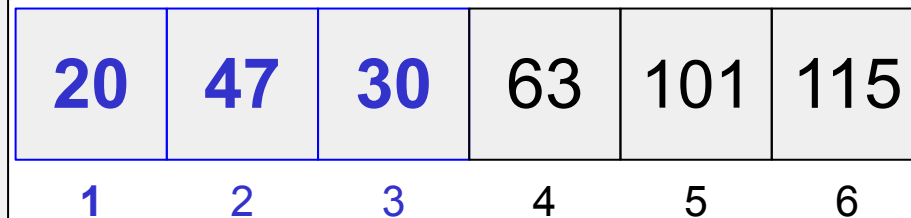
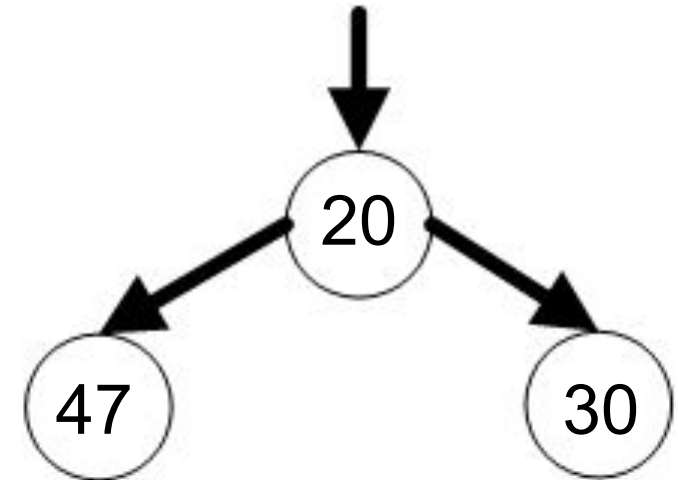
```



```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



Algoritmo em C like

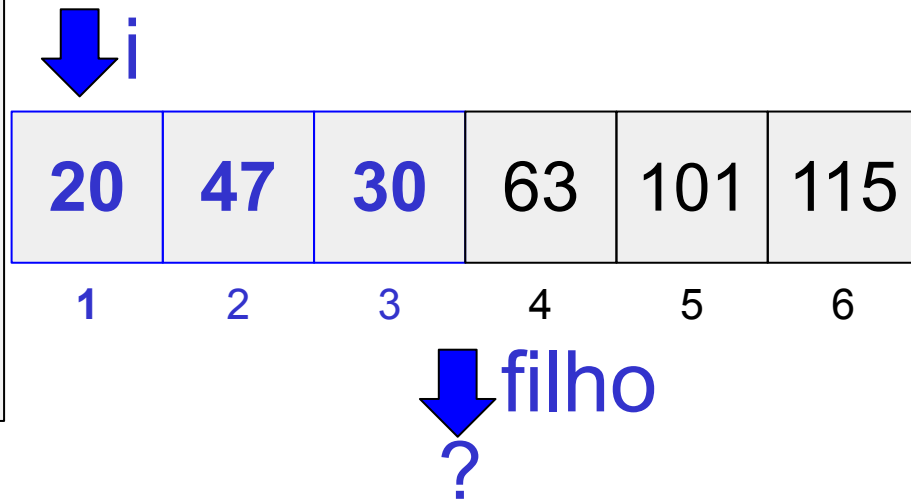
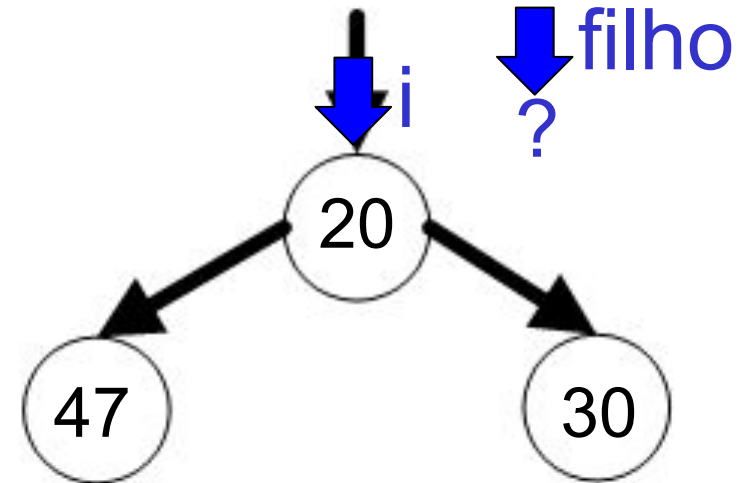
```

    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```

tam **3**

```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```



Algoritmo em C like

```

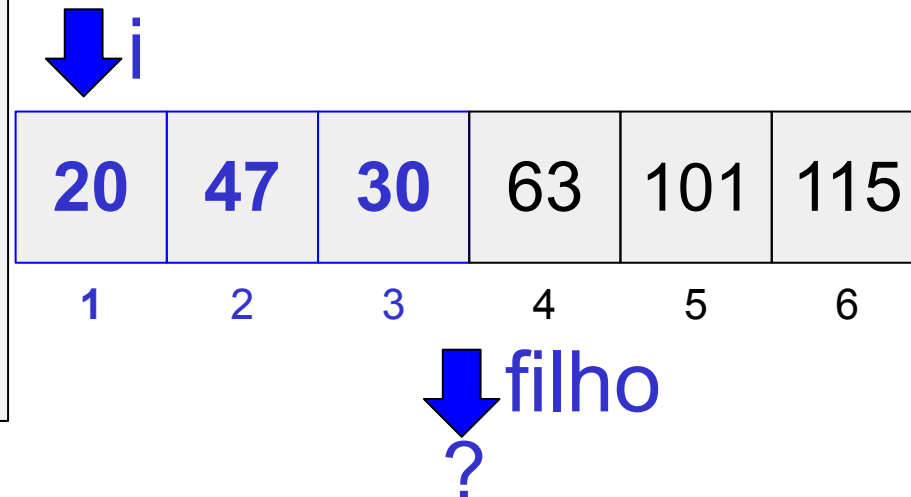
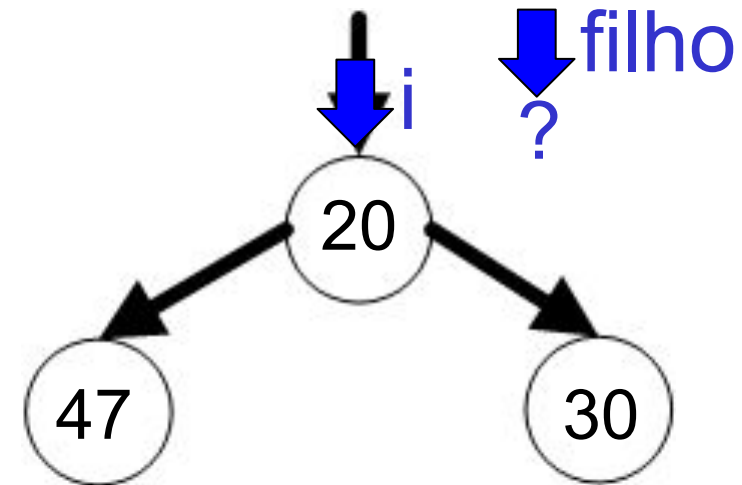
...
int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}
    
```

