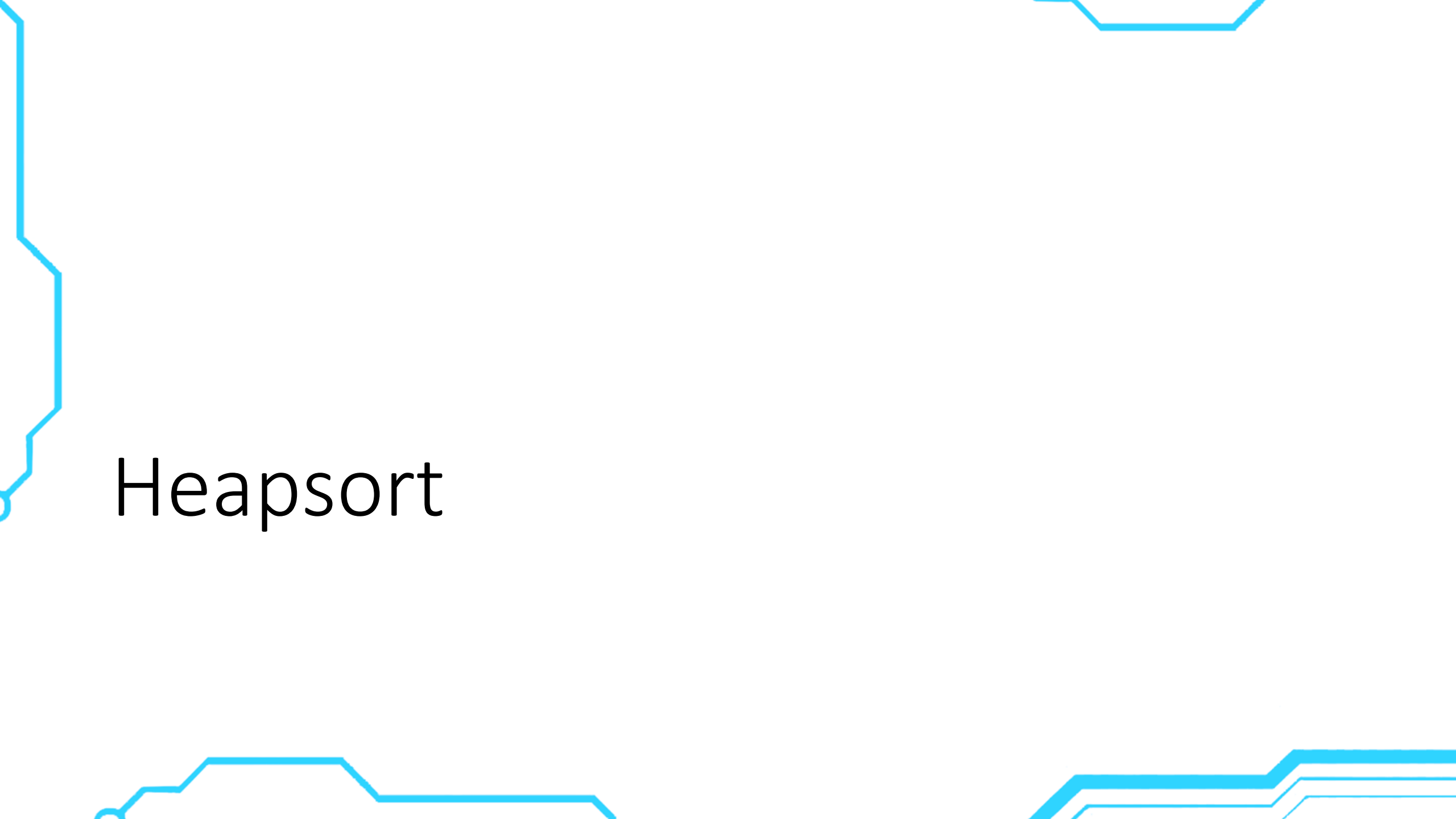


Classificação e Pesquisa de Dados

Cristiano Santos

cristiano.santos@amf.edu.br



Heapsort



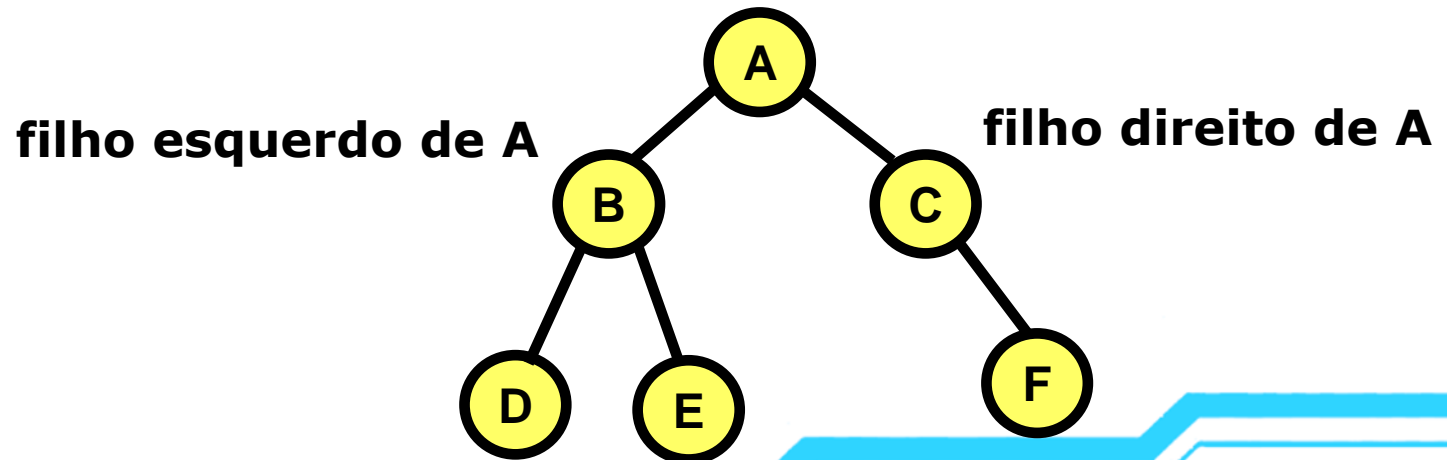
Heap – Revisão

Heap - Revisão

- **Heap:** Tipo específico de árvore binária
- Árvore **Binária:**
 - É uma árvore onde cada nó pode conter **nenhum, 1 ou 2 filhos apenas**
 - **Grau 2**
- Uma árvore binária é formada por nós onde cada nó contém uma referência para outros 2 nós:
 - **Filho esquerdo**
 - **Filho direito**

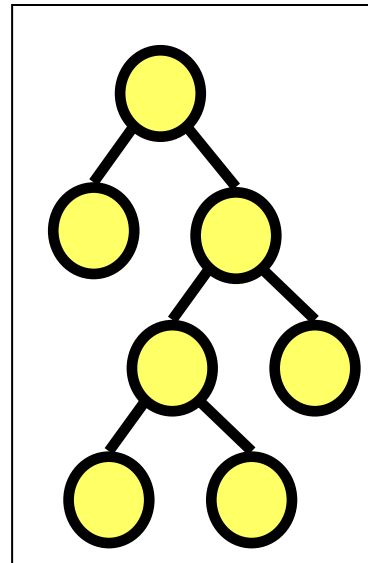
Heap - Revisão

- **Heap:** Tipo específico de árvore binária
- **Árvore Binária:**
 - É uma árvore onde cada nó pode conter nenhum, 1 ou 2 filhos apenas
 - Grau 2
- Uma árvore binária é formada por nós onde cada nó contém uma referência para outros 2 nós:
 - Filho esquerdo
 - Filho direito



Heap - Revisão

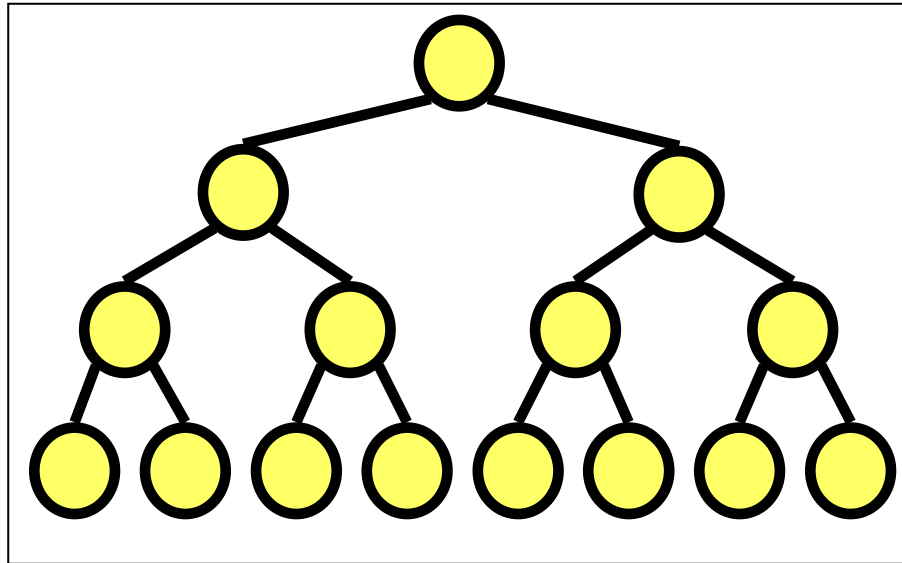
- Árvore **Estritamente** binária:
 - Ocorre quando **todo o nó que não é folha** tiver **sub-árvores esquerda e direita não vazias**
 - Todo nó tem **0 ou 2 filhos**



**Estritamente
Binária**

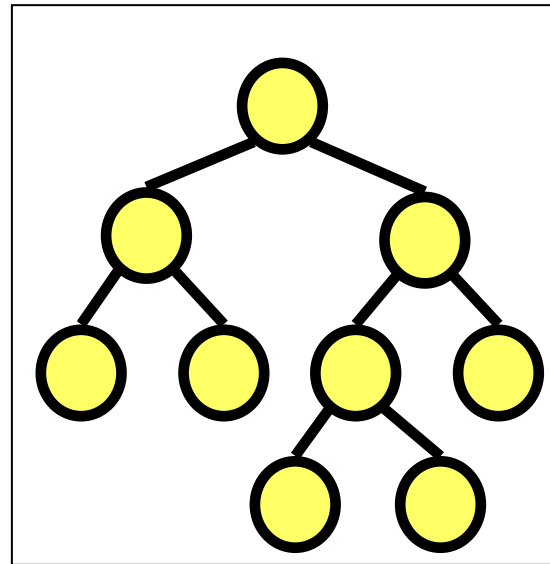
Heap - Revisão

- Árvore binária Cheia:
 - Árvore estritamente binária onde os **nós folhas se encontram no último nível**



Heap - Revisão

- Árvore binária **Completa**:
 - É uma árvore estritamente binária onde todos os nós folhas se encontram ou no **último** ou no **penúltimo** nível da árvore.



**Binária
Completa**

Heap - Revisão

- **Heap:**

- **É uma “Estrutura de prioridades”, na forma de árvore binária completa, que representa uma ordem parcial entre os elementos do conjunto;**

Heap - Revisão

- **Heap:**

- É uma “Estrutura de prioridades”, na forma de **árvore binária completa**, que **representa uma ordem parcial** entre os elementos do conjunto.
- A árvore está **complemente preenchida** em todos os níveis exceto talvez o mais baixo;

Heap - Revisão

- **Heap:**

- É uma “Estrutura de prioridades”, na forma de **árvore binária completa**, que **representa uma ordem parcial entre os elementos do conjunto**;
- A **árvore está completamente preenchida em todos os níveis exceto talvez o mais baixo**;
- **Nível mais baixo preenchido a partir da esquerda.**

Heap - Revisão

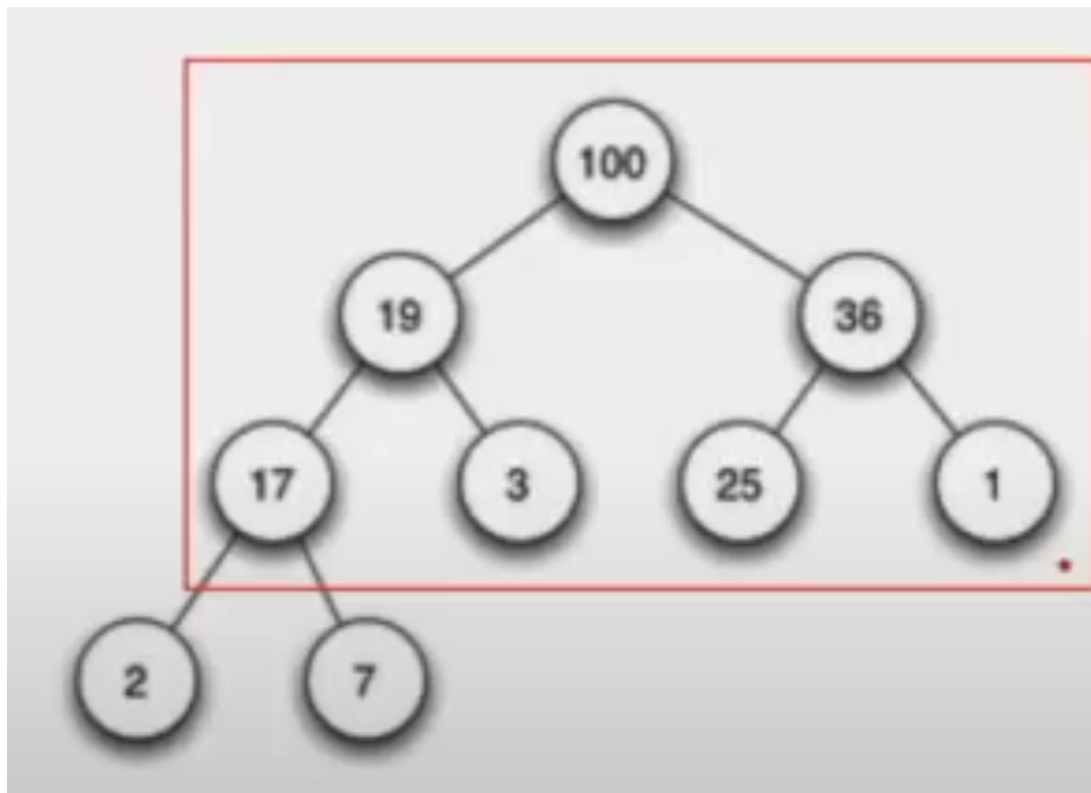
- **Heap:**

- É uma “Estrutura de prioridades”, na forma de árvore binária completa, que representa uma ordem parcial entre os elementos do conjunto;
- A árvore está completamente preenchida em todos os níveis exceto talvez o mais baixo;
- Nível mais baixo preenchido a partir da esquerda.

- **Atenção:** Ao contrário da Árvore Binária de Pesquisa em que existe a ordem total entre os elementos, Heap possui uma ordem parcial entre os elementos do conjunto.

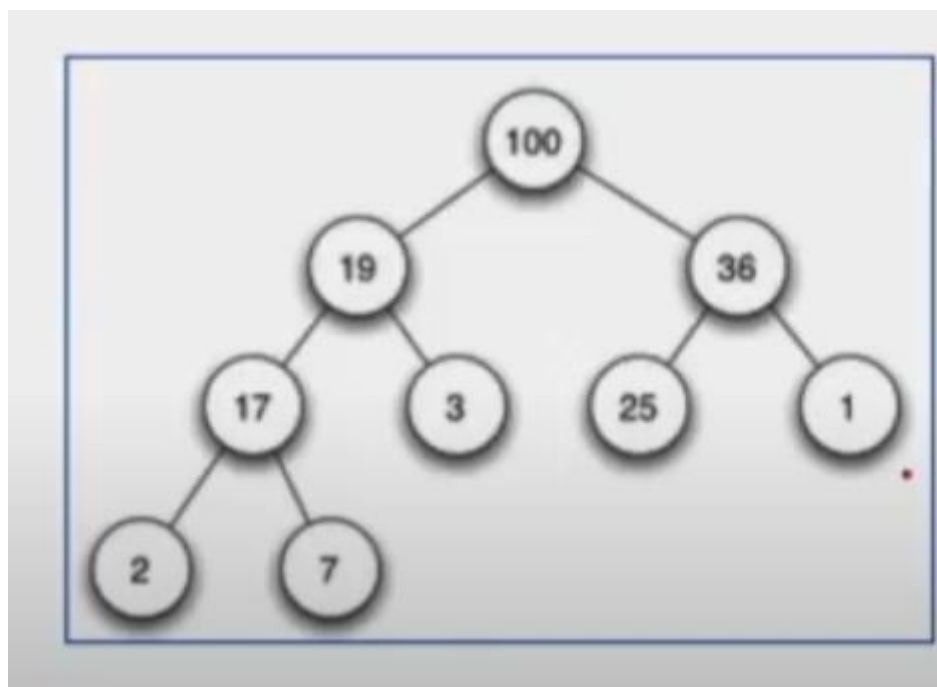
Estrutura de uma Heap

- Ignorando o último nível, temos uma árvore binária cheia (0 ou 2 filhos e que todas as folhas se encontram no mesmo nível).



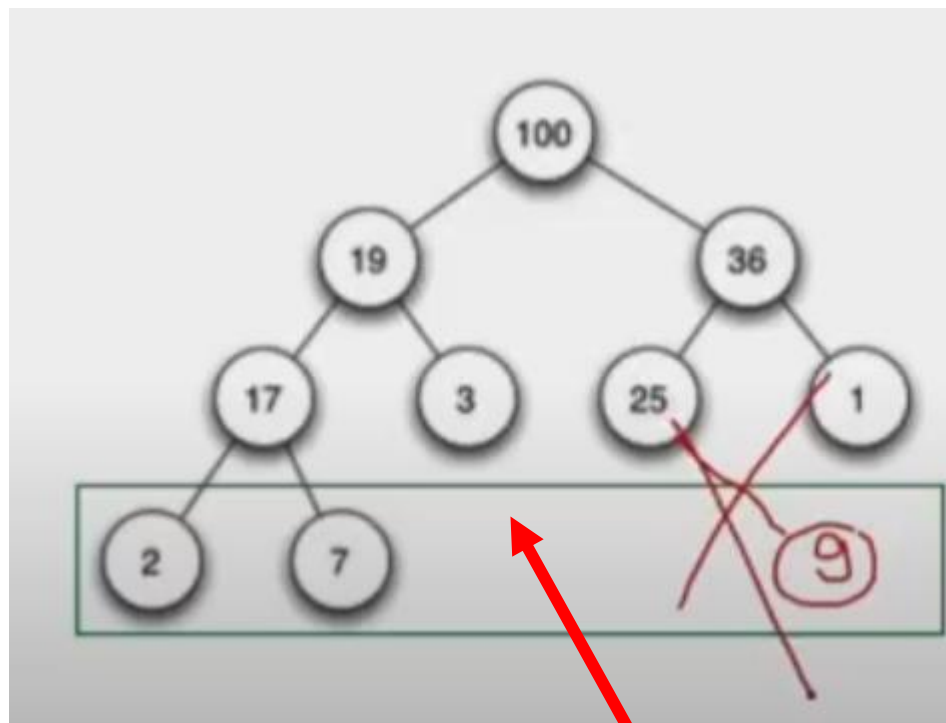
Estrutura de uma Heap

- **Contabilizando o último nível, temos uma árvore binária completa** (nós tem 0 ou 2 filhos, as folhas se encontram no último ou penúltimo nível)



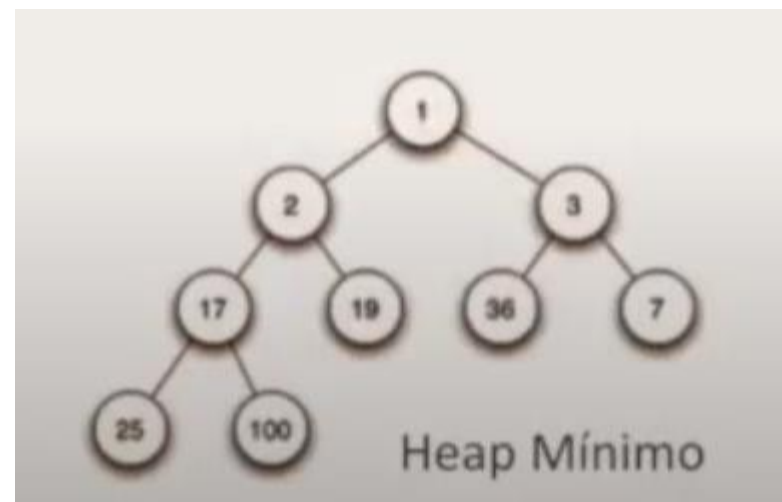
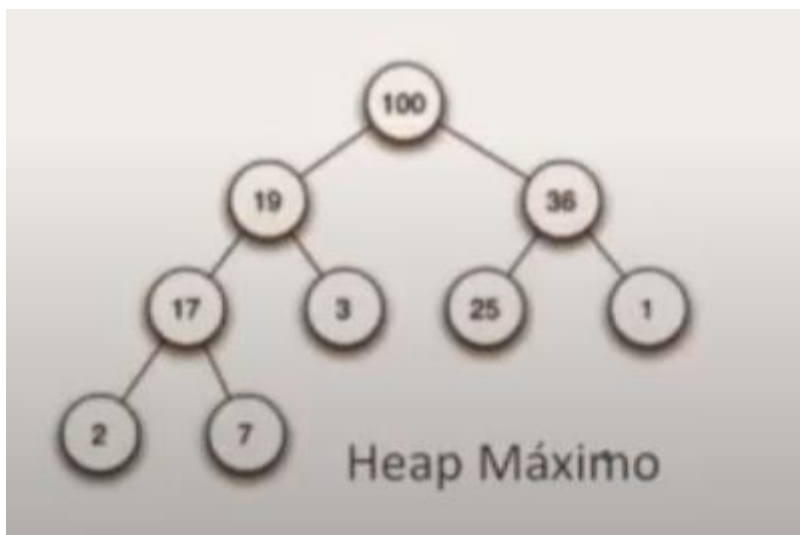
Estrutura de uma Heap

- O último nível não precisa ter todos os nós, mas não deve faltar nenhum nó entre o nó mais à esquerda e o nó mais à direita.



Tipos de Heap

- Dois tipos de heap:
 - Máximo
 - Mínimo

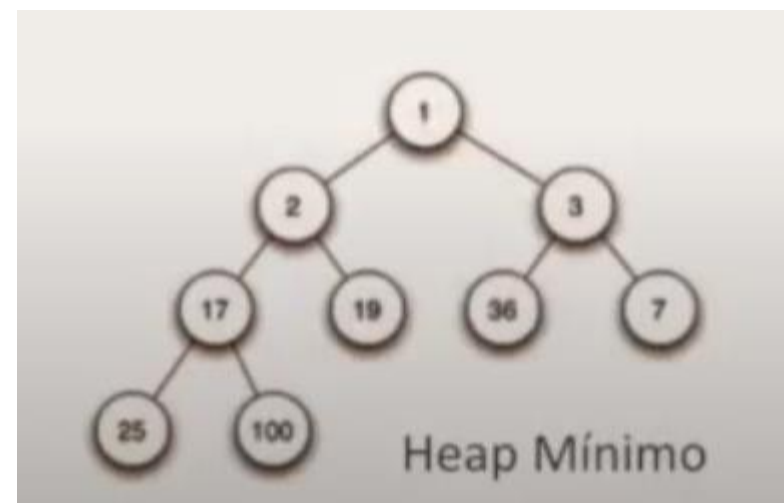
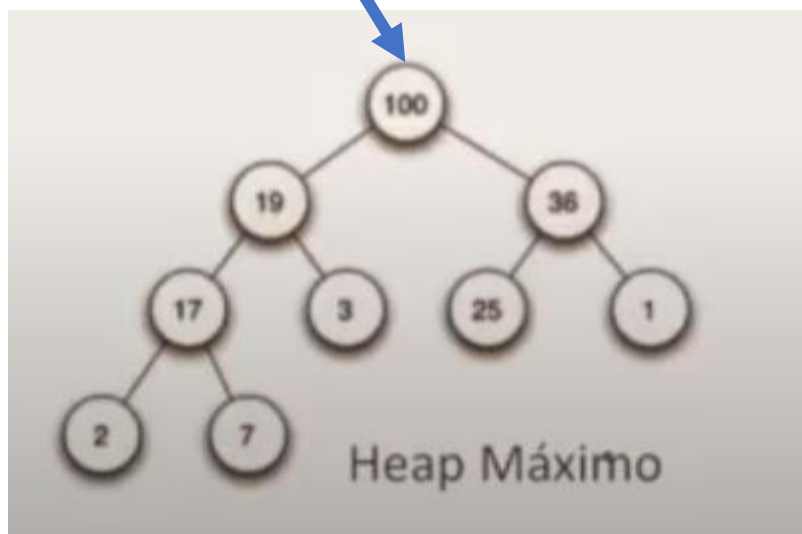


Tipos de Heap

- Dois tipos de heap:

- Máximo

- Mínimo

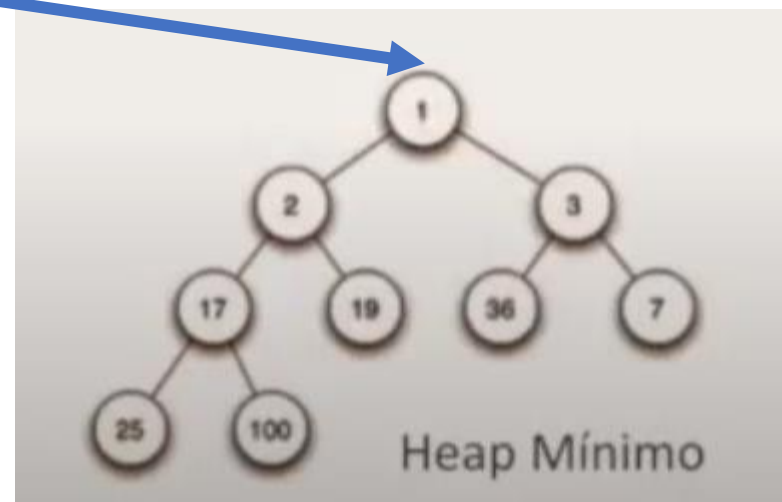
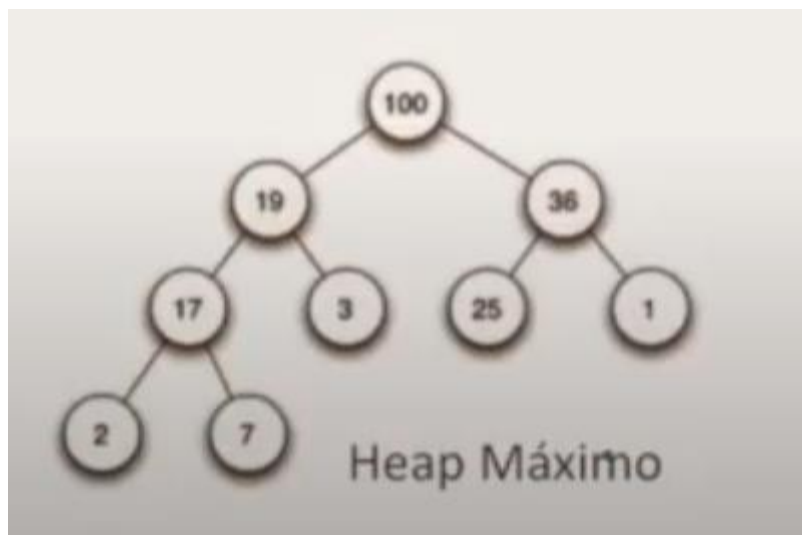


Tipos de Heap

- Dois tipos de heap:

- Máximo

- Mínimo



Heap Máximo

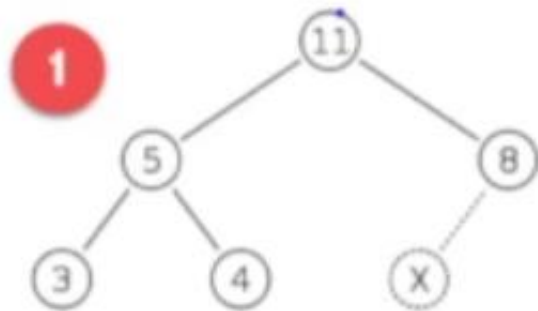
- Para todo nó, **o pai é maior ou igual aos filhos:**
 - $A[i] \geq \text{Filho_esq}[i]$
 - $A[i] \geq \text{Filho_dir}[i]$
- **O maior elemento de um heap máximo está armazenado na raiz**
- **O Heap Máximo é usado em algoritmos de classificação HeapSort**

Heap - Operações

- Inserção
- Remoção
- Busca

Heap - Inserção

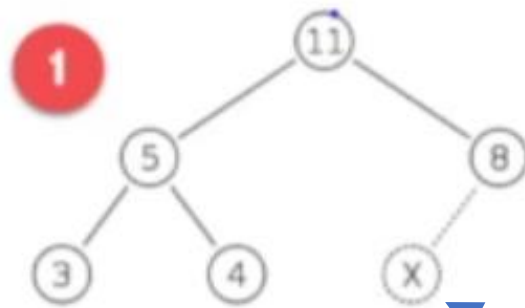
- Exemplo: Inserção da chave “15”



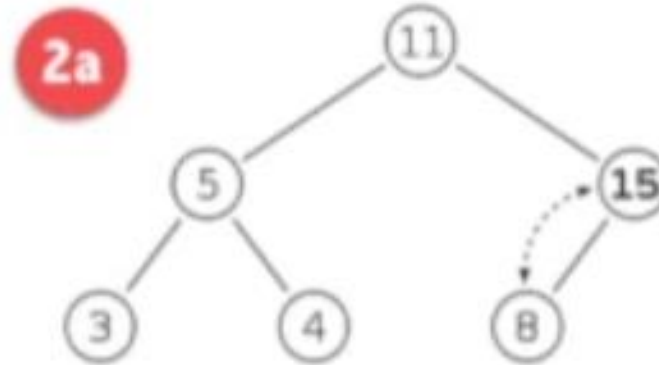
Coloca o **novo nó** no **último nível**, na posição **mais à esquerda** possível

Heap - Inserção

- Exemplo: Inserção da chave “15”



Coloca o **novo nó** no **último nível**, na **posição mais à esquerda possível**

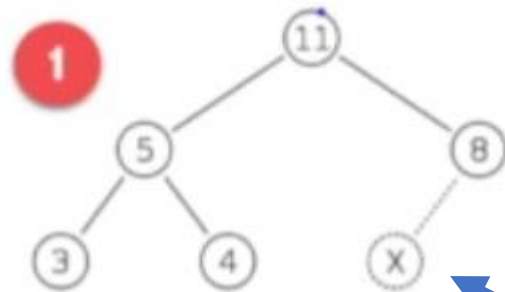


Tenta ordenar
do novo nó
para a raiz
heapfy-up

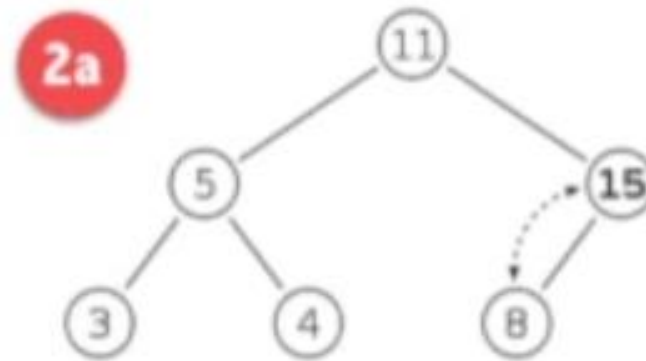
Satisfaz a condição de árvore binária completa, mas não a condição de Heap.

Heap - Inserção

- Exemplo: Inserção da chave “15”



Coloca o novo nó no último nível, na posição mais à esquerda possível



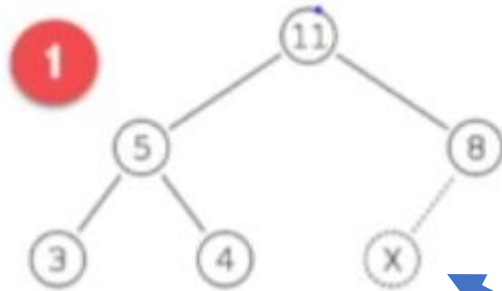
Tenta ordenar do novo nó para a raiz

heapfy-up

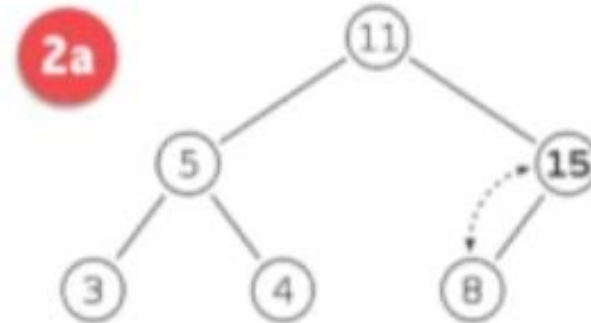
Satisfaz a condição de árvore binária completa, mas ainda não a condição de Heap.

Heap - Inserção

- Exemplo: Inserção da chave “15”



Coloca o **novo nó** no **último nível**, na posição **mais à esquerda** possível



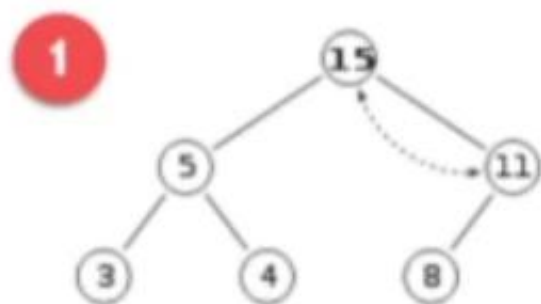
Tenta ordenar
do novo nó
para a raiz

heapfy-up



Heap - Remoção

- Exemplo: da raiz (geralmente)

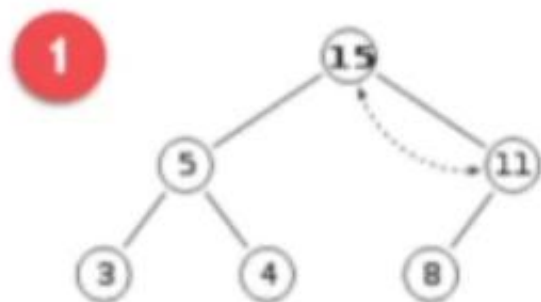


Em um heap máximo, maior chave encontra-se na raiz

Começa pela raiz – maior elemento neste caso

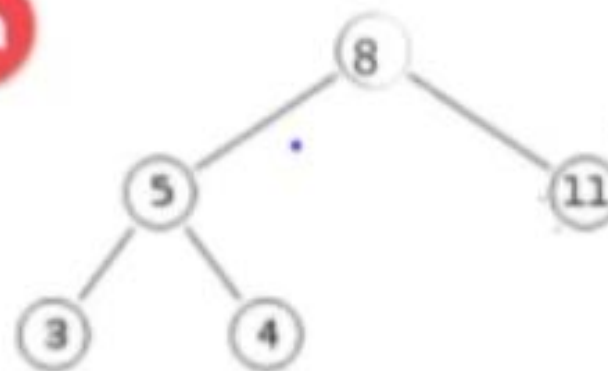
Heap - Remoção

- Exemplo: da raiz (geralmente)



Em um heap máximo, maior chave encontra-se na raiz

2a



Substitui raiz
pelo último
nó no último
nível

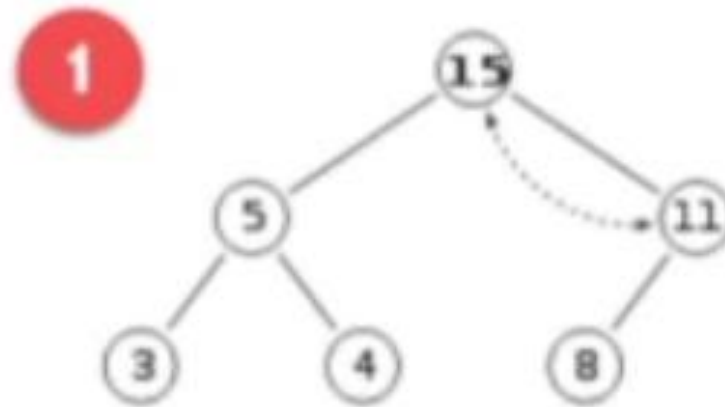
2b



Heapfy-down

Heap – Busca

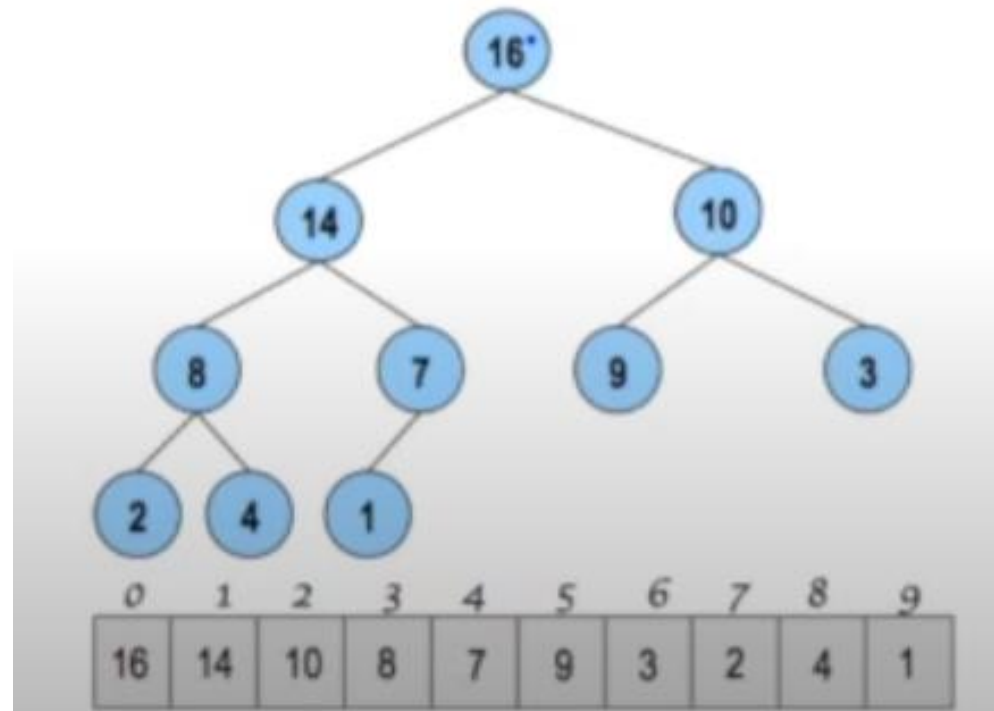
- Exemplo: da raiz (geralmente)

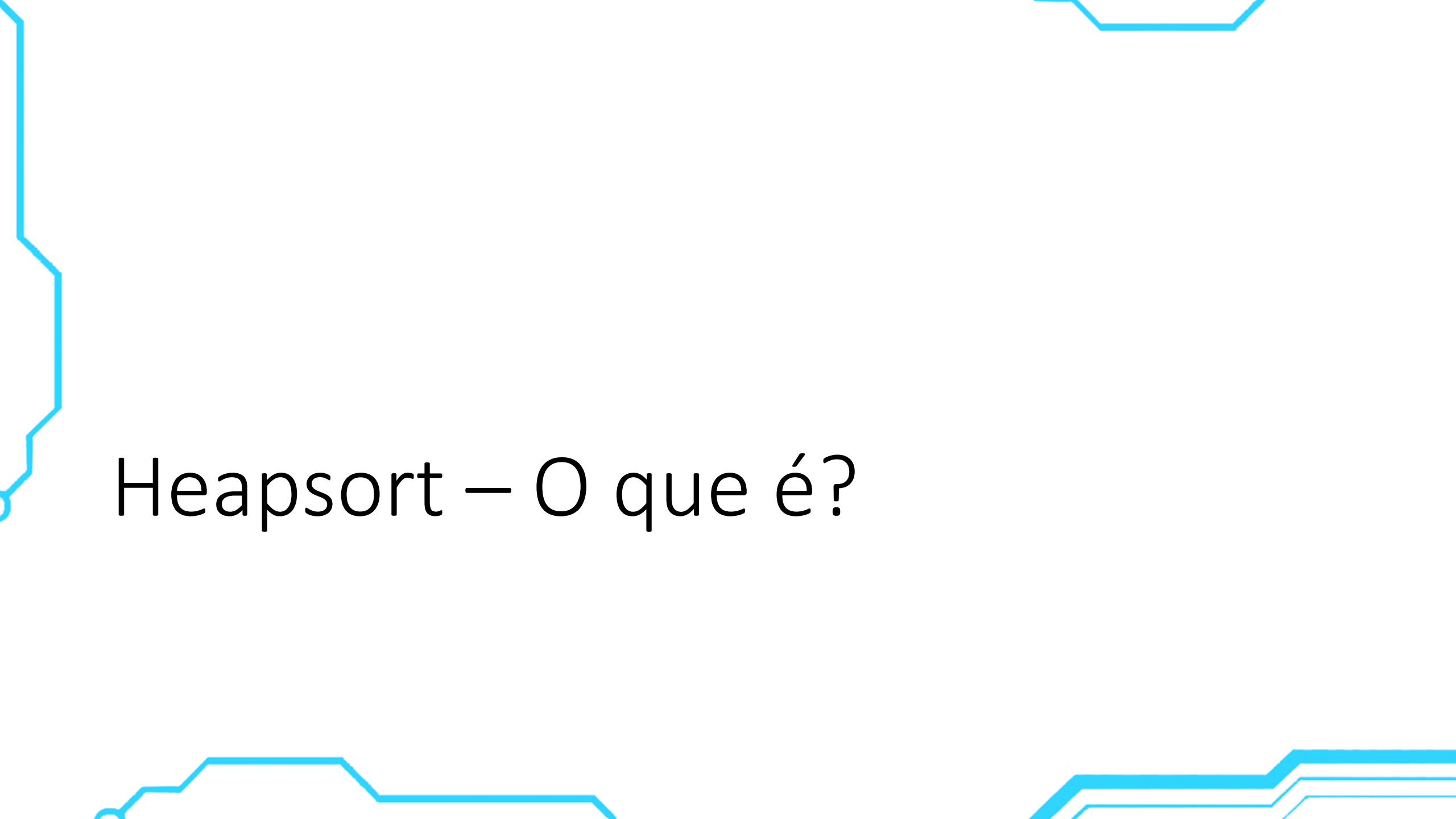


Em um heap máximo, maior chave encontra-se na raiz

Heap – Representação

- Um heap pode ser apresentado por um array unidimensional:
 - Raiz = 0
 - $\text{Pai}(i) = (i-1)/2$
 - $\text{Filho_esq}(i) = i * 2 + 1$
 - $\text{Filho_dir}(i) = i * 2 + 2$





Heapsort – O que é?

HeapSort

- Desenvolvido em 1964 por Robert W. Floyd (New York, USA) e J.W.J Williams;
- **Método de seleção em árvore binária do tipo Heap, de forma ordenada em relação aos valores de suas chaves;**

HeapSort

- Também conhecido como ordenação por “heap” (do inglês, monte), é um algoritmo de ordenação bastante sofisticado e que **compete em desempenho com o quicksort**.
- A **ideia** básica desse algoritmo é **transformar o array de dados em uma estrutura do tipo heap**.

HeapSort

- Consiste em duas fases:
 - Fase 1: **Construção do heap** (build-Heap) – **Heap máximo**
 - Fase 2: **Seleção dos elementos na ordem desejada** (Heapsort)

HeapSort

Fase 1

- **Dado um array, trocar as chaves de lugar de forma que a árvore representada pela array passe a ser um heap**
- **Para toda sub-árvore:**
$$A[\text{Pai}(i)] \geq A[i]$$
- Os testes das chaves se iniciam pela última sub-árvore prosseguindo, a partir daí, para as sub-árvores que antecedem essa, até testar a raiz da árvore.