tam 3

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}
    
```

true: 1 <= 1



Algoritmo em C like

```

int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}

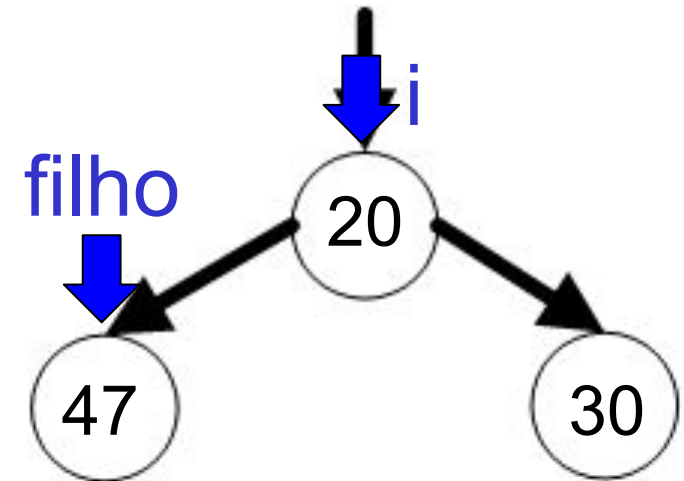
```

tam **3**

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



i **filho**

20	47	30	63	101	115
1	2	3	4	5	6

Algoritmo em C like

```

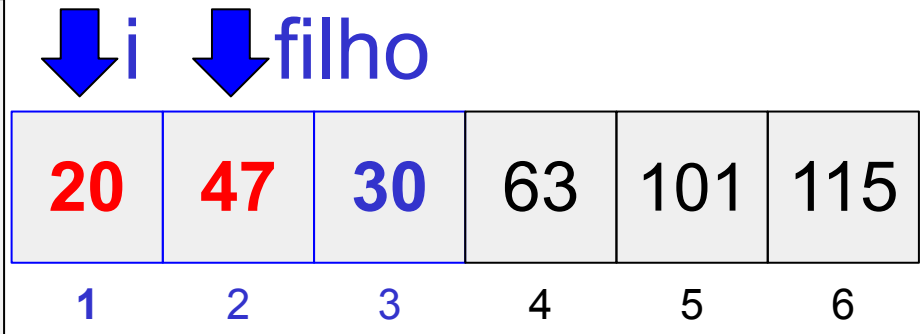
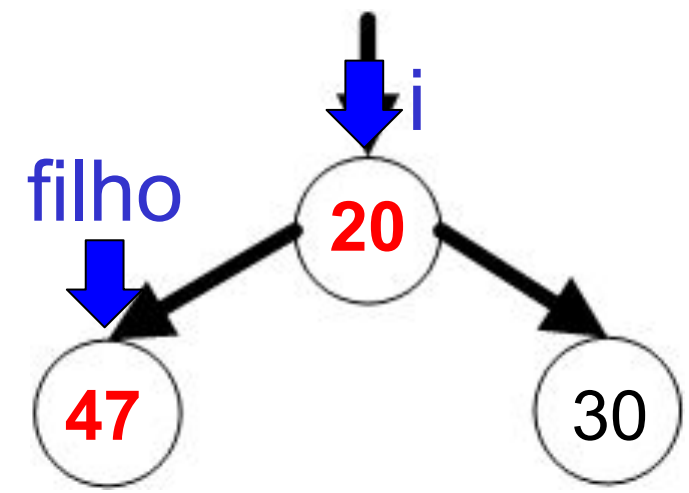
    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```



```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```

true: 20 < 47



Algoritmo em C like

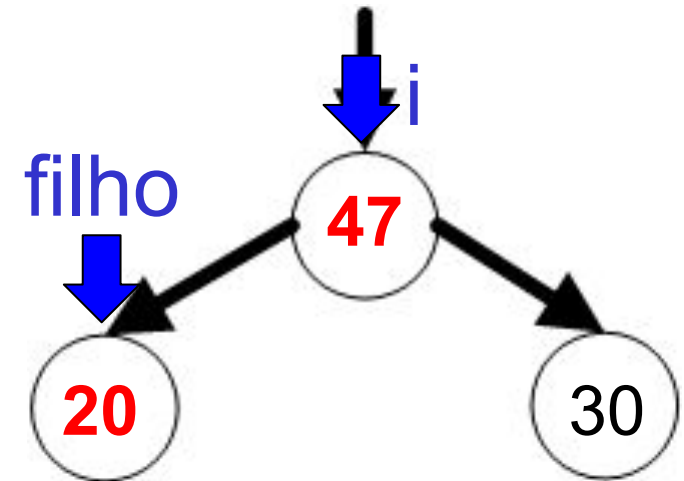
```