HeapSort

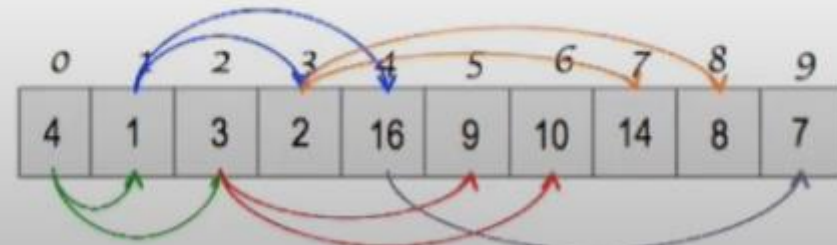
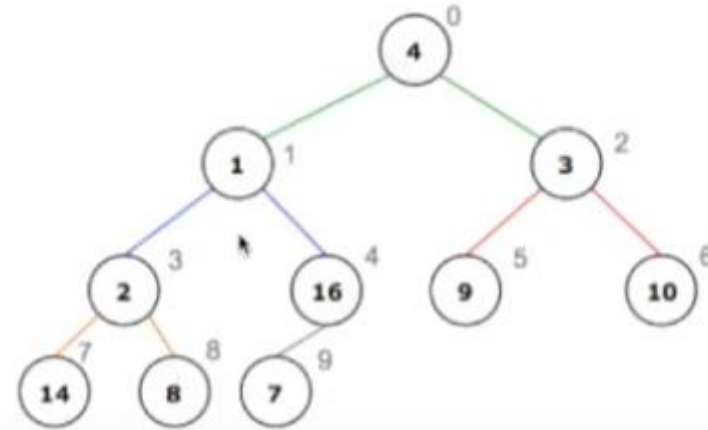
Fase 1: Exemplo

Raiz = 0

$\text{Pai}(i) = (i-1)/2$

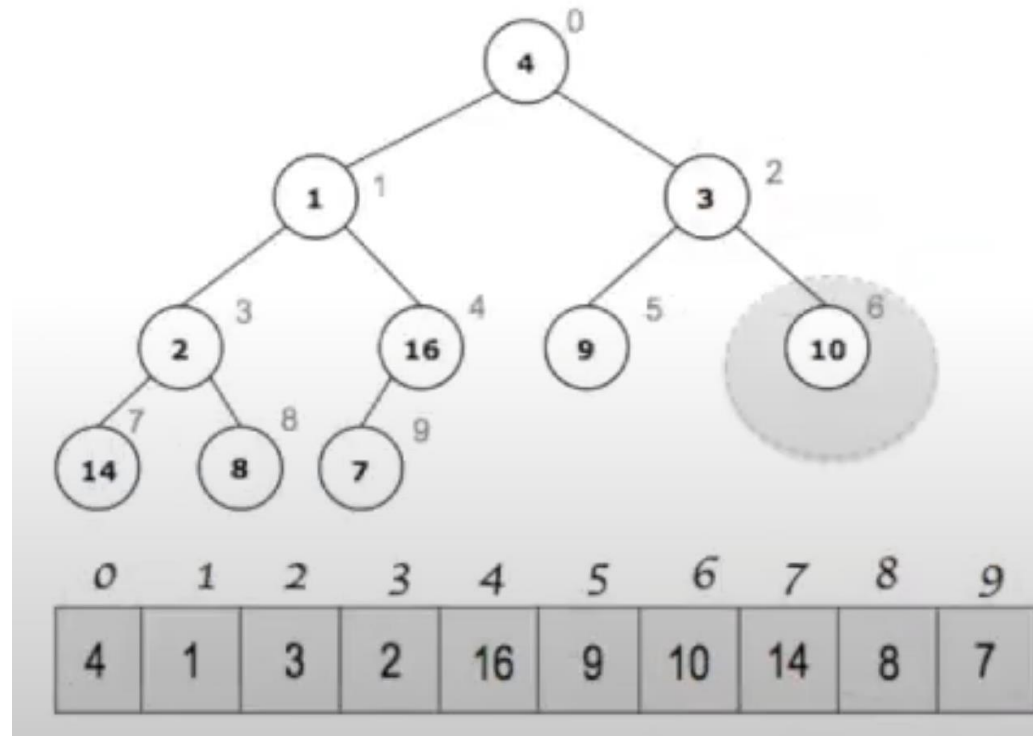
$\text{Filho_esq}(i) = i * 2 + 1$

$\text{Filho_dir}(i) = i * 2 + 2$



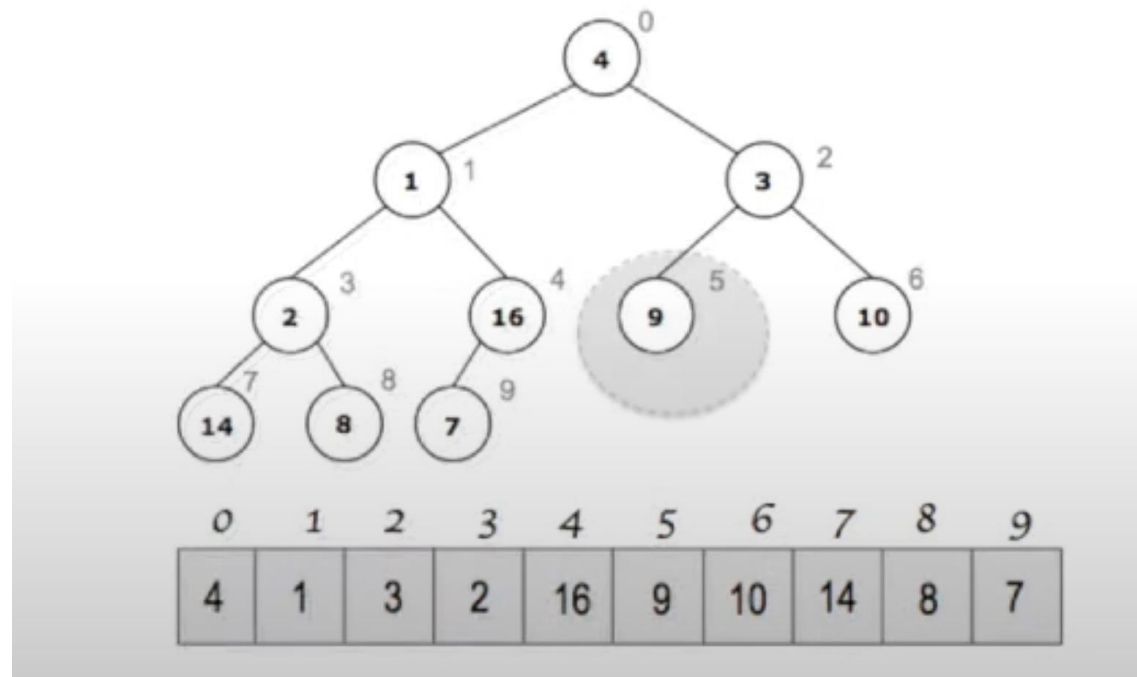
HeapSort

- Primeira fase consiste em transformar um árvore binária em Heap
- **Nota**: Heapsort trabalha com array (aqui a árvore é representativa)



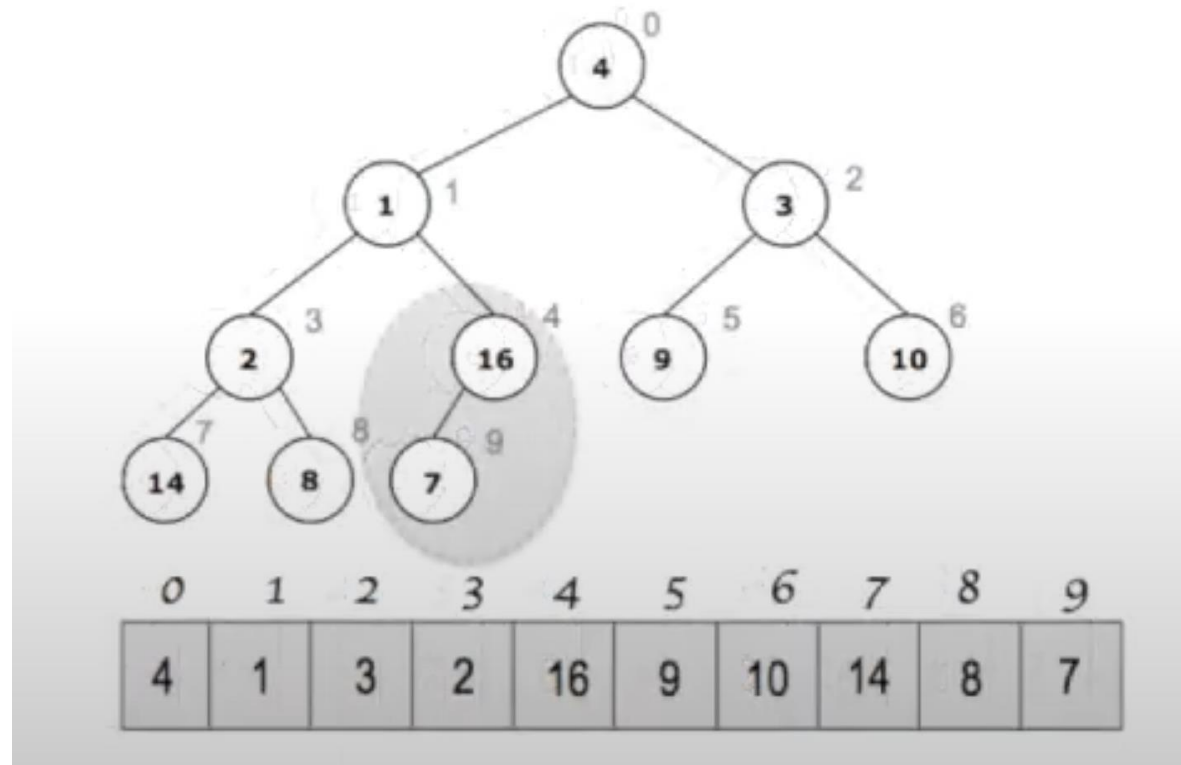
HeapSort

- Primeira fase consiste em transformar um árvore binária em Heap



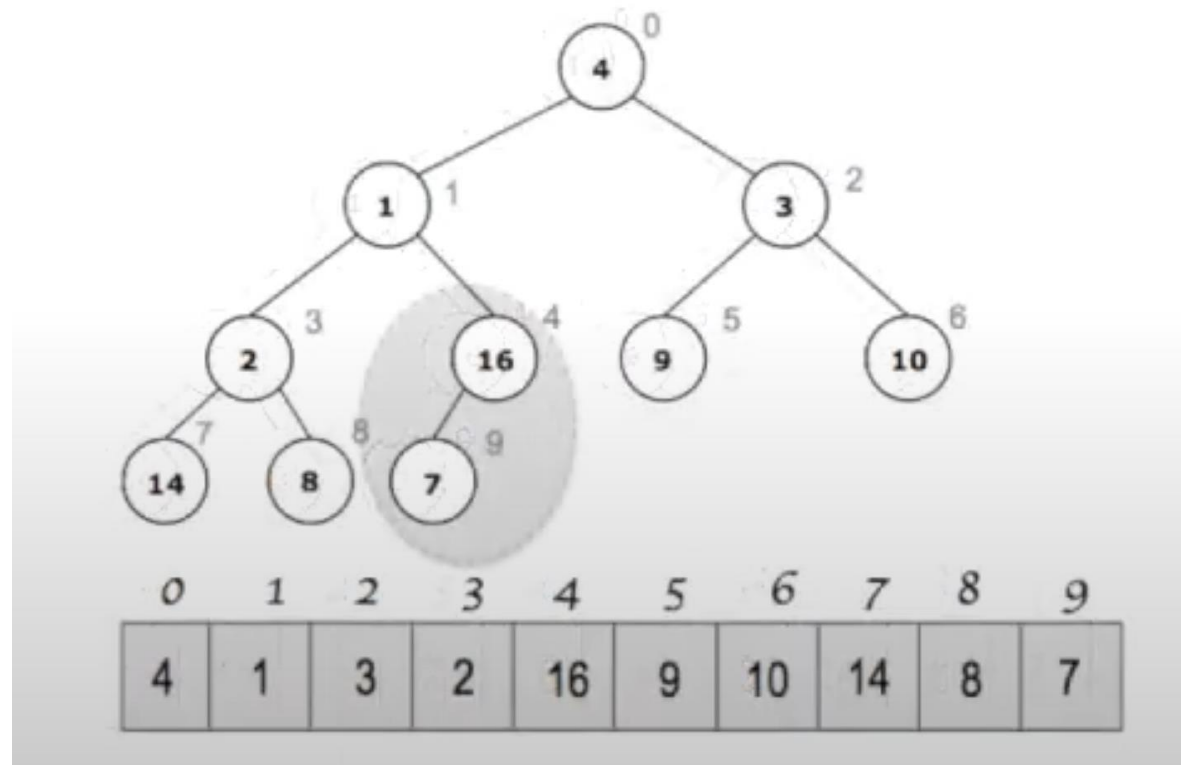
HeapSort

- Primeira fase consiste em transformar um árvore binária em Heap
- Começa pelo $\text{tamanho_array}/2-1$



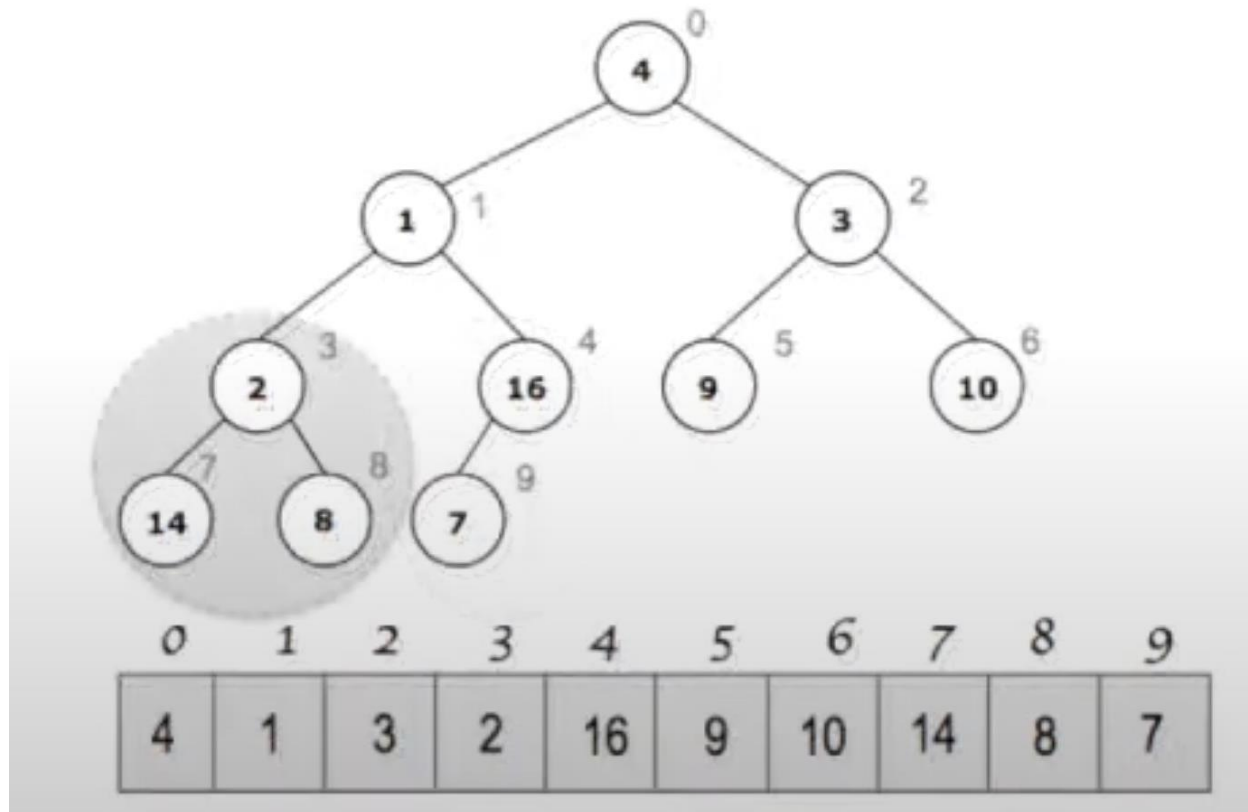
HeapSort

- Primeira fase consiste em transformar um array em uma árvore de heap. Característica importante!
- Começa pelo tamanho_array/2-1



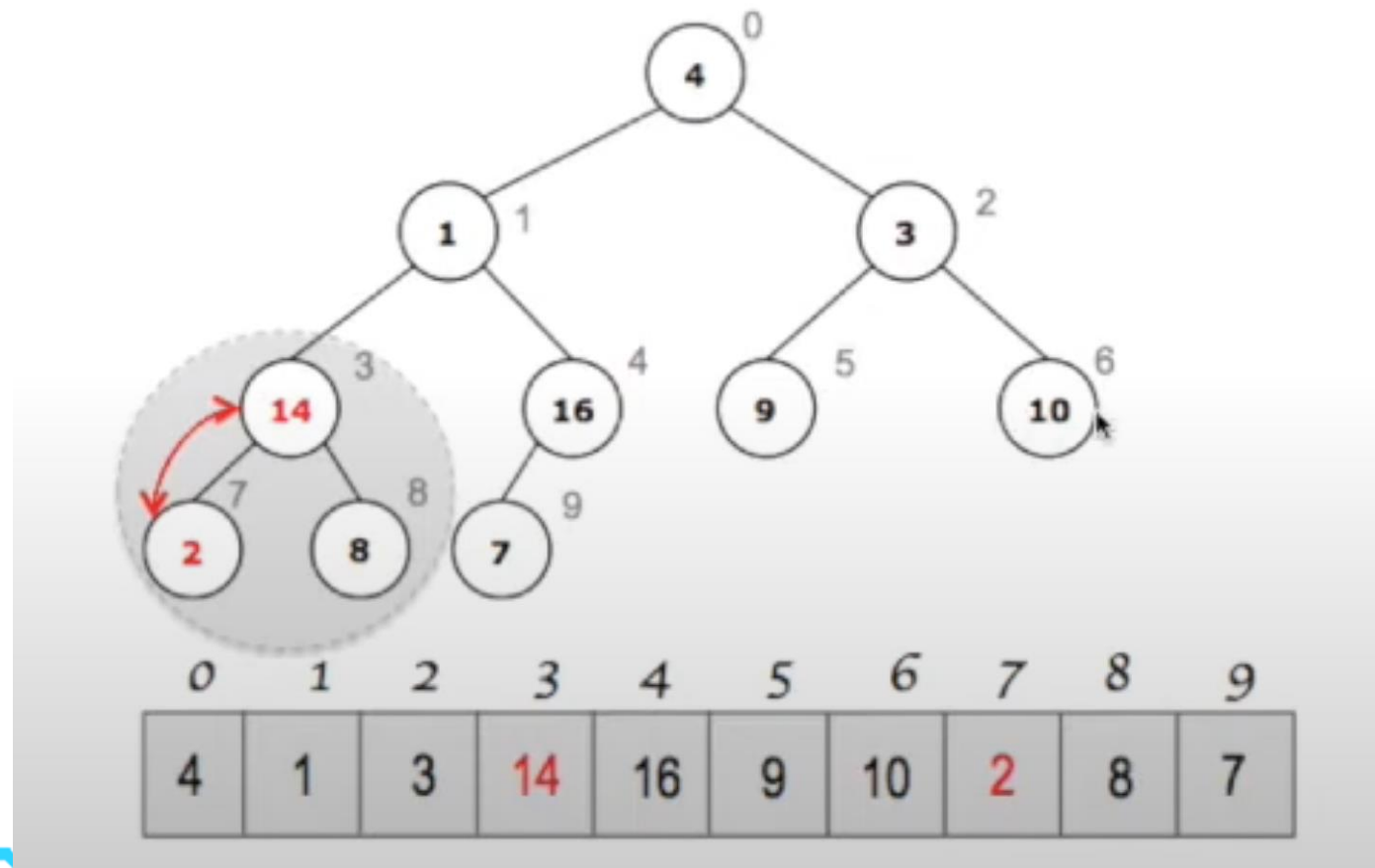
HeapSort

- Neste caso, essa sub-árvore não é uma Heap



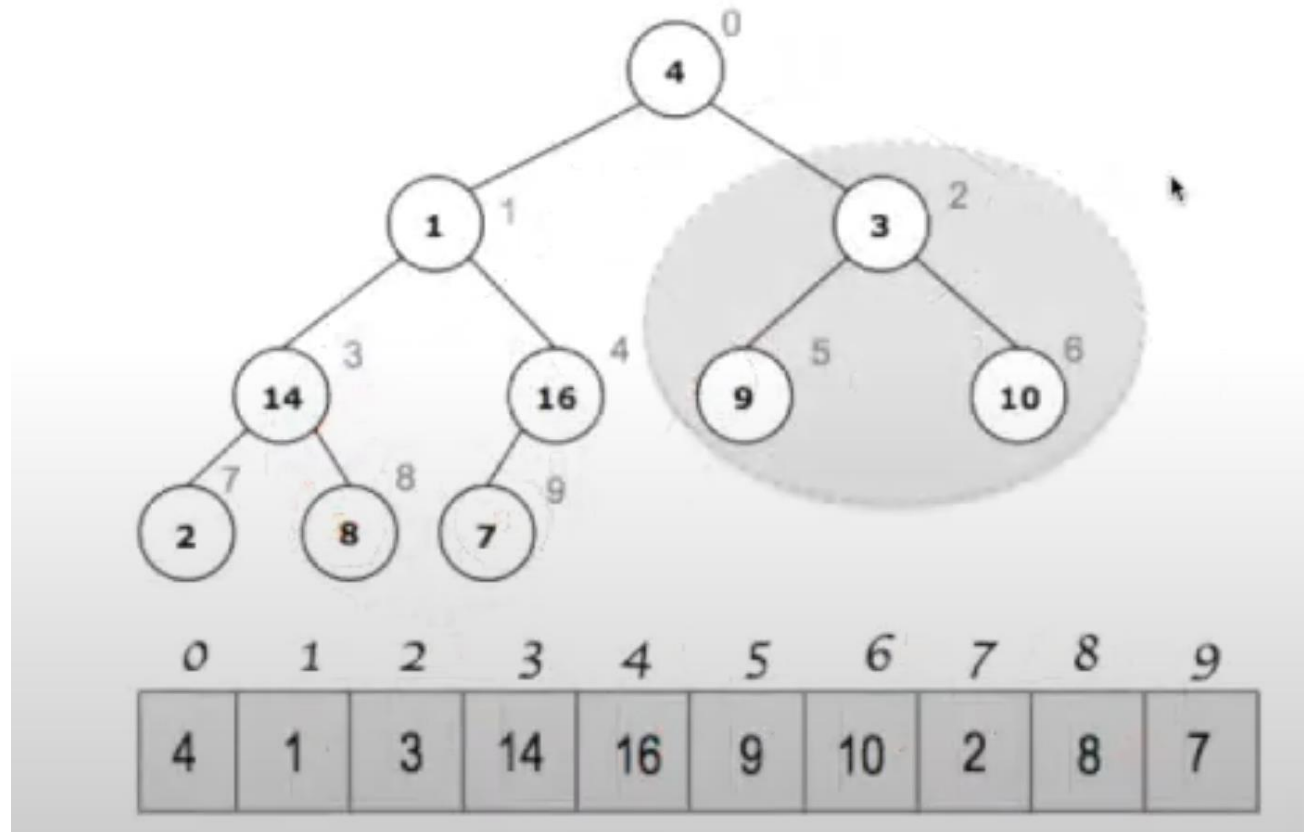
HeapSort

- Troca de lugar com o maior elemento entre os filhos



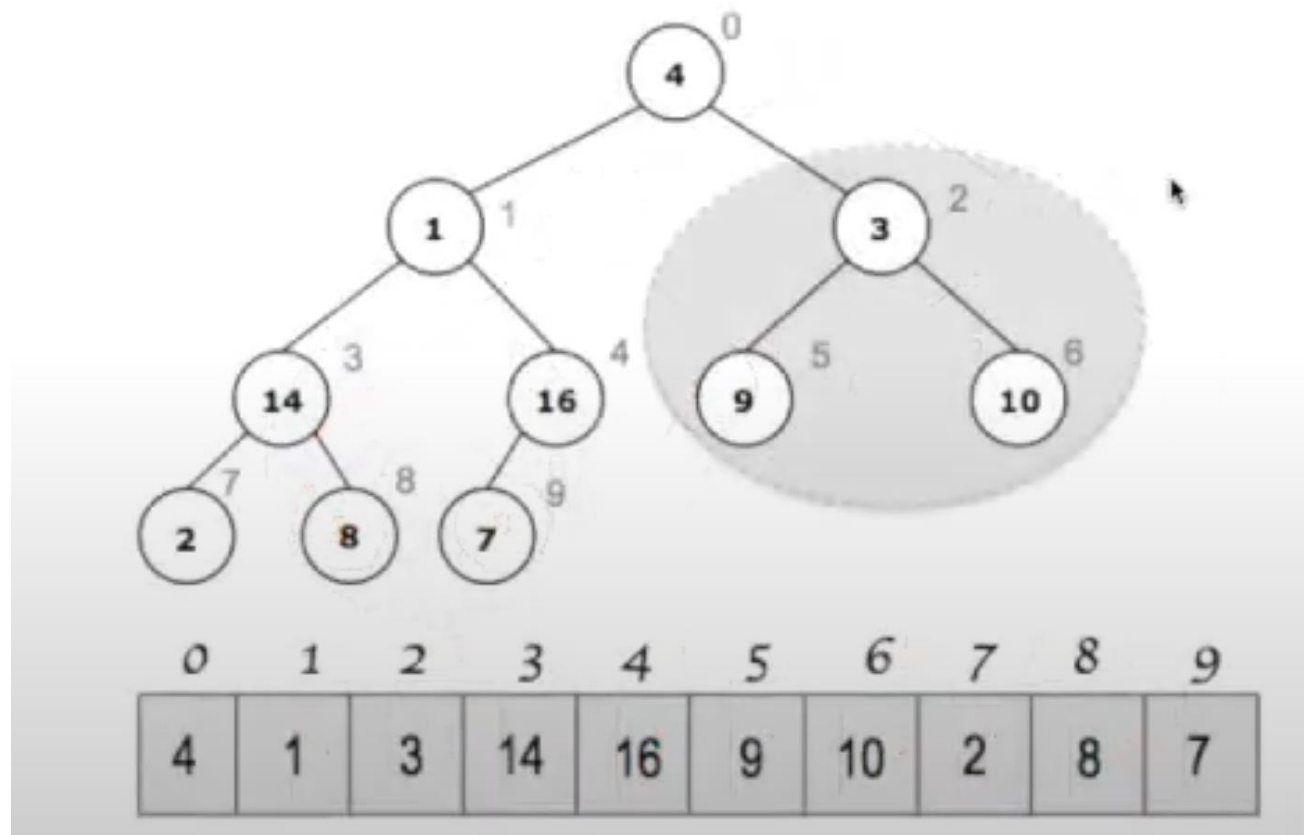
HeapSort

- Próximo nível, da direita para esquerda



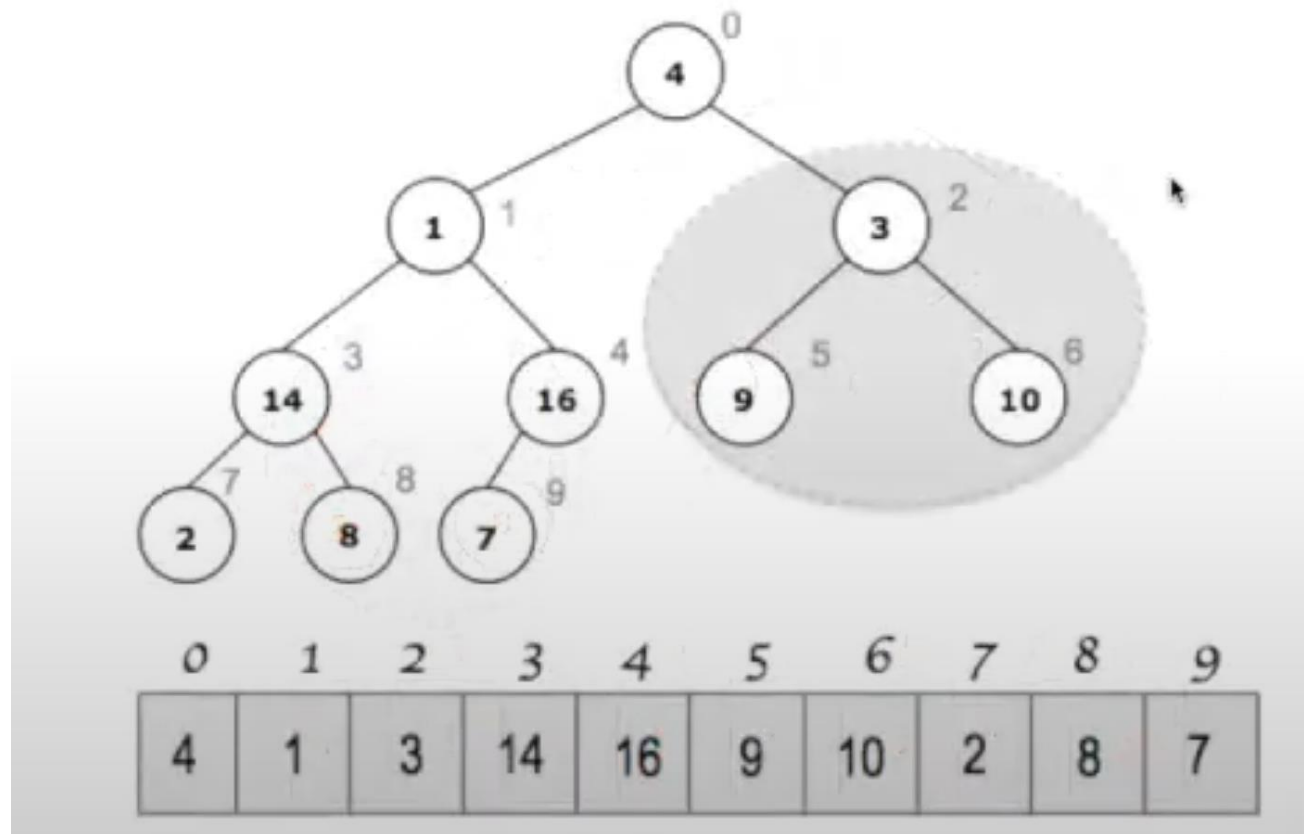
HeapSort

- Não é uma Heap



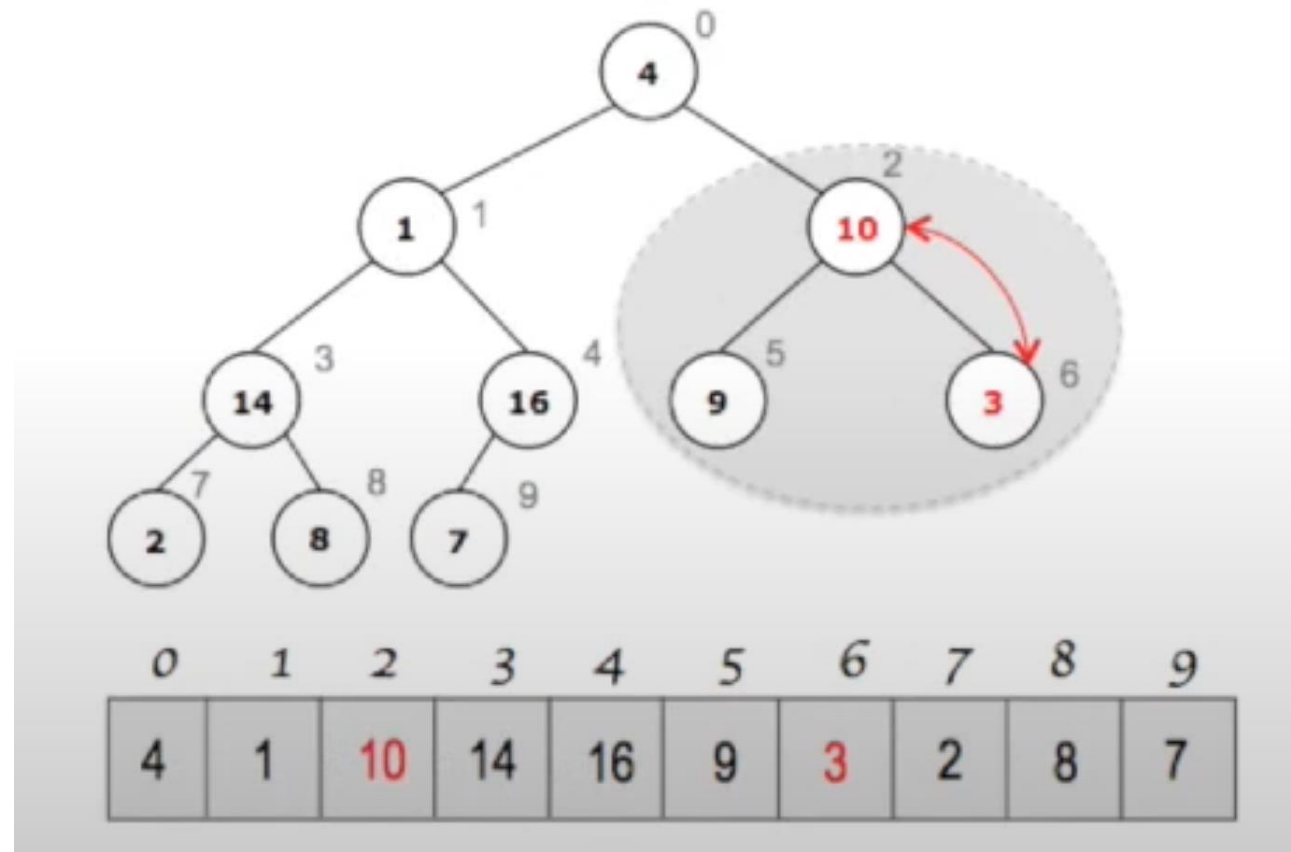
HeapSort

- Pegamos a maior chave entre os filhos e trocamos



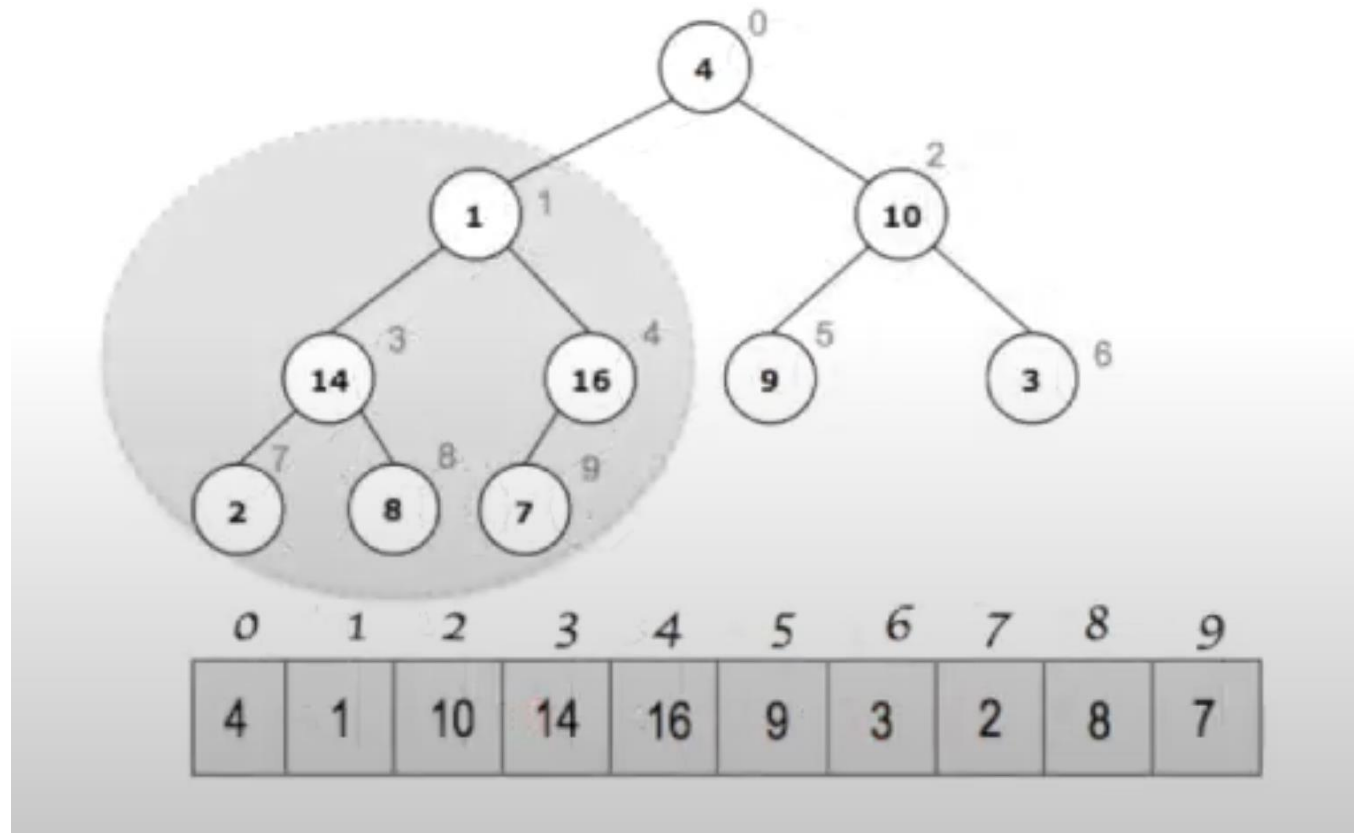
HeapSort

- Pegamos a maior chave entre os filhos e trocamos de lugar com o pai (elemento 2)



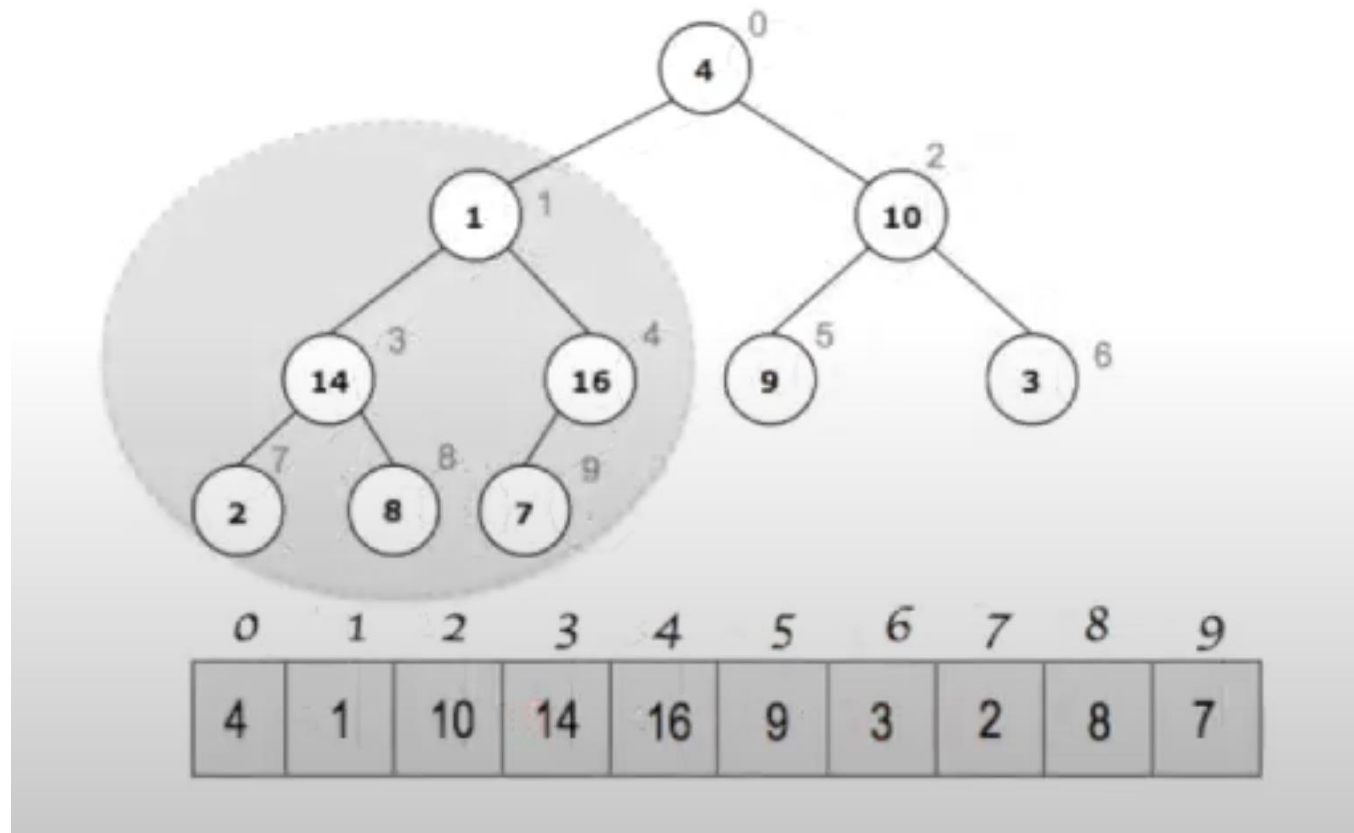
HeapSort

- Avalia sub-árvore da esquerda



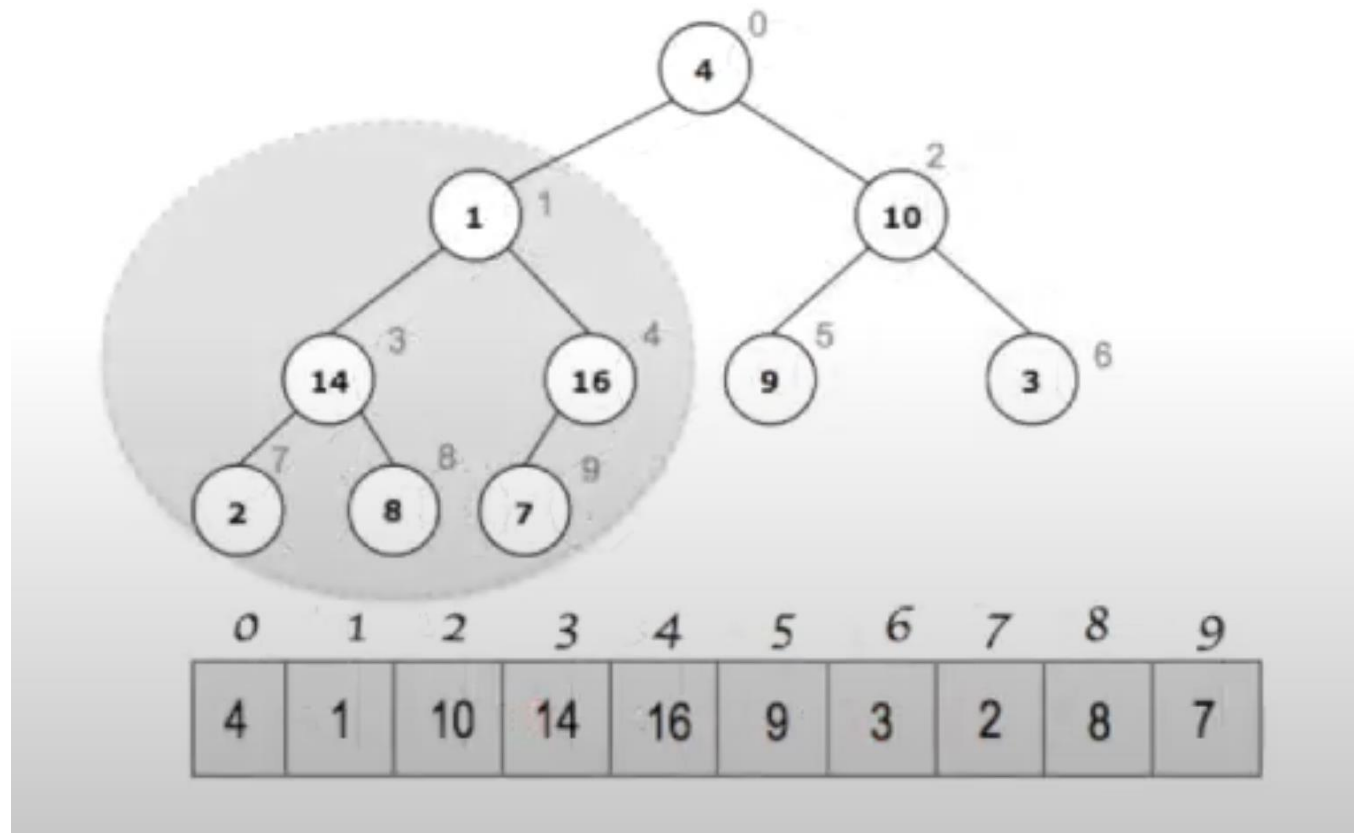
HeapSort

- Não é uma Heap



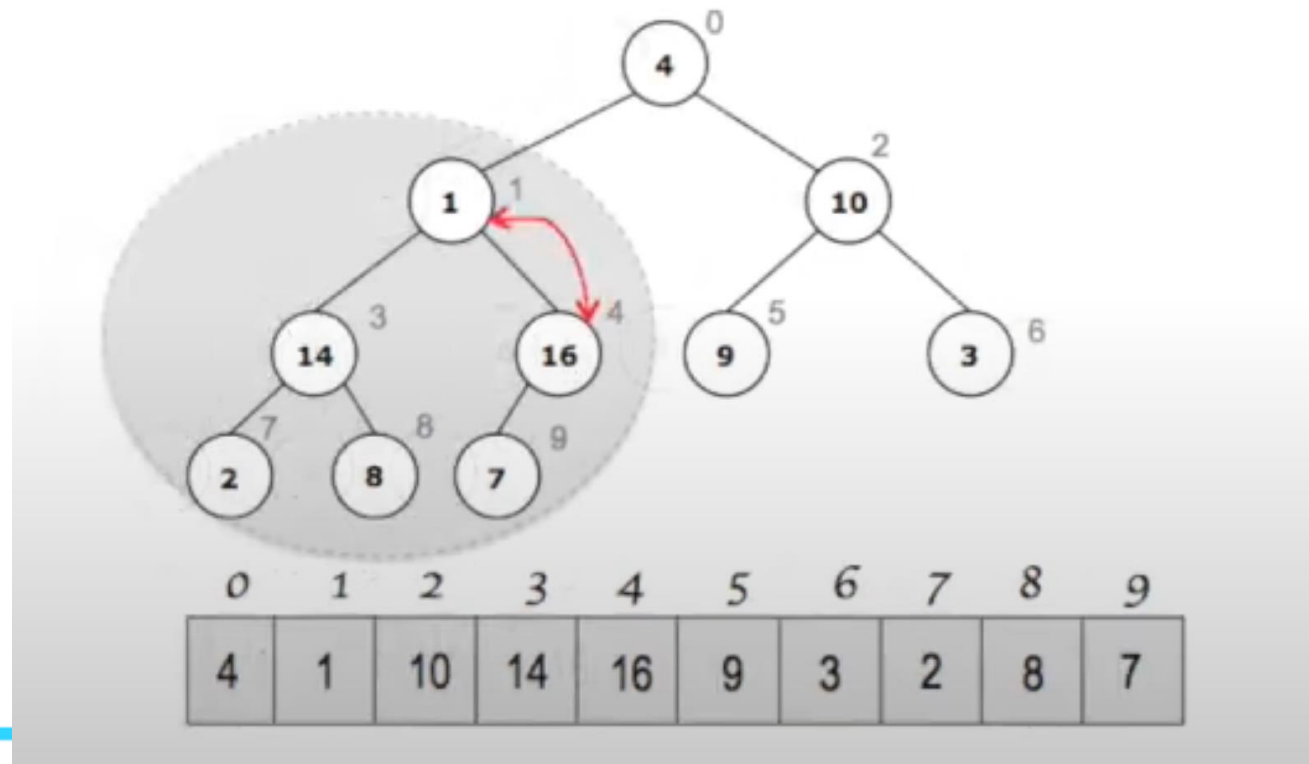
HeapSort

- Começamos com a raiz, entre os filhos qual o maior entre eles?