int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}
    
```

tam **3**

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}
    
```



i **filho**

47	20	30	63	101	115
1	2	3	4	5	6

Algoritmo em C like

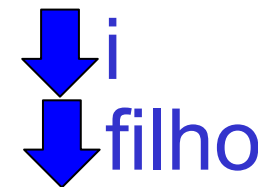
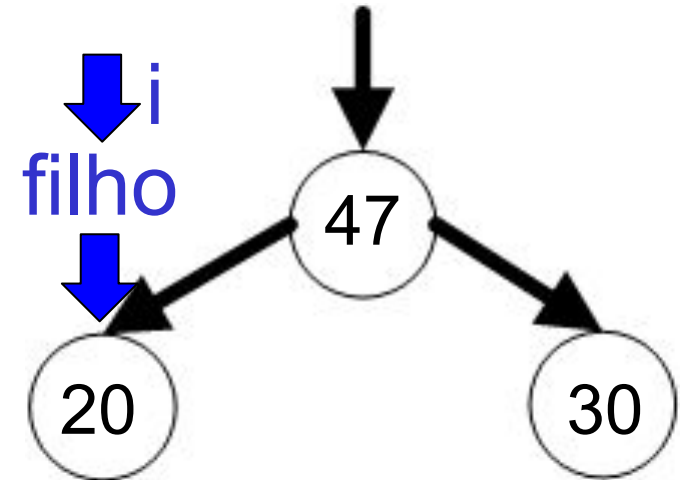
```

int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}
    
```

tam **3**

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}
    
```



47	20	30	63	101	115
1	2	3	4	5	6

Algoritmo em C like

```

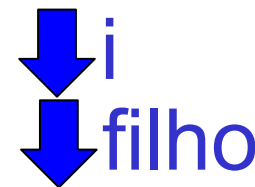
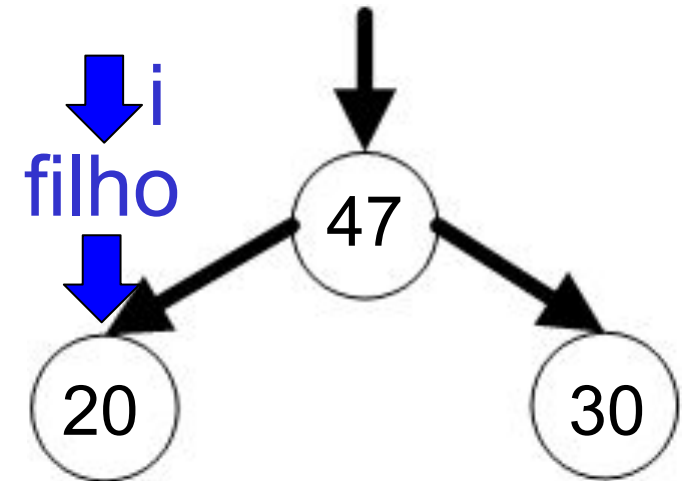
    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```

tam 3

```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```

false: $2 \leq 1$



47	20	30	63	101	115
1	2	3	4	5	6

Algoritmo em C *like*

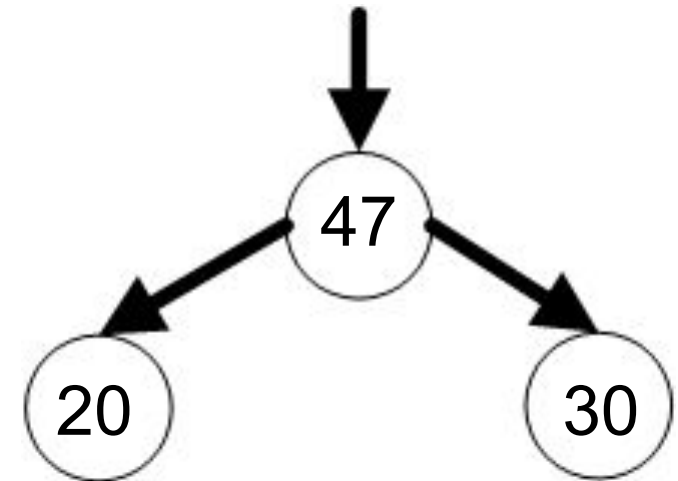
```

    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```

tam **3**

```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```



47	20	30	63	101	115
1	2	3	4	5	6

Algoritmo em C like

```

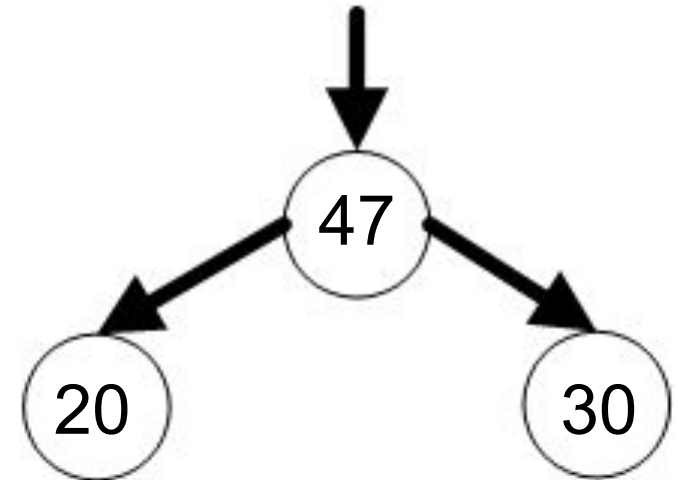
    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```

tam **3**

true: 3 > 1

```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```

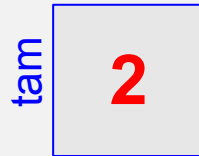


47	20	30	63	101	115
1	2	3	4	5	6

Algoritmo em C *like*

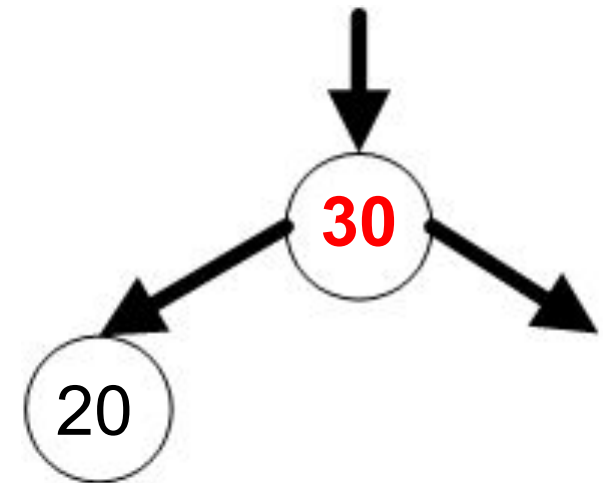
```