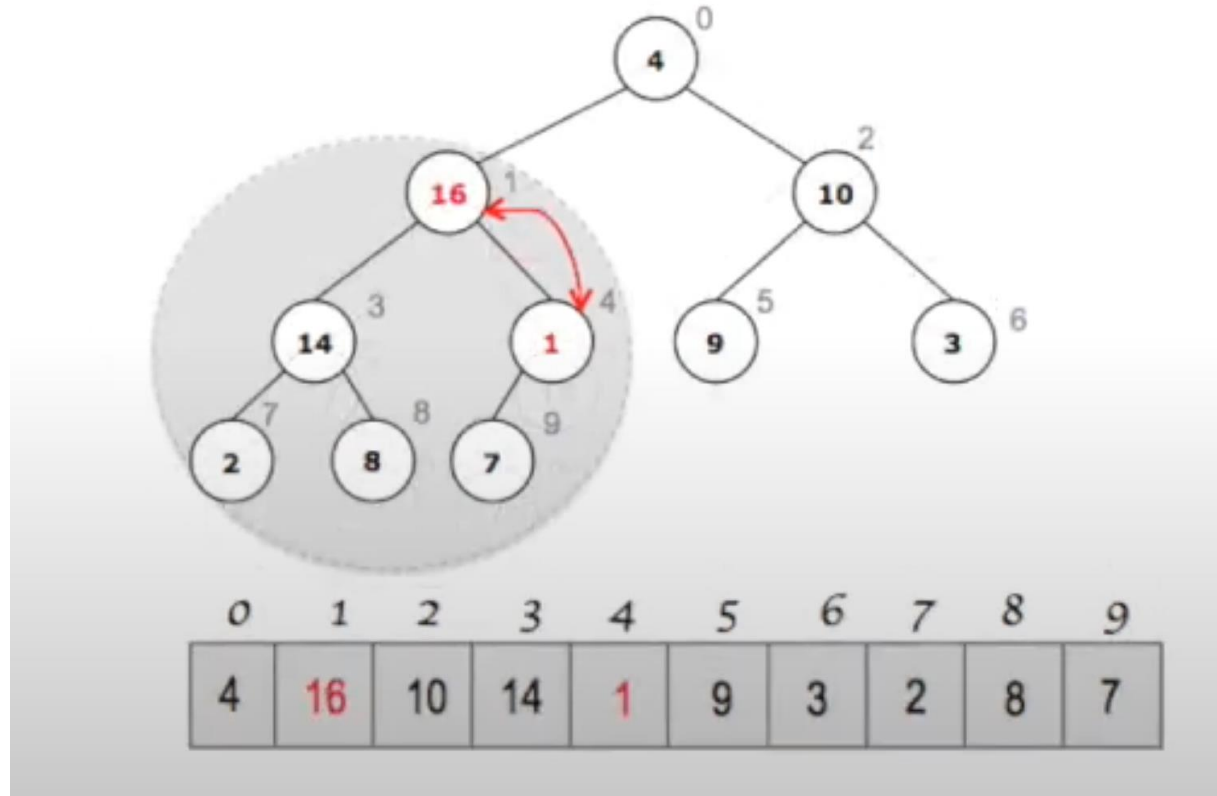
HeapSort

- Começamos com a raiz, entre os filhos qual o maior entre eles?



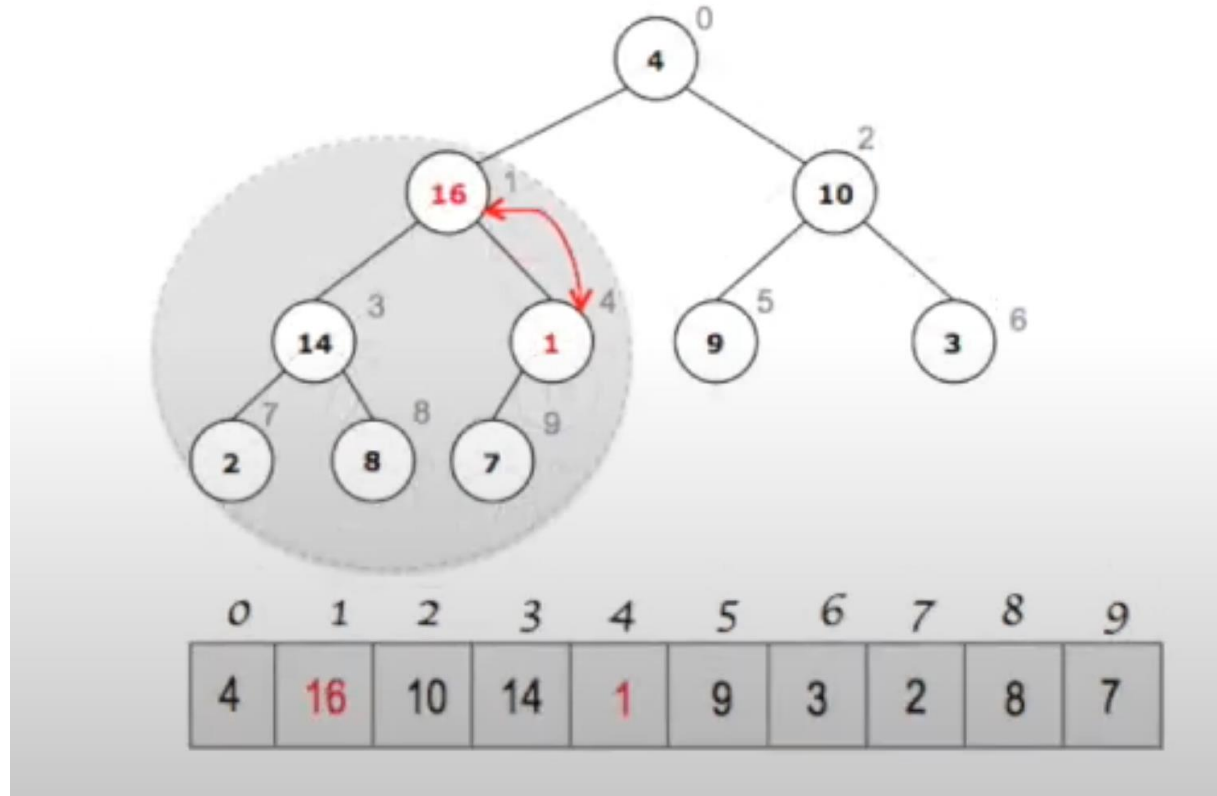
HeapSort

- Começamos com a raiz, entre os filhos qual o maior entre eles?