    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
}
    
```



```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}
    
```



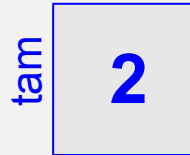
30	20	47	63	101	115
1	2	3	4	5	6

Algoritmo em C *like*

```

    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
}

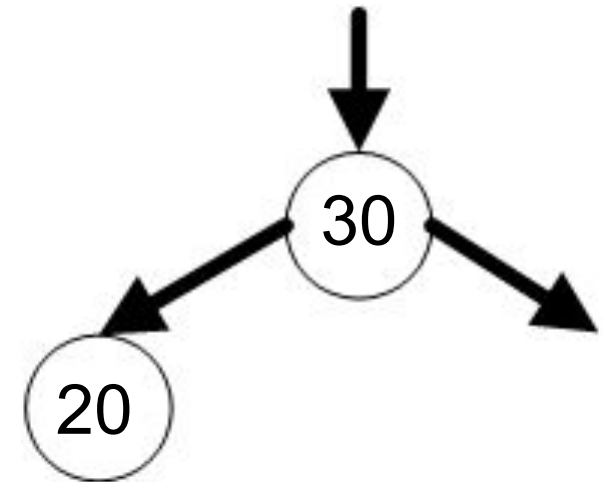
```



```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```

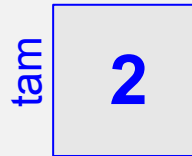


30	20	47	63	101	115
1	2	3	4	5	6

Algoritmo em C like

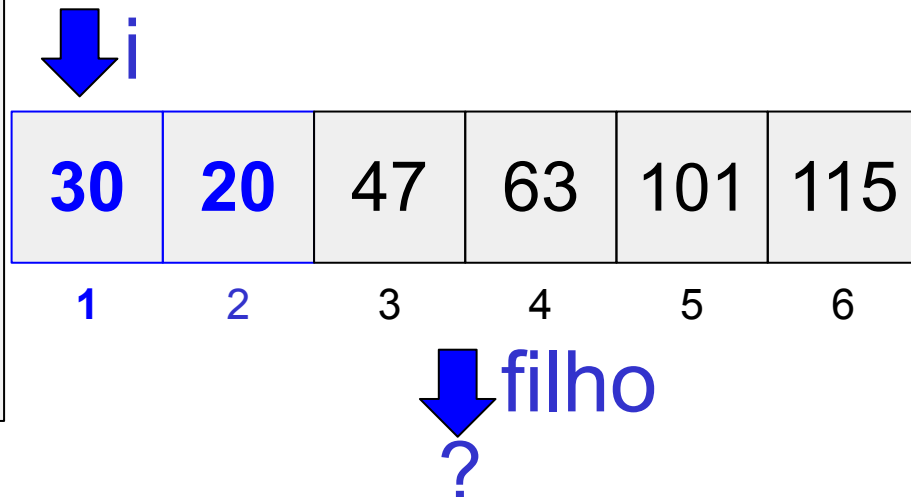
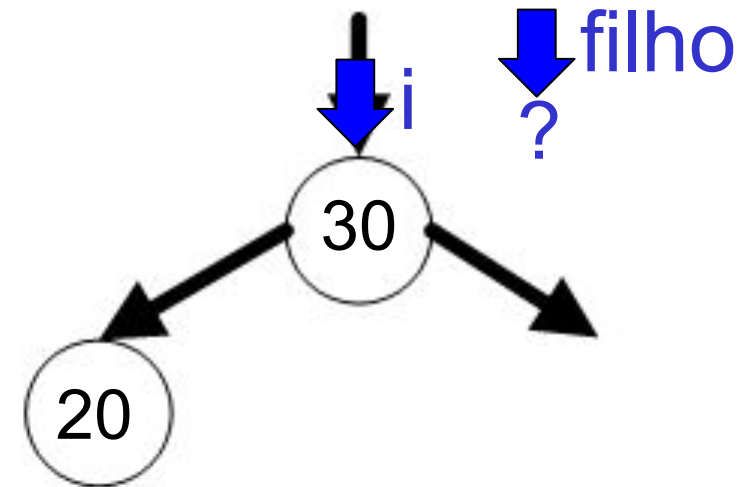
```

    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```



```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```



Algoritmo em C like

```

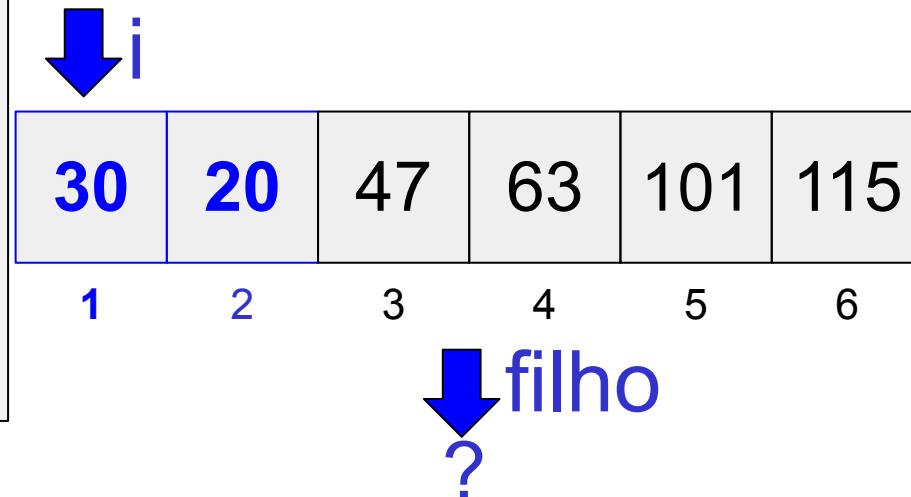
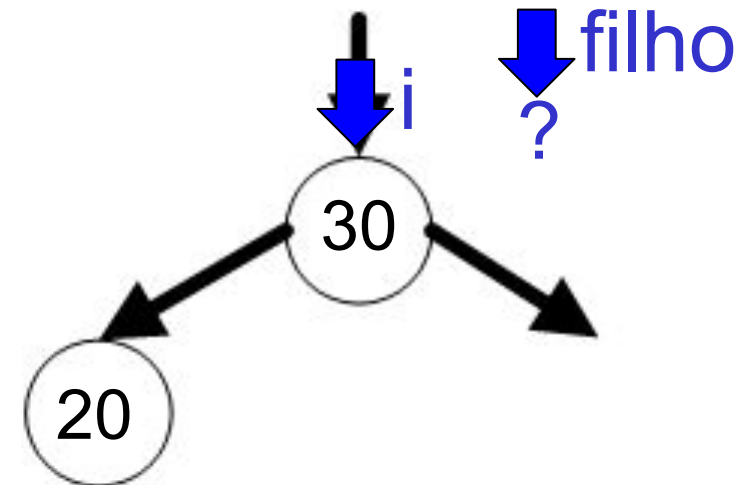
    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```

tam 2

```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```

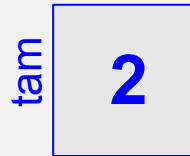
true: $1 \leq 1$



Algoritmo em C *like*

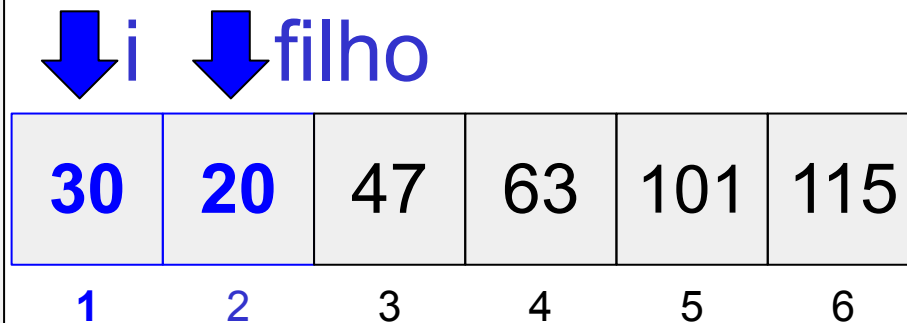
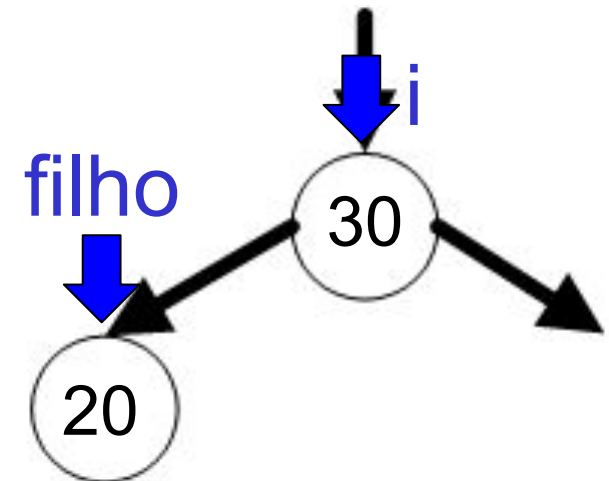
```

    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```



```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```



Algoritmo em C like

```

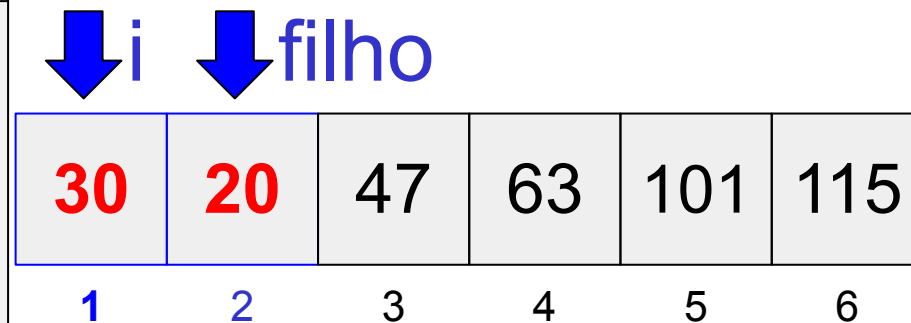
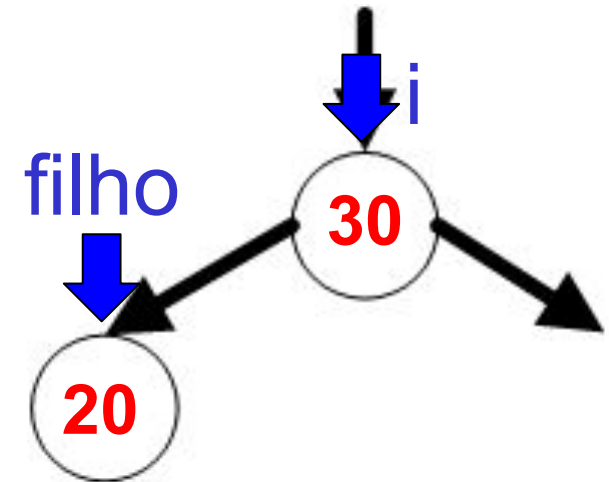
    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```

tam 2