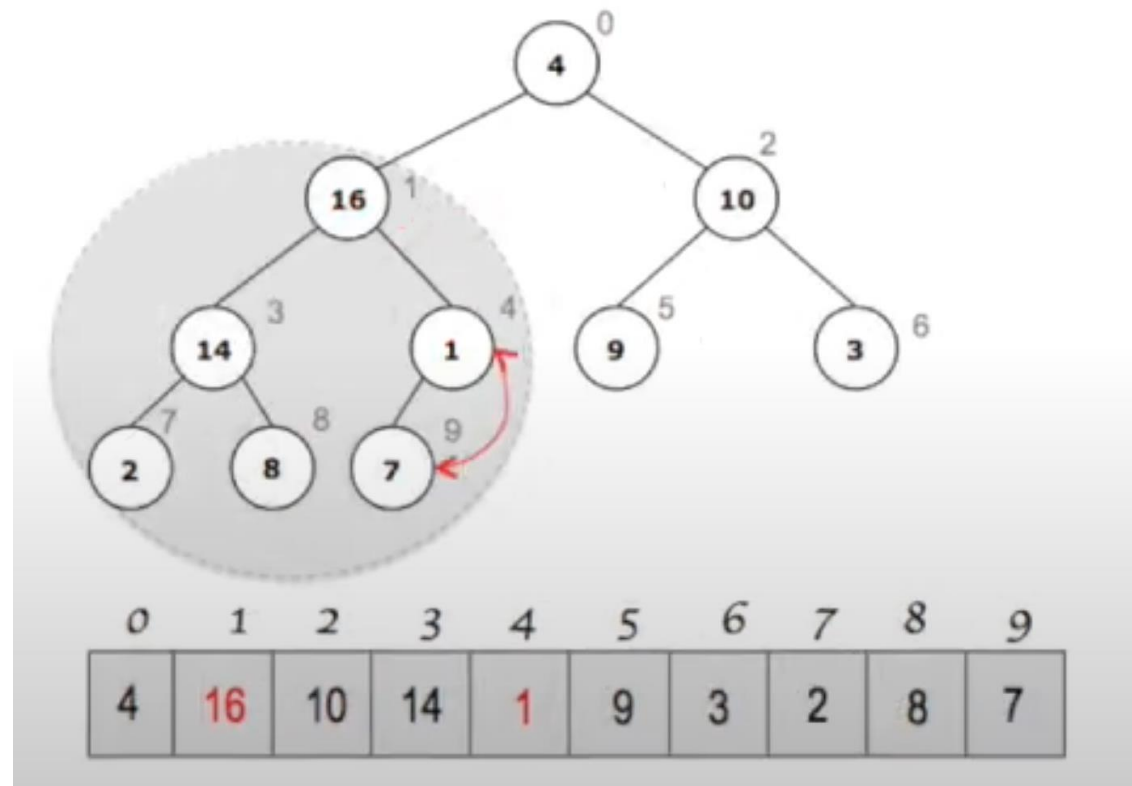
HeapSort

- No entanto, agora a sub-árvore da direita não é mais uma Heap



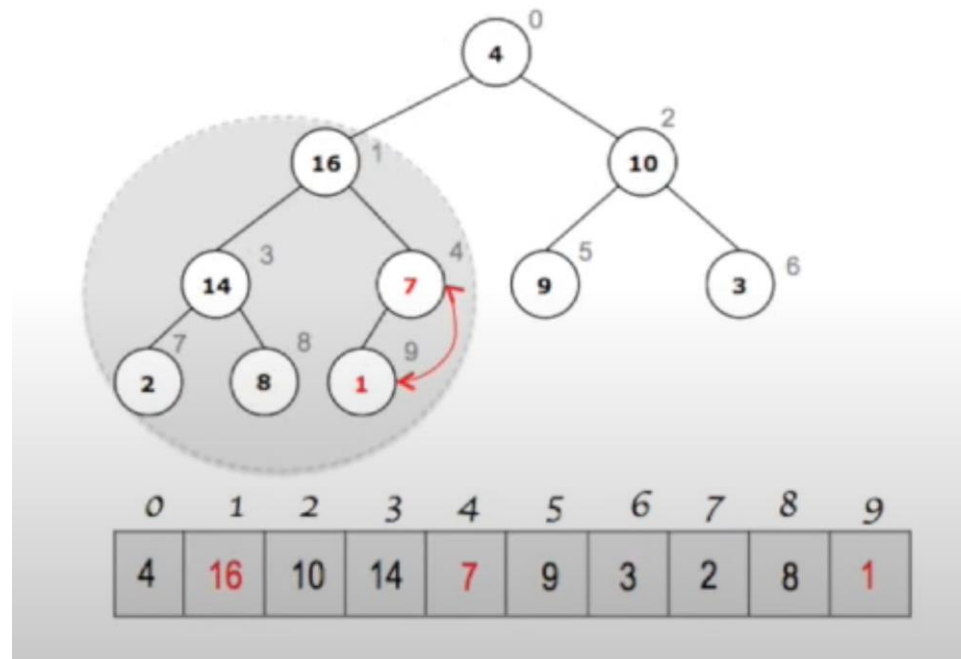
HeapSort

- Troca do 7 com o 1



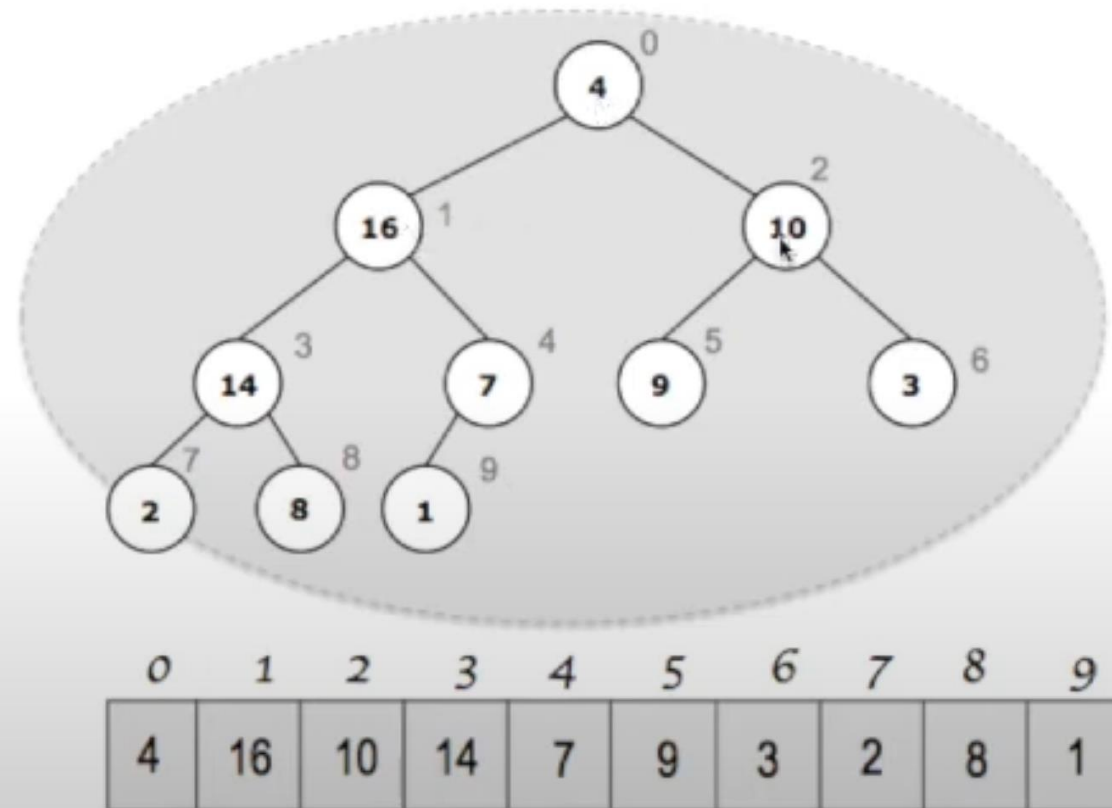
HeapSort

- Agora sim, toda essa sub-árvore é uma heap



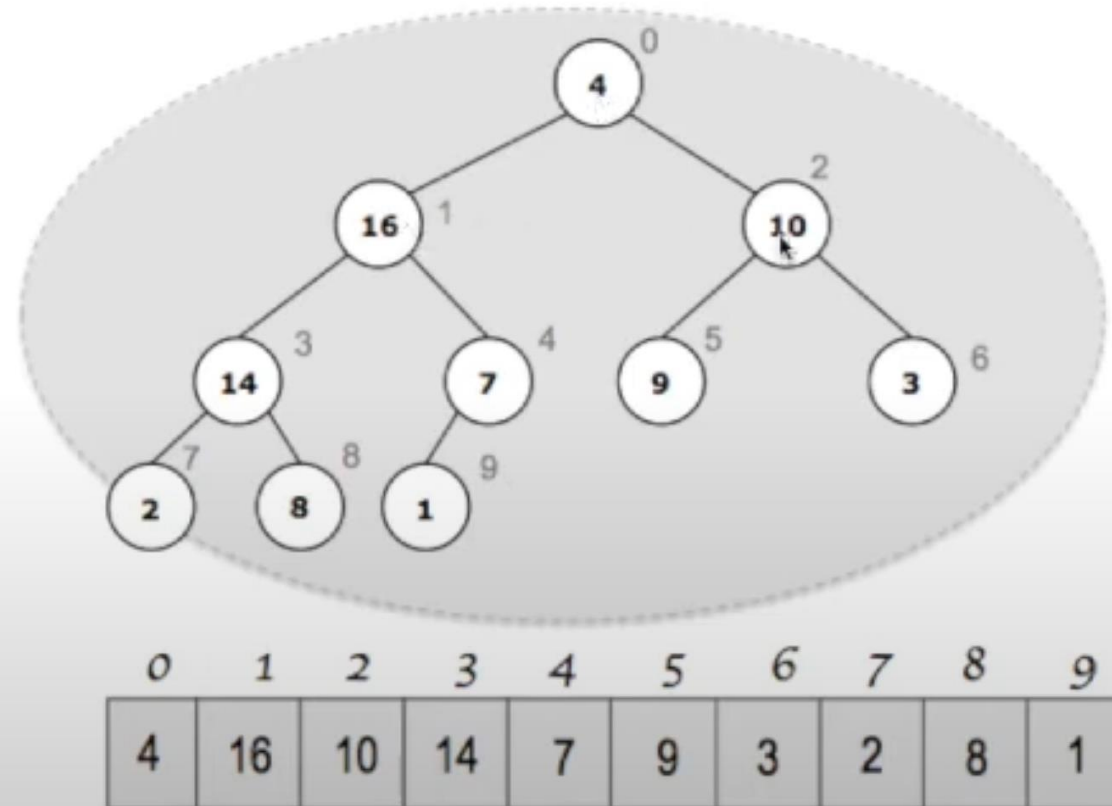
HeapSort

- Subimos de nível novamente



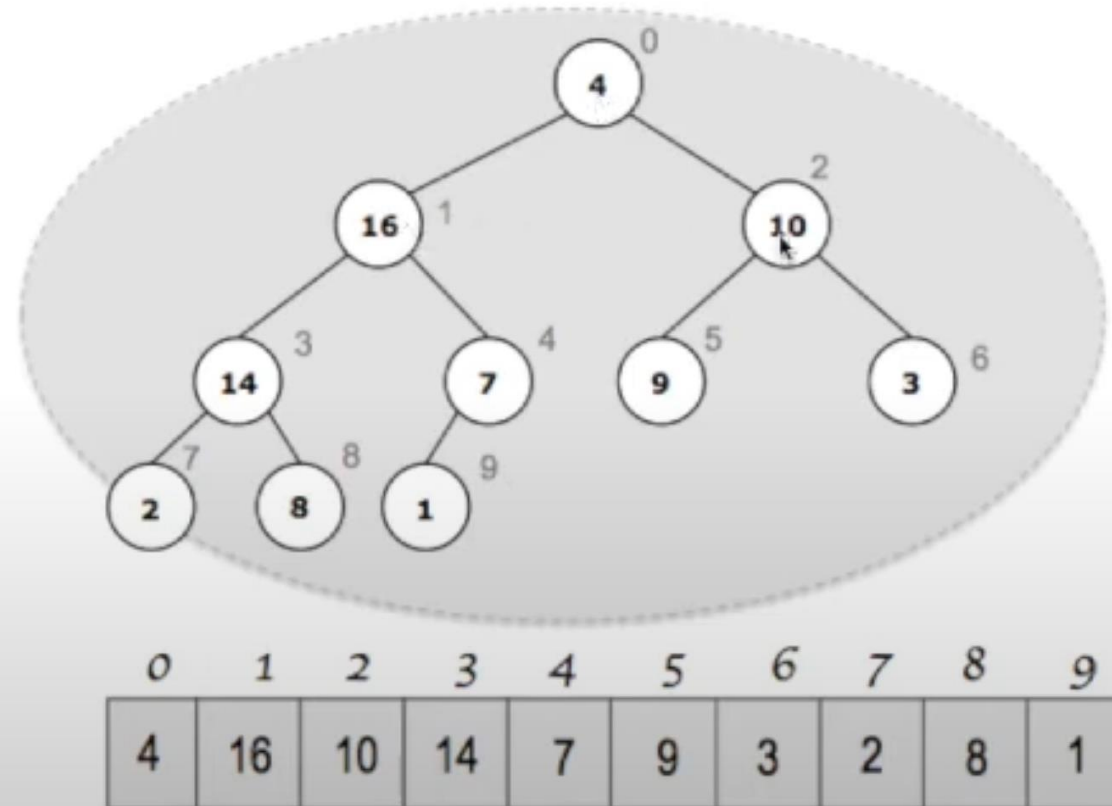
HeapSort

- Não é uma Heap



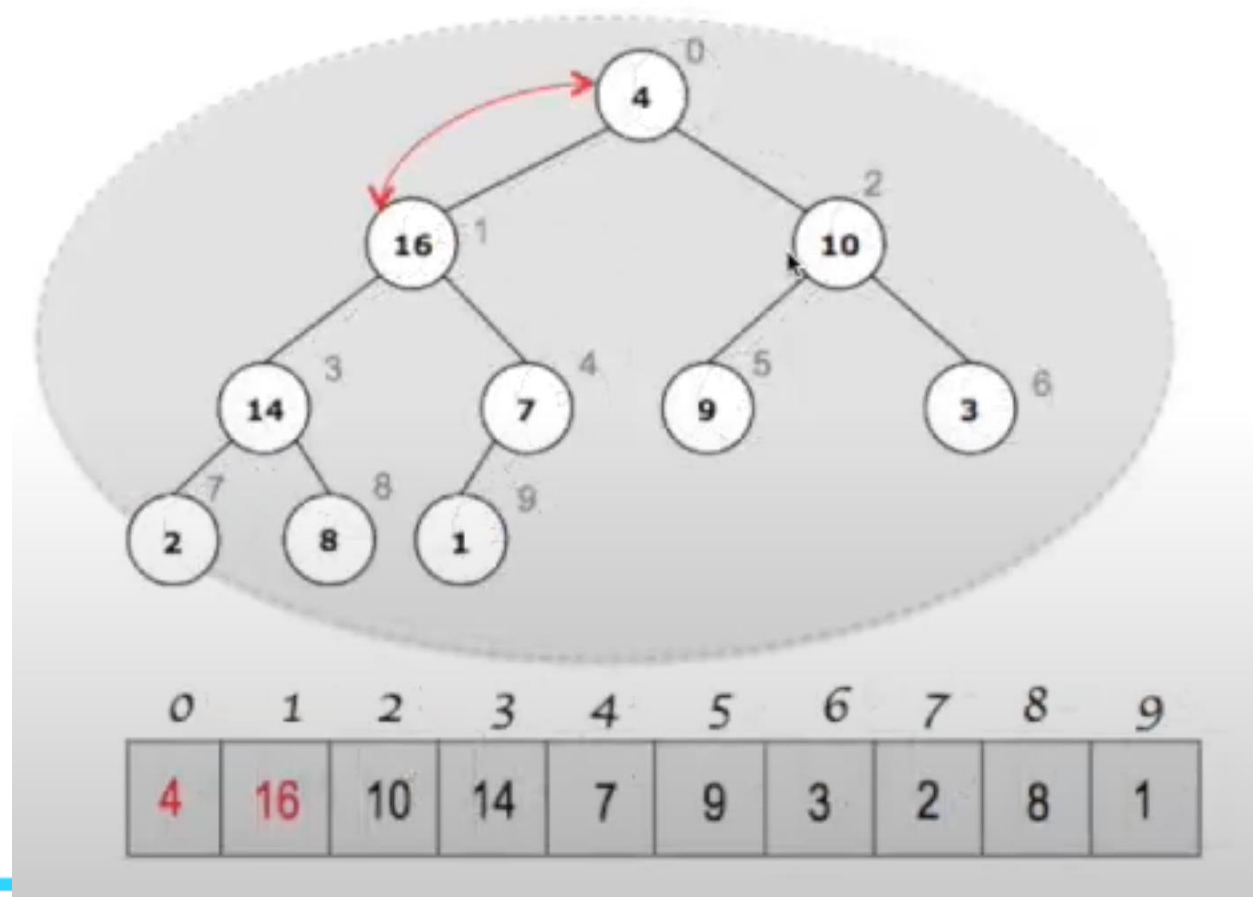
HeapSort

- Entre os filhos do nó raiz 4, qual o maior?