```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```

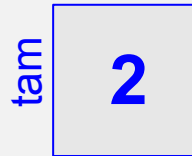
false: 30 < 20



Algoritmo em C *like*

```

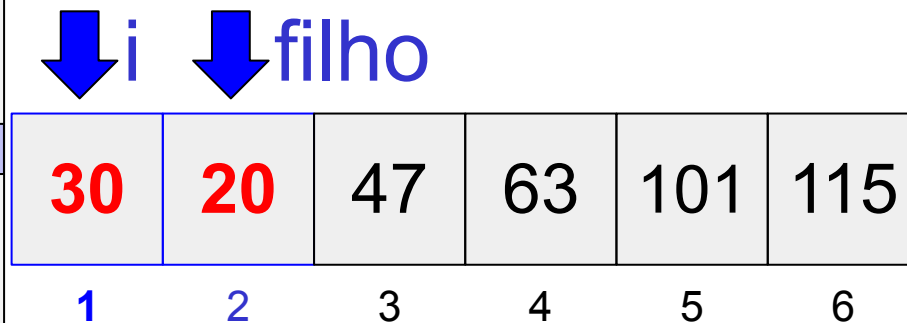
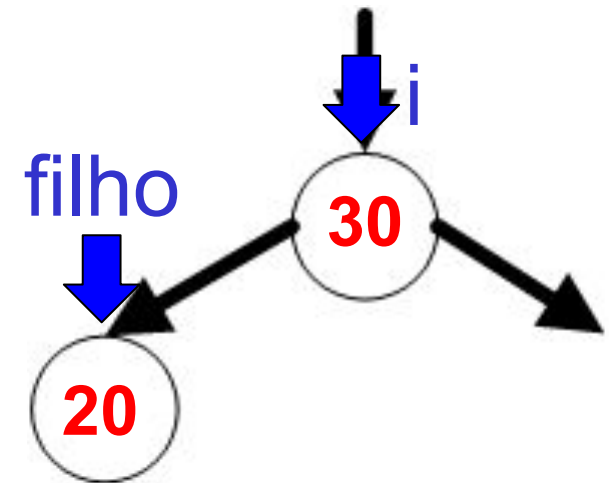
    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```



```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```

false: 30 < 20



Algoritmo em C *like*

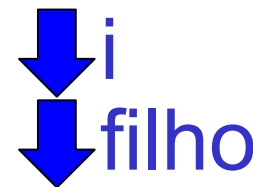
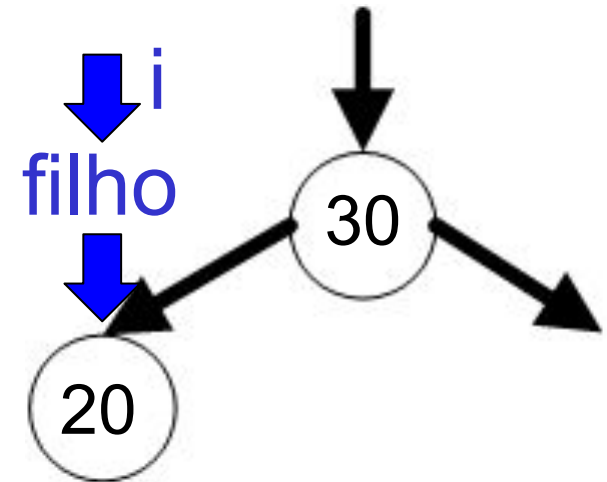
```

    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```

tam 2

```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```



30	20	47	63	101	115
1	2	3	4	5	6

Algoritmo em C *like*

```

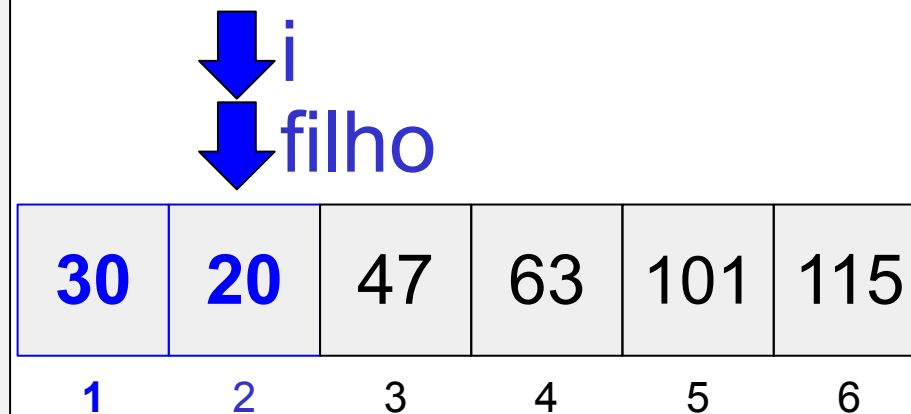
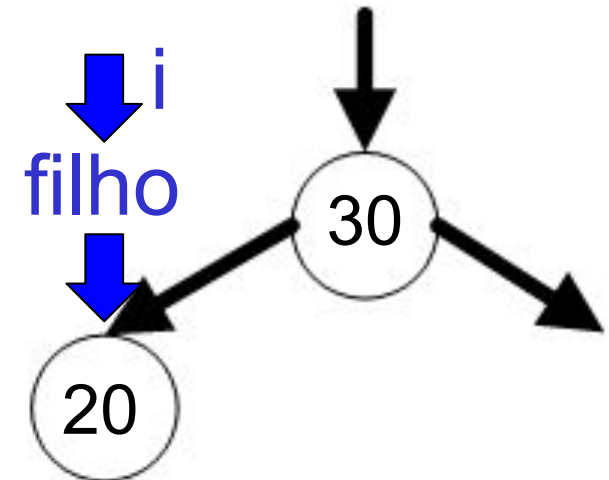
    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```

tam 2

```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```

false: $2 \leq 1$



Algoritmo em C *like*

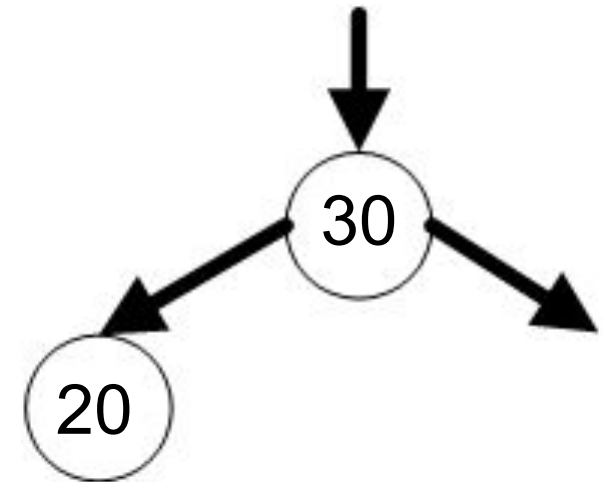
```

    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```

tam **2**

```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```



30	20	47	63	101	115
1	2	3	4	5	6

Algoritmo em C *like*

```

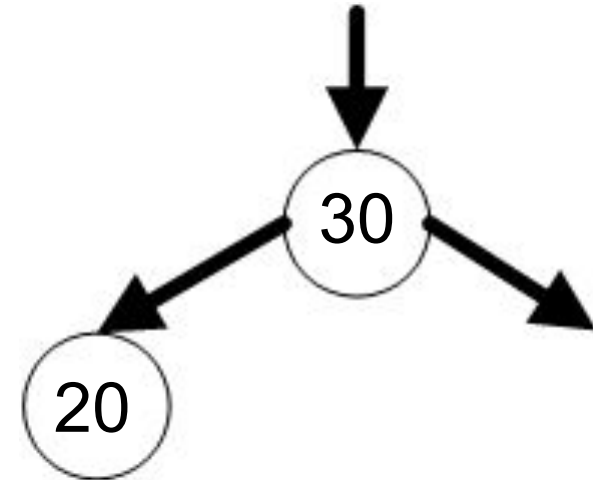
    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```

tam **2**

true: 2 > 1

```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```



30	20	47	63	101	115
1	2	3	4	5	6

Algoritmo em C *like*

```

int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}

```

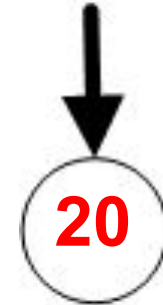
tam

1

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



20	30	47	63	101	115
1	2	3	4	5	6

Algoritmo em C *like*

```

int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}

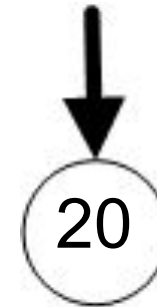
```

tam 1

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



20	30	47	63	101	115
1	2	3	4	5	6

Algoritmo em C *like*

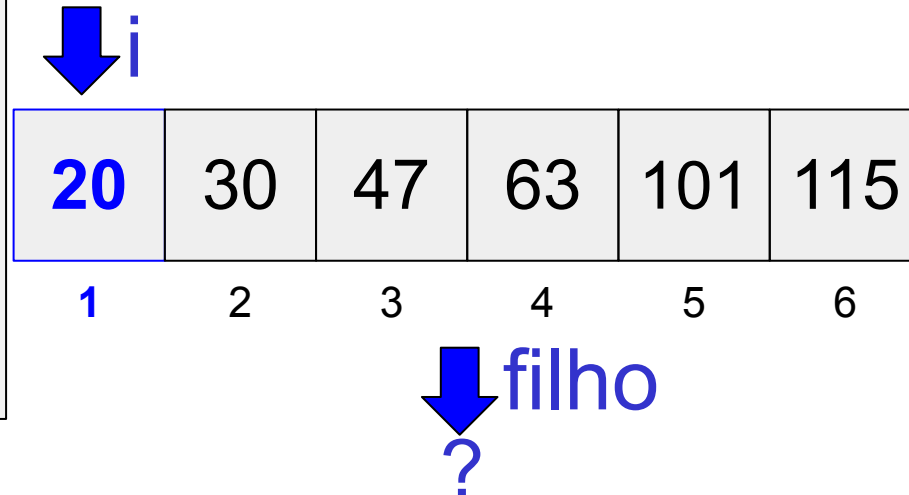
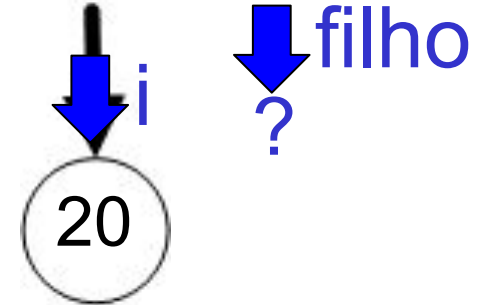
```

    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```

tam 1

```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```



Algoritmo em C like

```

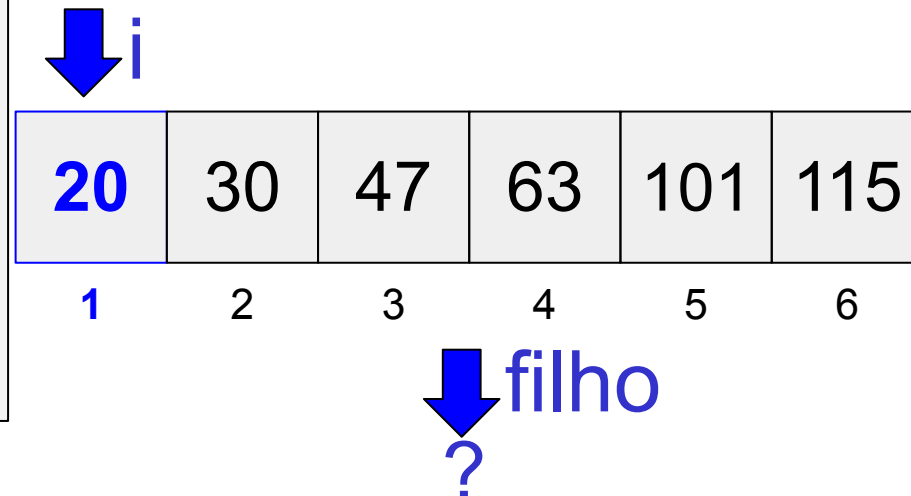
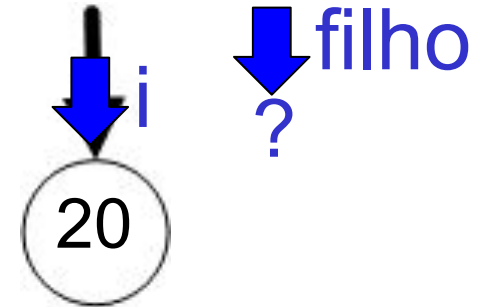
    ...
    int tam = n;
    while (tam > 1){
        swap(1, tam--);
        reconstruir(tam);
    }
  
```

tam 1

```

void reconstruir(int tam){
  int i = 1;
  while (i <= (tam/2)){
    int filho = getMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
      swap(i, filho);
      i = filho;
    } else {
      i = tam;
    }
  }
}
  