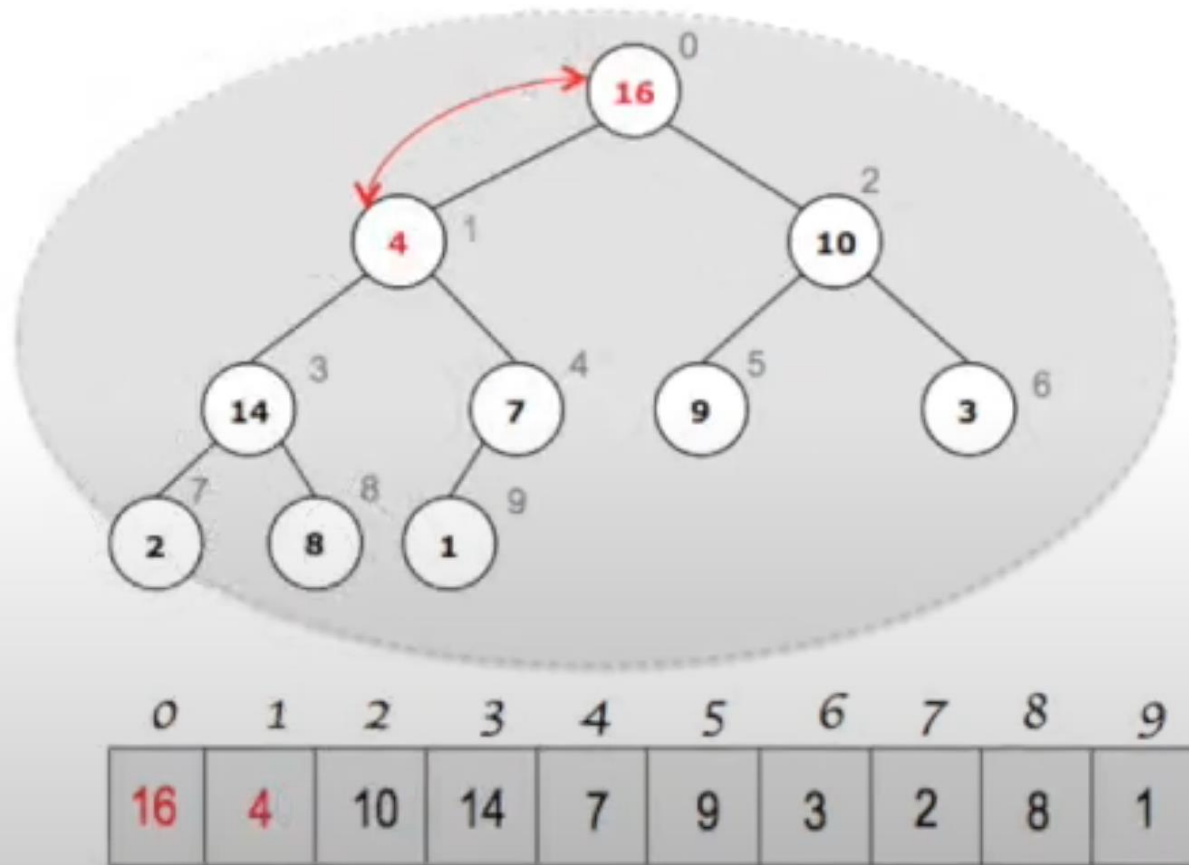
HeapSort

- Troca o 16 com o 4



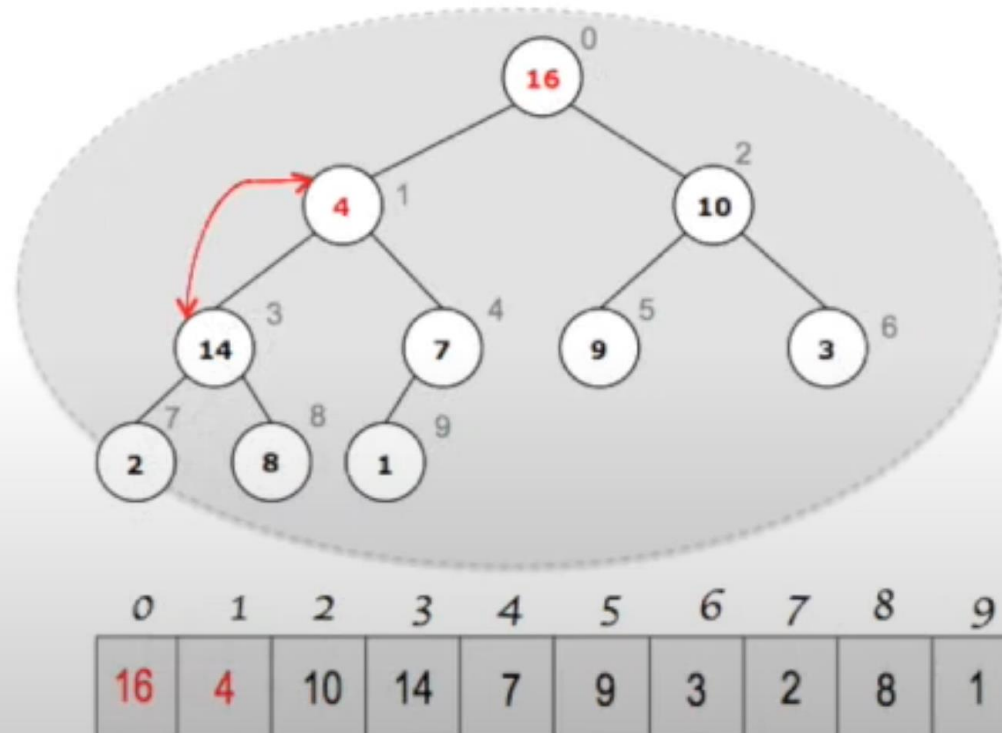
HeapSort

- Troca o 16 com o 4



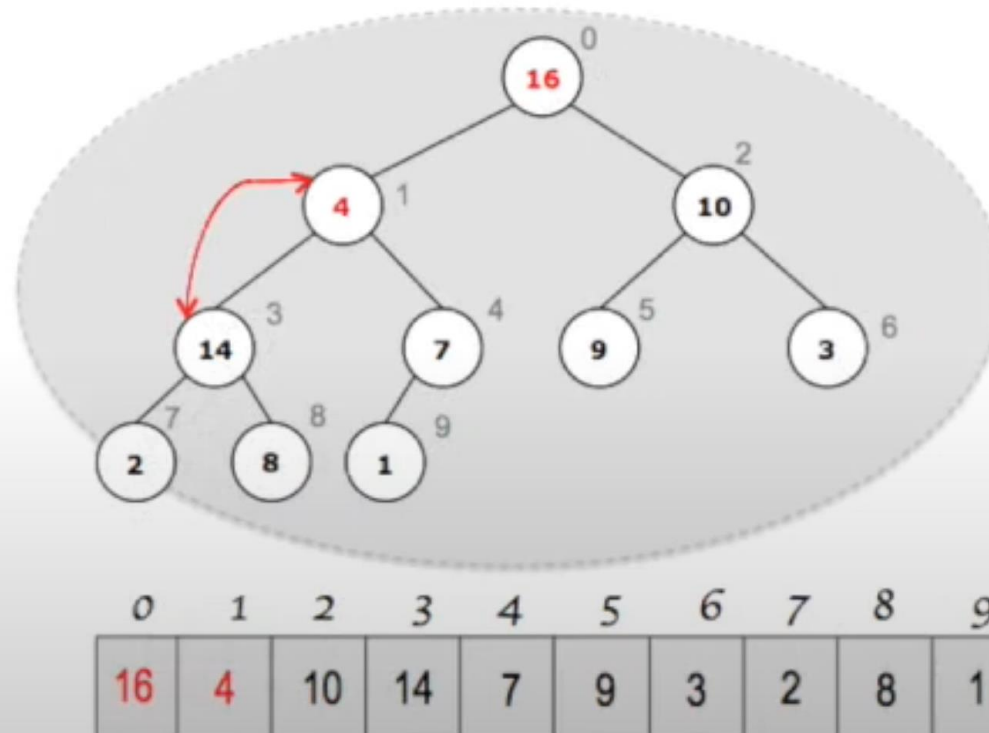
HeapSort

- No entanto, a sub-árvore que possui o nó raiz 4 deixou de ser uma Heap



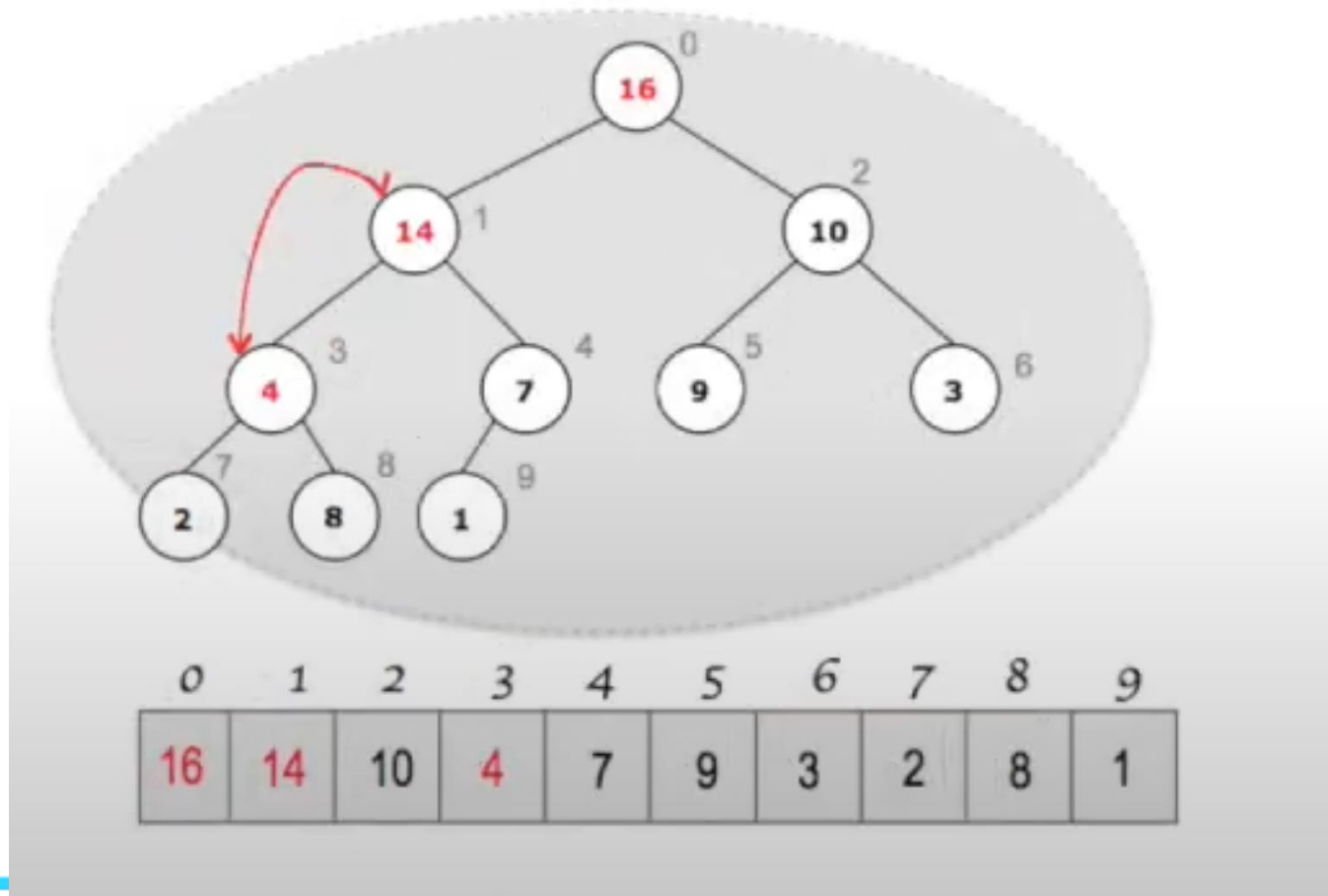
HeapSort

- Neste caso, troca o 4 com o filho de maior valor, ou seja, o 14



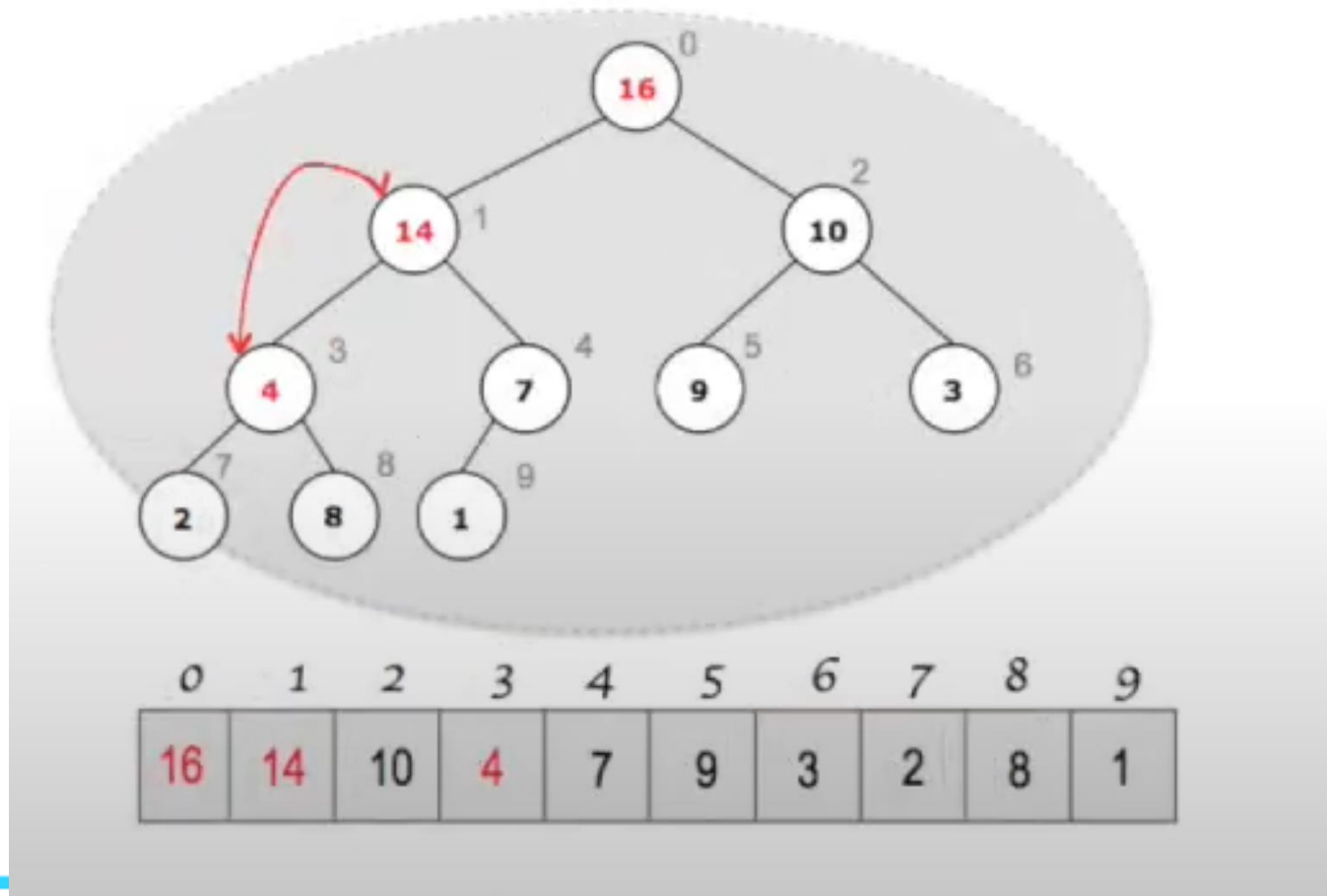
HeapSort

- Agora, a sub-árvore com o valor 4 deixou de ser uma Heap



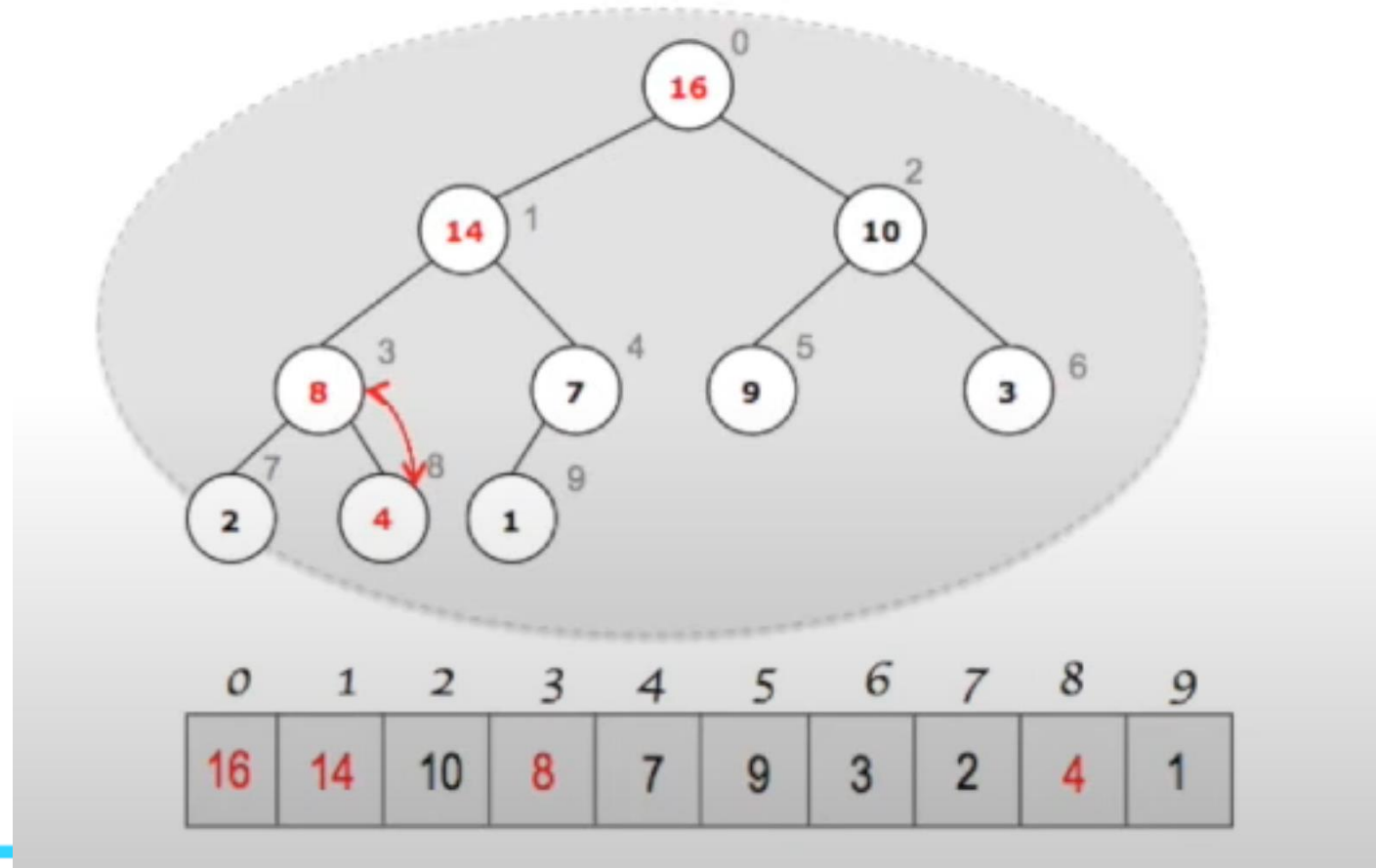
HeapSort

- Precisamos realizar uma troca com do quadro com o seu filho de maior valor



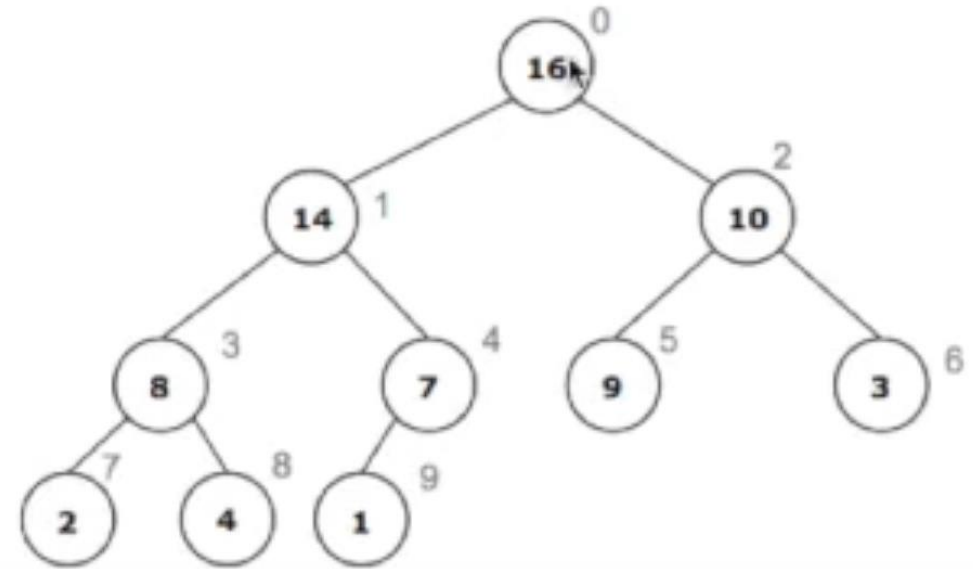
HeapSort

- Troca o 4 com o 8 e agora sim temos uma Heap.
- **Toda a árvore ficou Heap!!!**



HeapSort

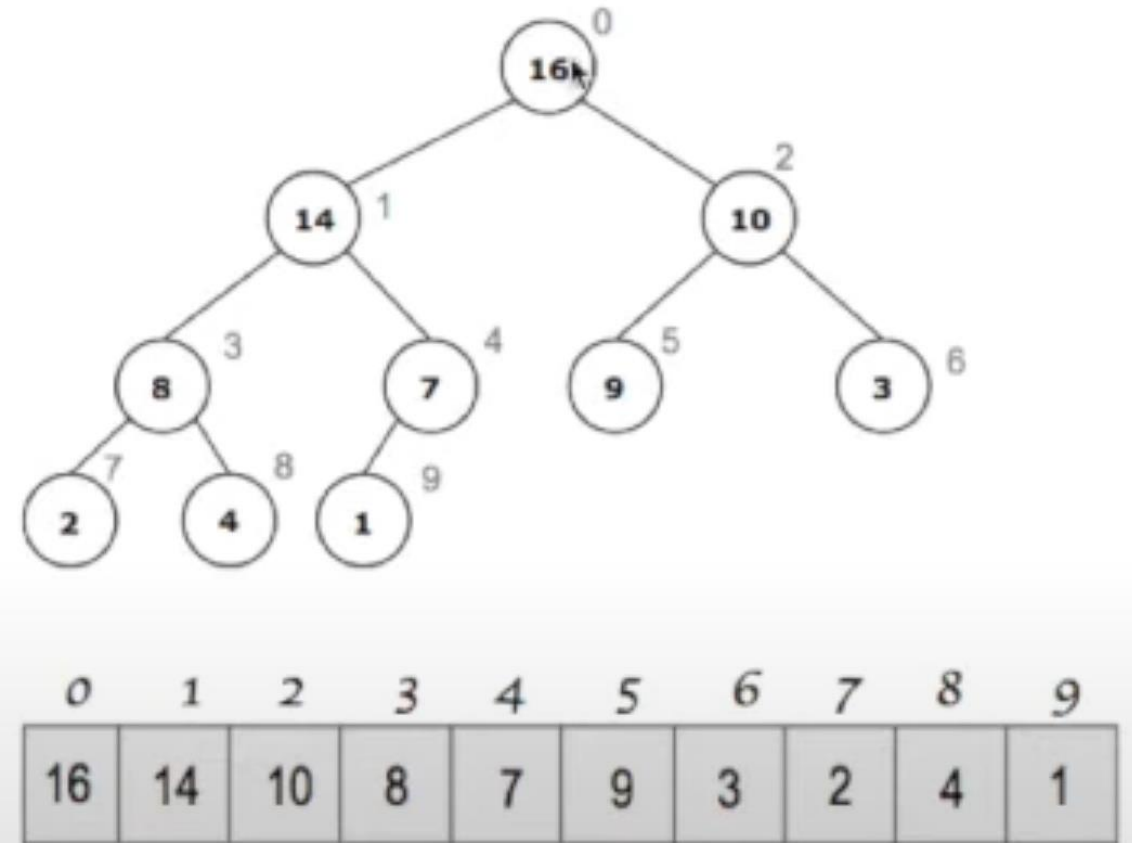
- **Fase 2:** Seleção dos elementos na ordem desejada



0	1	2	3	4	5	6	7	8	9
16	14	10	8	7	9	3	2	4	1

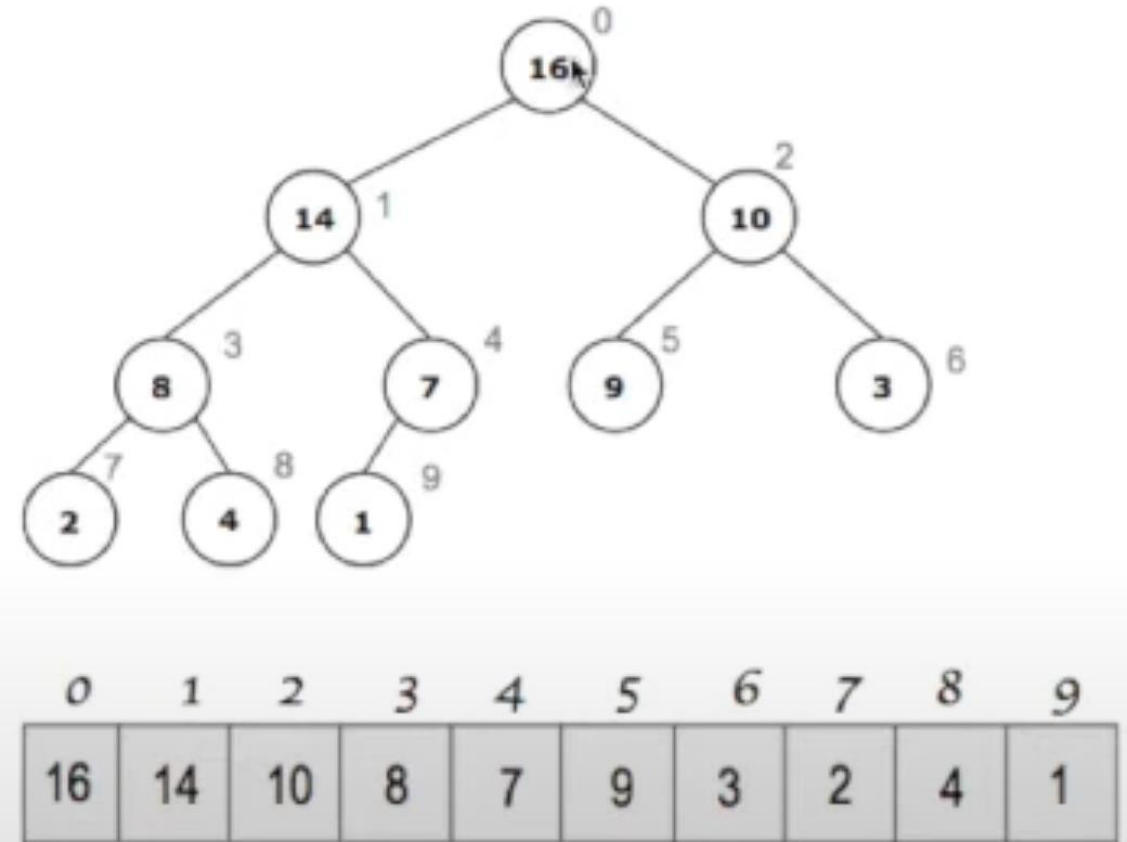
HeapSort

- **Fase 2:** Seleção dos elementos na ordem desejada
 - Se a chave que está na raiz é a maior de todas, então sua posição definitiva correta, na ordem crescente, é a última posição no vetor;

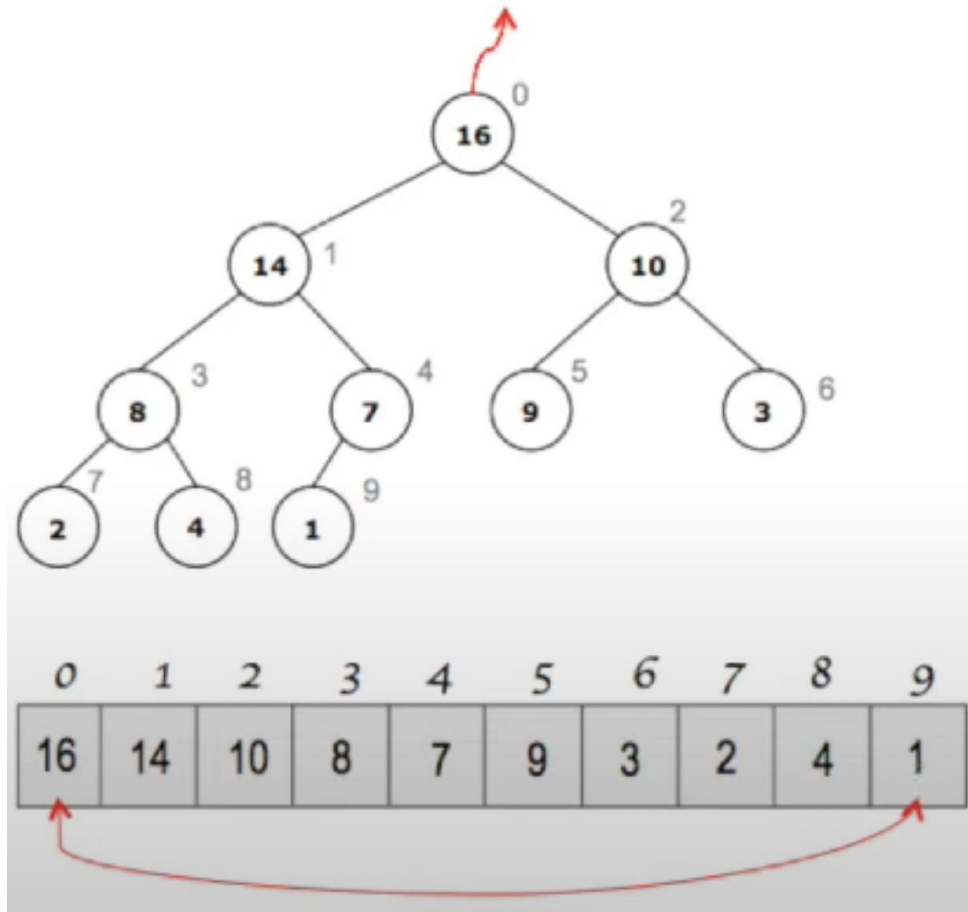


HeapSort

- **Fase 2:** Seleção dos elementos na ordem desejada
 - Se a chave que está na raiz é a maior de todas, então sua posição definitiva correta, na ordem crescente, é a última posição no vetor;
 - Então, esta maior chave é colocada na última posição do vetor, por troca, com a chave que ocupa aquela posição.

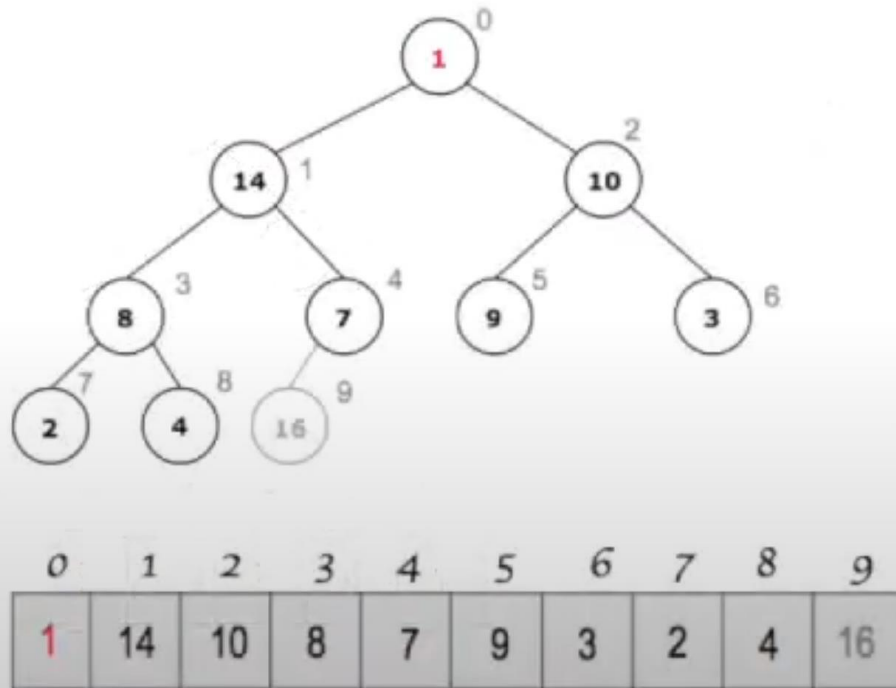


HeapSort



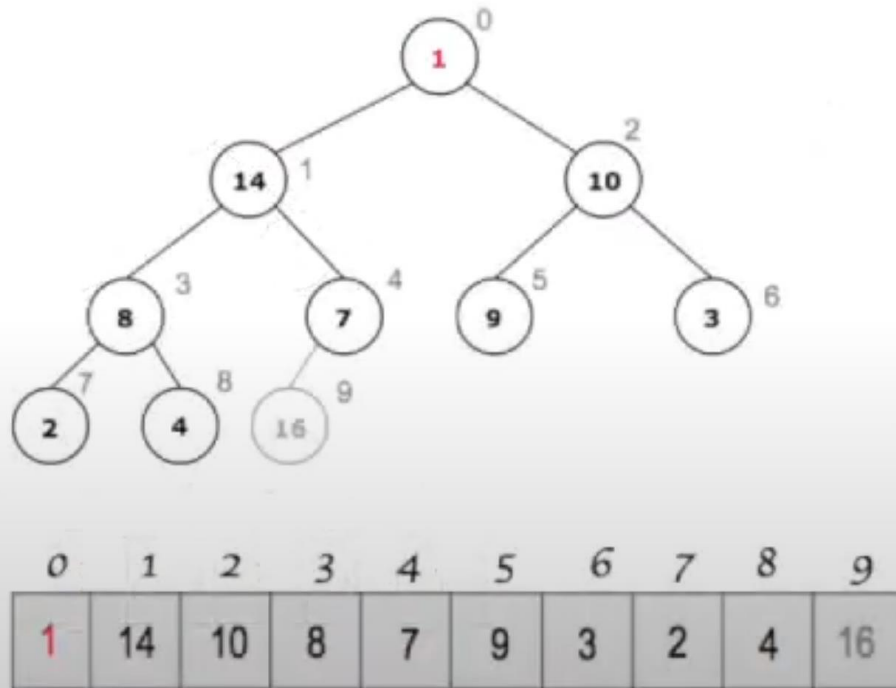
- Seleção da maior chave
 - Troca de posição com a chave na última posição da array

HeapSort



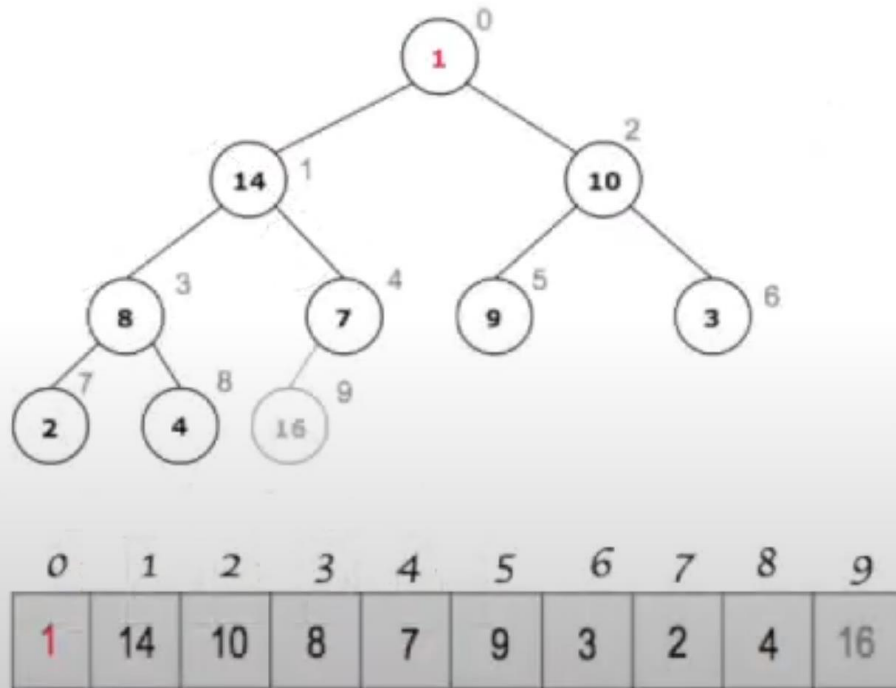
- A partir deste momento, teremos 2 setores na array.

HeapSort



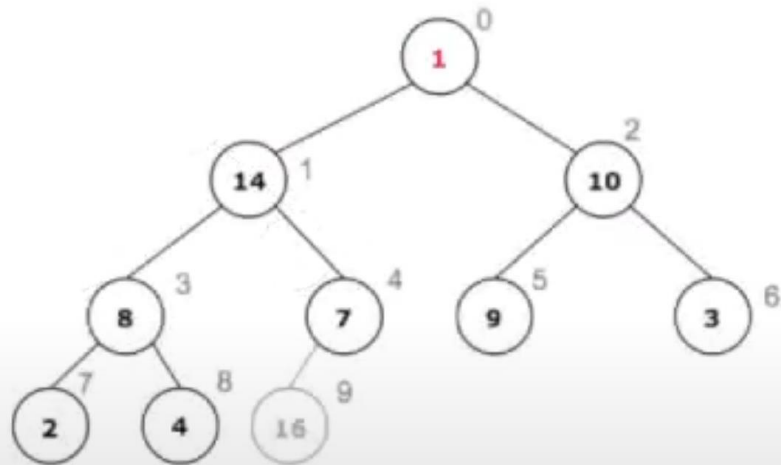
- A partir deste momento, teremos 2 setores na array.
 - A parte final será a parte ordenada
 - A parte inicial será a parte ainda não ordenada

HeapSort



- Na árvore, isso representa:
 - A parte que esta em cima (ordenada)
- O restante da array será utilizada para sempre ser transformada em um heap para pegar o maior valor e deslocar para o fim da array

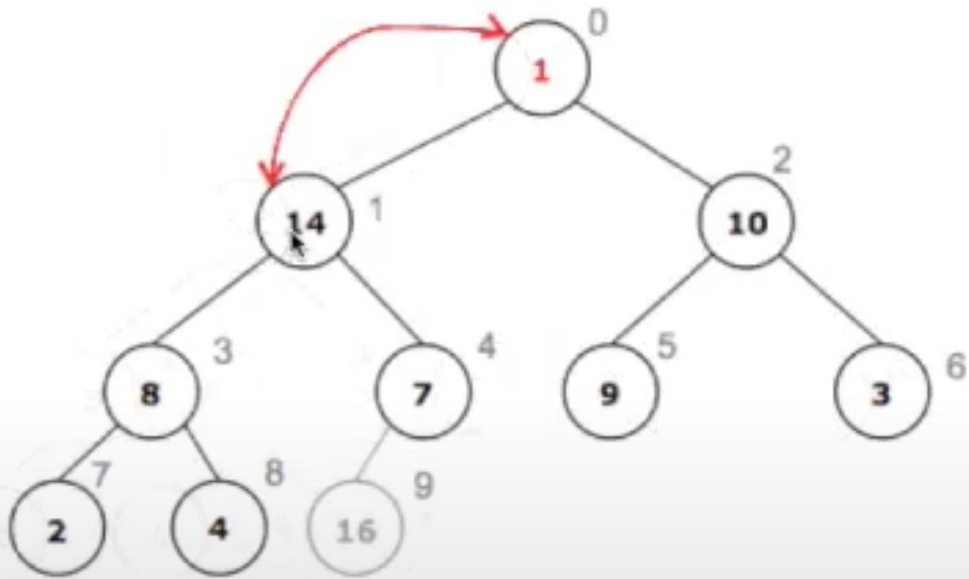
HeapSort



0	1	2	3	4	5	6	7	8	9
1	14	10	8	7	9	3	2	4	16

- **Portanto,** precisa novamente transformar em uma heap começando sempre pela raiz