```

false: $1 \leq 0$



Algoritmo em C *like*

```

int tam = n;
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}

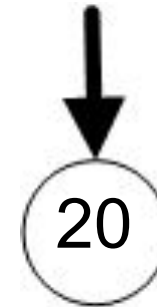
```

tam 1

```

void reconstruir(int tam){
    int i = 1;
    while (i <= (tam/2)){
        int filho = getMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



20	30	47	63	101	115
1	2	3	4	5	6

Algoritmo em C *like*

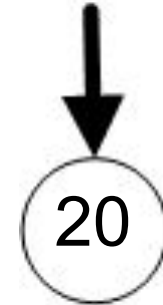
```

int tam = n;
...
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}

```

tam **1**

false: 1 > 1



20	30	47	63	101	115
1	2	3	4	5	6

Algoritmo em C *like*

```
int tam = n;           ...
while (tam > 1){
    swap(1, tam--);
    reconstruir(tam);
}
```

20	30	47	63	101	115
1	2	3	4	5	6

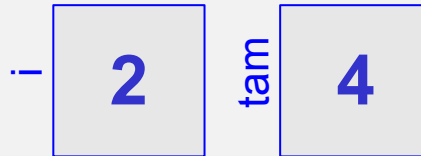
Exercício Resolvido (8)

- Implemente o método `int` `getMaiorFilho(int i, int tam)` apresentado anteriormente

Exercício Resolvido (8)

- Implemente o método `int getMaiorFilho(int i, int tam)` apresentado anteriormente

```
int getMaiorFilho(int i, int tam){  
    int filho;  
    if (2*i == tam || array[2*i] > array[2*i+ 1]) {  
        filho = 2*i;  
    } else {  
        filho = 2*i + 1;  
    }  
    return filho;  
}
```

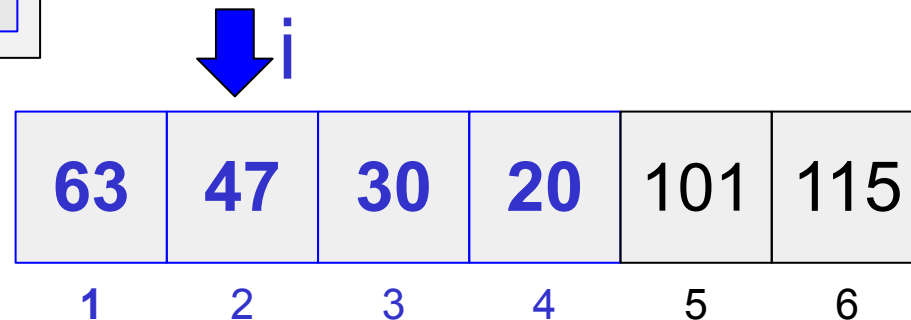
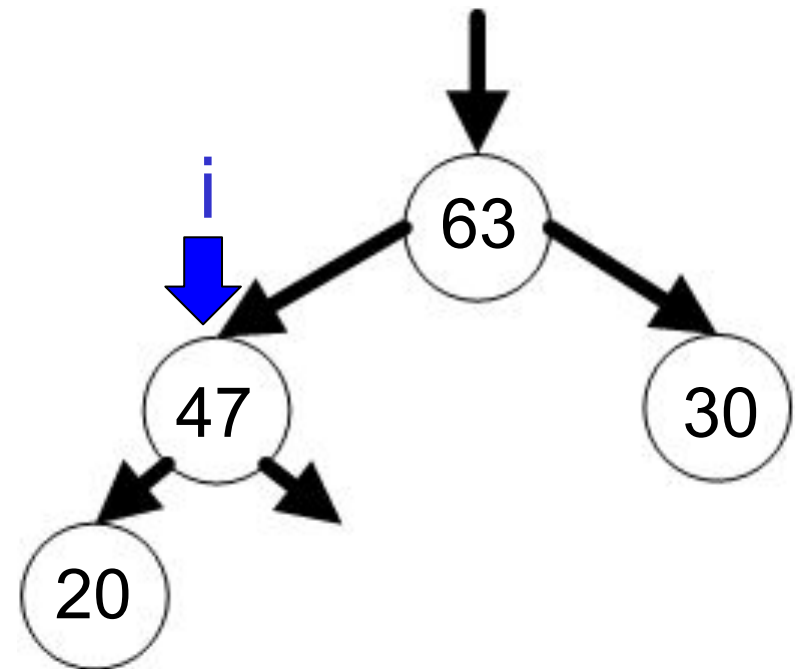
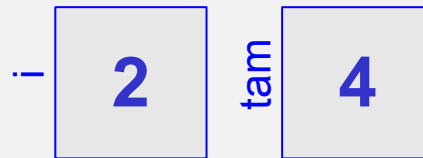



Exercício Resolvido (8)

- Implemente o método `int getMaiorFilho(int i, int tam)` apresentado anteriormente

```

int getMaiorFilho(int i, int tam){
    int filho;
    if (2*i == tam || array[2*i] > array[2*i+ 1]) {
        filho = 2*i;
    } else {
        filho = 2*i + 1;
    }
    return filho;
}
    
```



- Definição de Heap
- Funcionamento básico
- Algoritmo em *C like*
- **Análise dos número de comparações e movimentações** 

Análise do Número de Comparações

- As operações de inserção e remoção podem percorrer um ramo completo da árvore, com comparações e trocas em cada nó
- O pior caso para os números de comparações ou trocas depende da altura da árvore que será $\lceil \lg(n) \rceil$ (árvore balanceada)
- Assim, no pior caso, os números de comparações ou trocas serão $\Theta(\lg(n))$

Análise do Número de Comparações

- Criamos o heap com n inserções e efetuamos $(n-1)$ remoções para ordenar o *array*, assim, para todos os casos, temos:

$$n * \Theta(\lg(n)) + (n-1) * \Theta(\lg(n))$$

- Logo:

$$\Theta(n * \lg(n))$$

Análise do Número de Movimentações

- O número de movimentações é três vezes o de trocas mais as $(n-1)$ movimentações correspondentes às remoções
- Como o número de trocas tem seu limite superior dado pelo de comparações, a complexidade, no pior caso, é $\Theta(n \cdot \lg(n))$

Exercício (4)

- Mostre todas as comparações e movimentações do algoritmo anterior para o *array* abaixo:

12	4	8	2	14	17	6	18	10	16	15	5	13	9	1	11	7	3
----	---	---	---	----	----	---	----	----	----	----	---	----	---	---	----	---	---