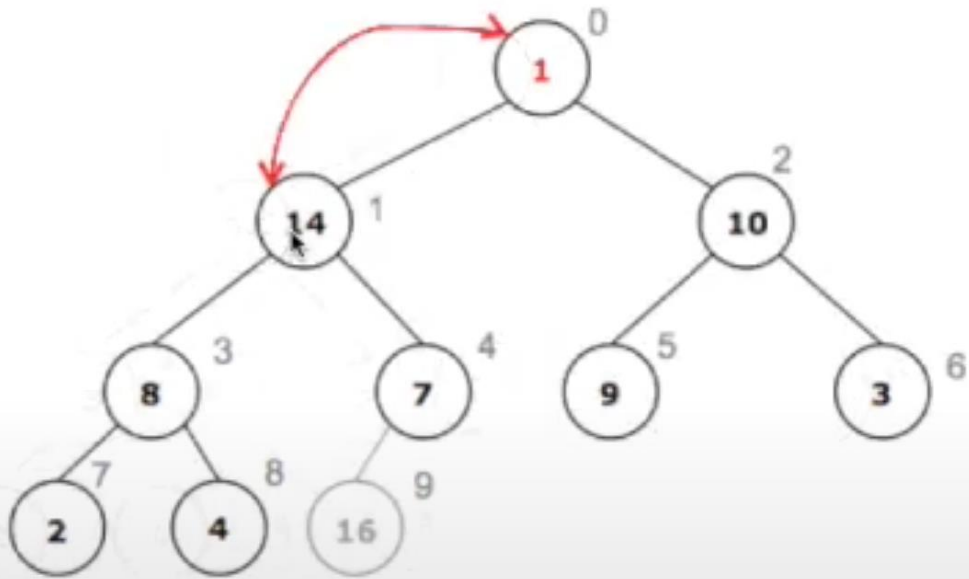
HeapSort



0	1	2	3	4	5	6	7	8	9
1	14	10	8	7	9	3	2	4	16

- Pega o maior filho do nó 1 que o 14 e troca de lugar

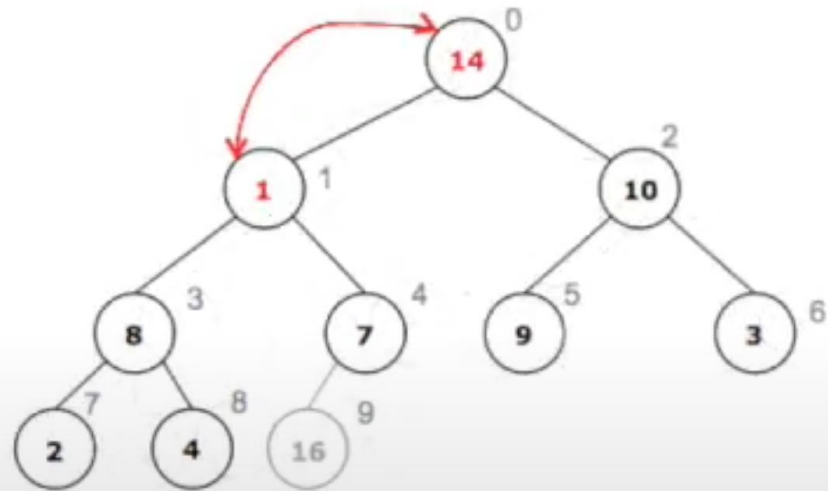
HeapSort



0	1	2	3	4	5	6	7	8	9
1	14	10	8	7	9	3	2	4	16

- Pega o maior filho do nó 1 que o 14 e troca de lugar
- Processo chamado de heapfy (neste caso, heapfy-down)

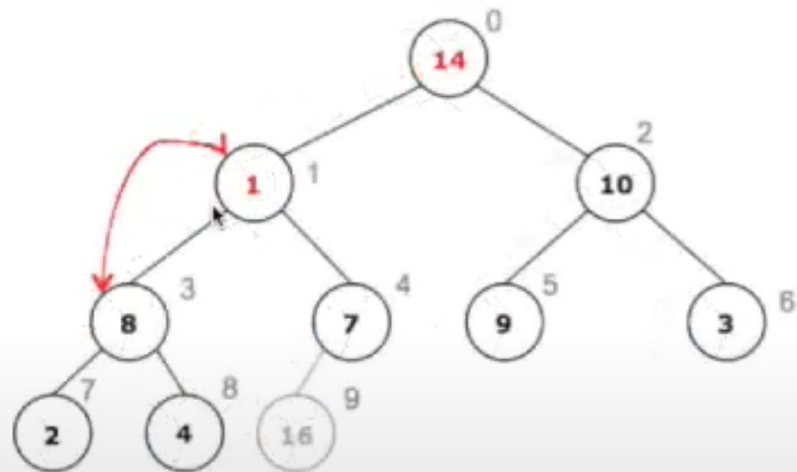
HeapSort



0	1	2	3	4	5	6	7	8	9
14	1	10	8	7	9	3	2	4	16

- Ao fazer isso, o restante da sub-árvore deixou de ser heap

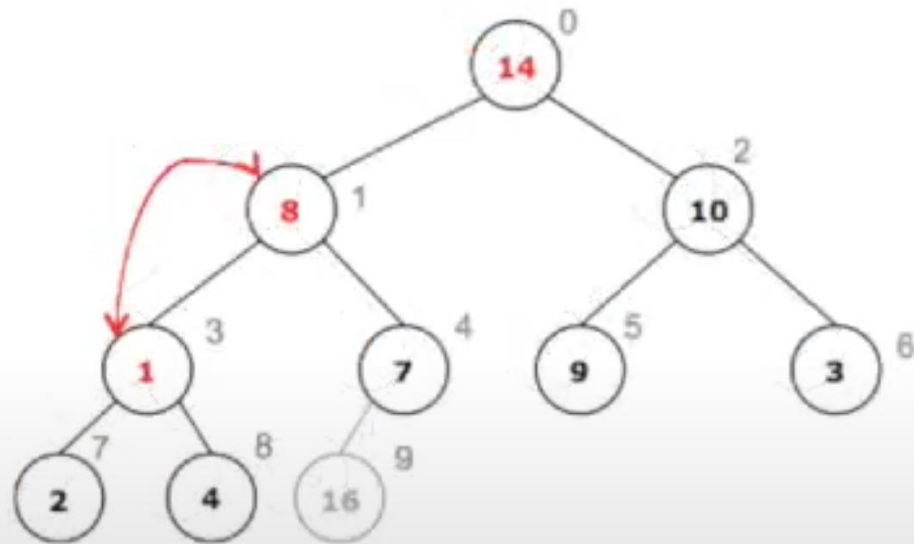
HeapSort



0	1	2	3	4	5	6	7	8	9
14	1	10	8	7	9	3	2	4	16

- Verifica o maior filho do 1, que é o 8 e troca de lugar.

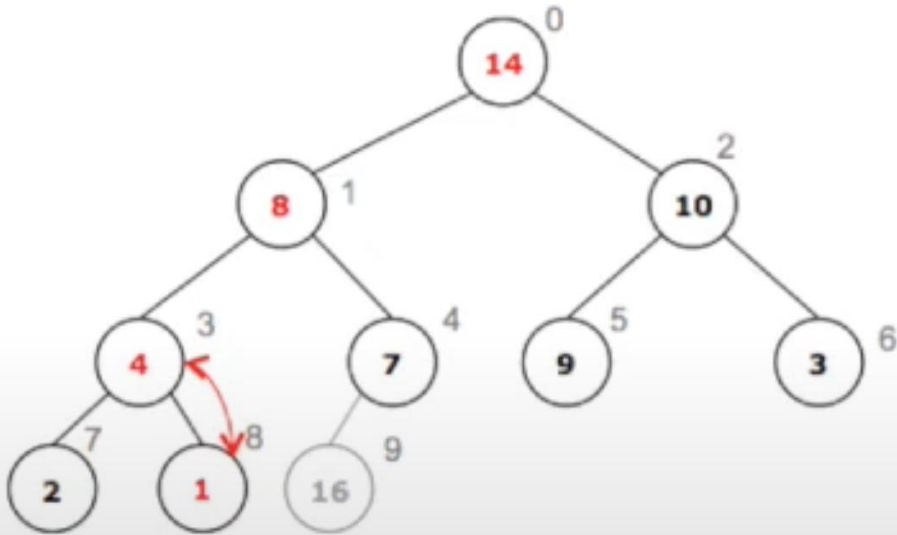
HeapSort



0	1	2	3	4	5	6	7	8	9
14	8	10	1	7	9	3	2	4	16

- Agora, deve ser feito com o restante da sub-árvore de 1

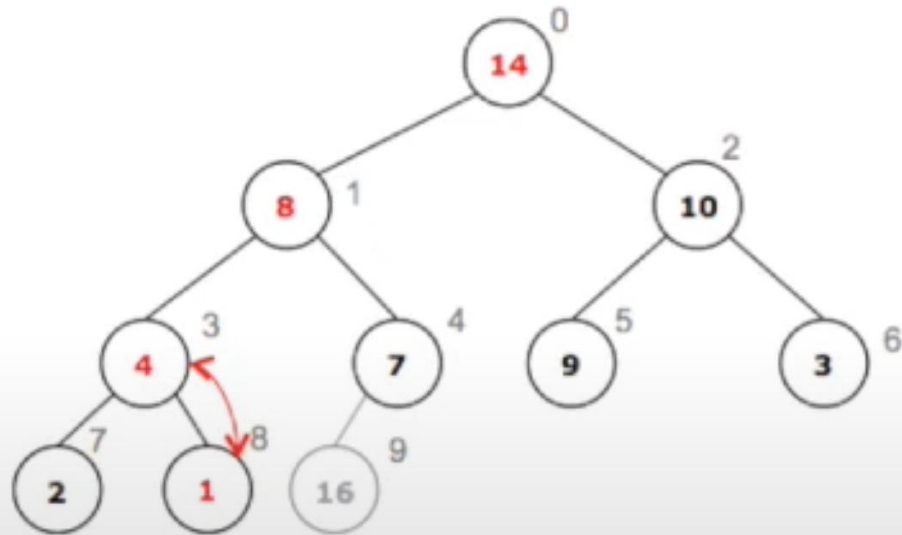
HeapSort



0	1	2	3	4	5	6	7	8	9
14	8	10	4	7	9	3	2	1	16

- Troca 1 com 4 e temos o seguinte resultado.

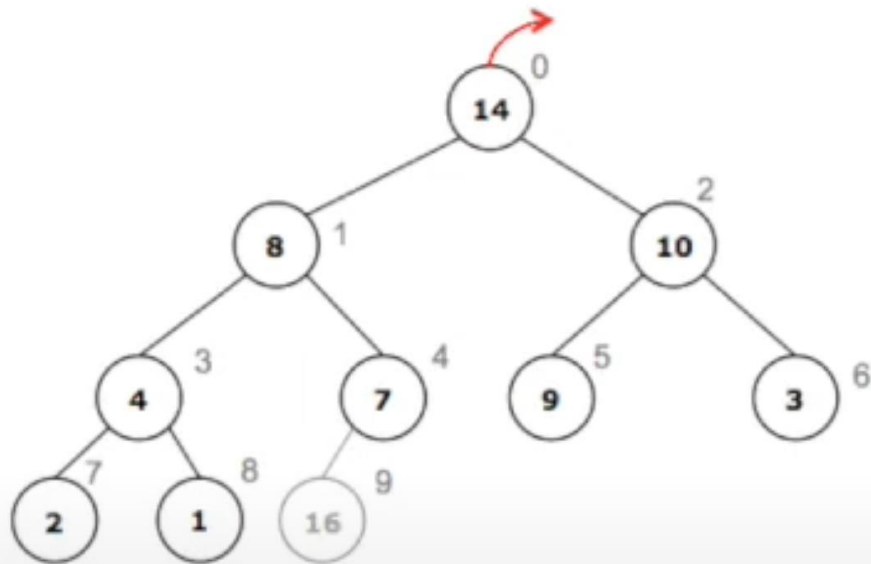
HeapSort



0	1	2	3	4	5	6	7	8	9
14	8	10	4	7	9	3	2	1	16

- Troca 1 com 4 e temos o seguinte resultado.
- **Temos novamente uma árvore Heap!**

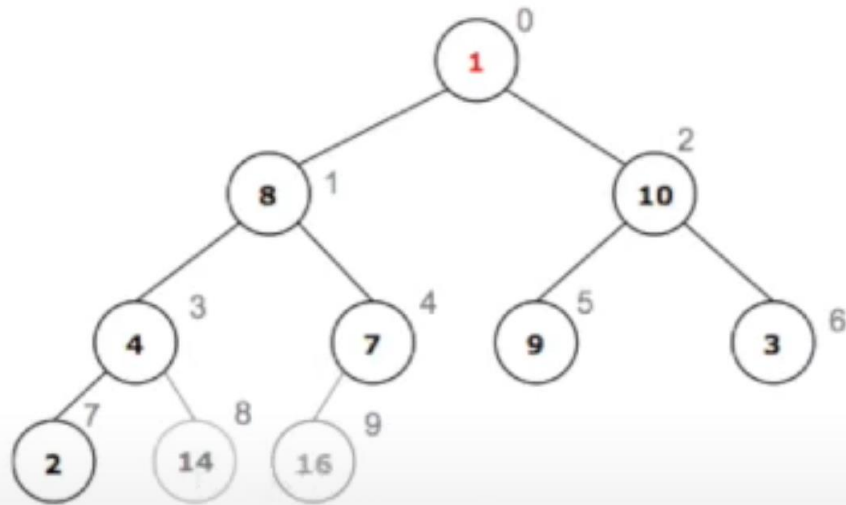
HeapSort



0	1	2	3	4	5	6	7	8	9
14	8	10	4	7	9	3	2	1	16

- Novamente, iremos trocar o elemento raiz a última posição do array (inserir o maior elemento no setor de elementos não ordenados)

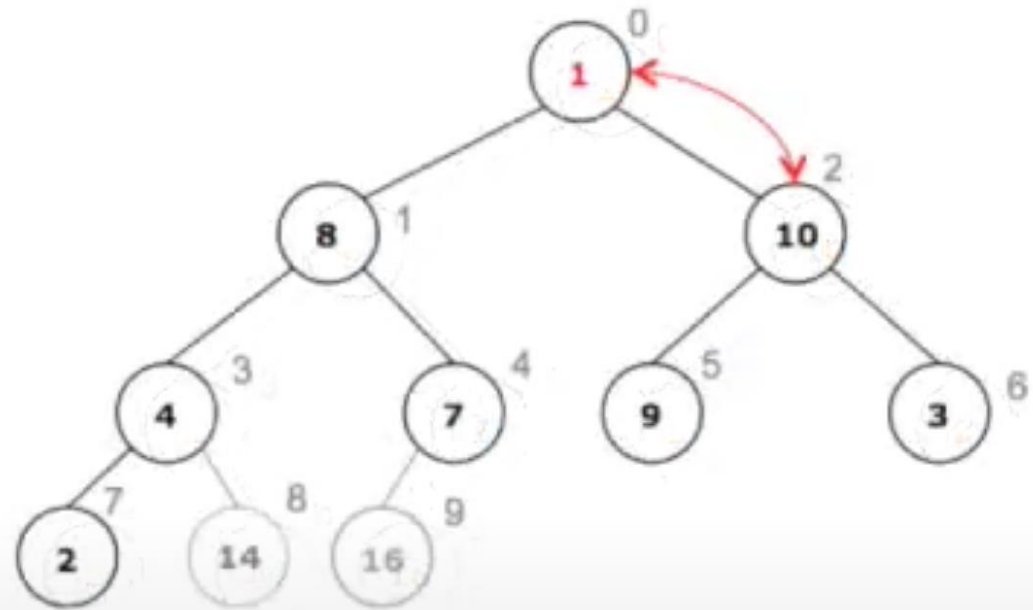
HeapSort



0	1	2	3	4	5	6	7	8	9
1	8	10	4	7	9	3	2	14	16

- Necessário transformar a árvore novamente em uma Heap

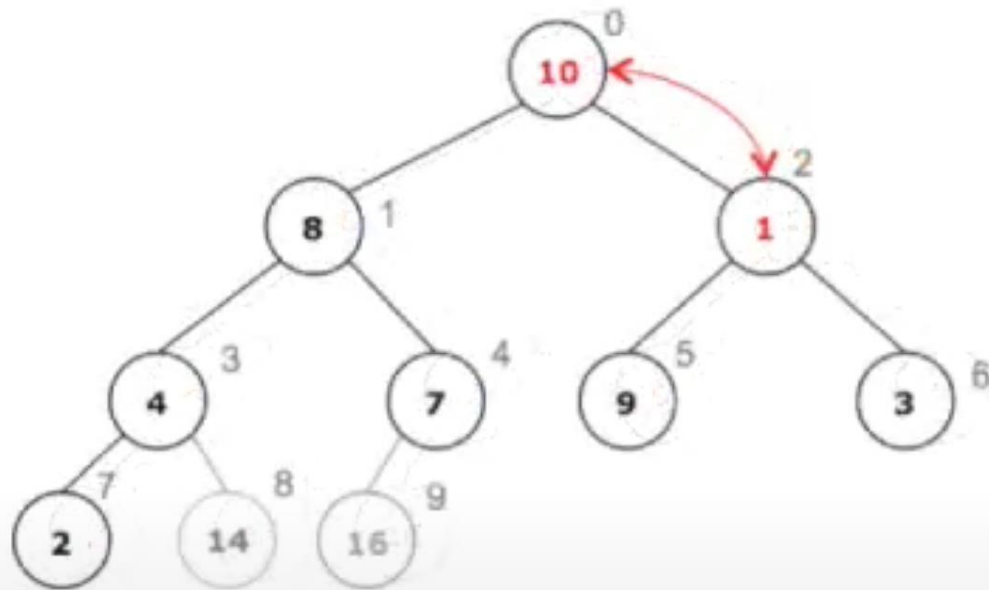
HeapSort



- Troca o maior o 1 com o maior filho

0	1	2	3	4	5	6	7	8	9
1	8	10	4	7	9	3	2	14	16

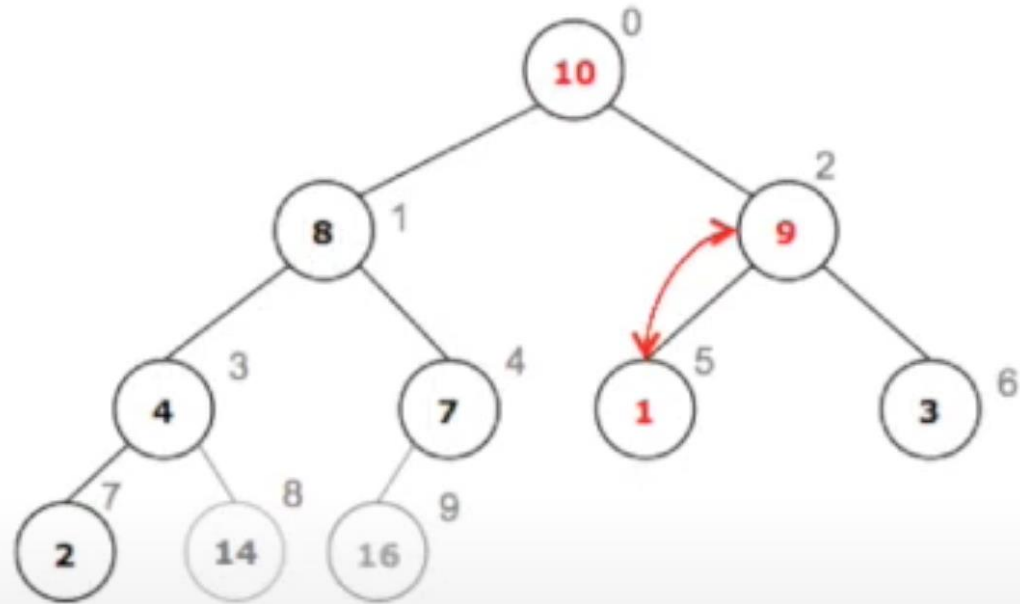
HeapSort



0	1	2	3	4	5	6	7	8	9
10	8	1	4	7	9	3	2	14	16

- Mesmo processo deve ser feito na sub-árvore da direita

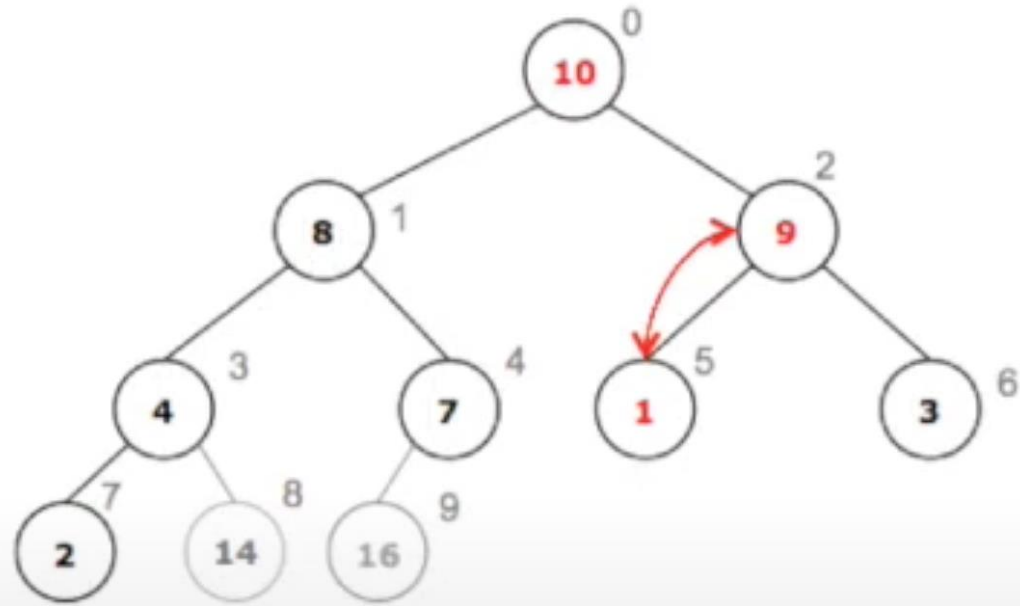
HeapSort



0	1	2	3	4	5	6	7	8	9
10	8	9	4	7	1	3	2	14	16

- Troca o 1 com o maior filho

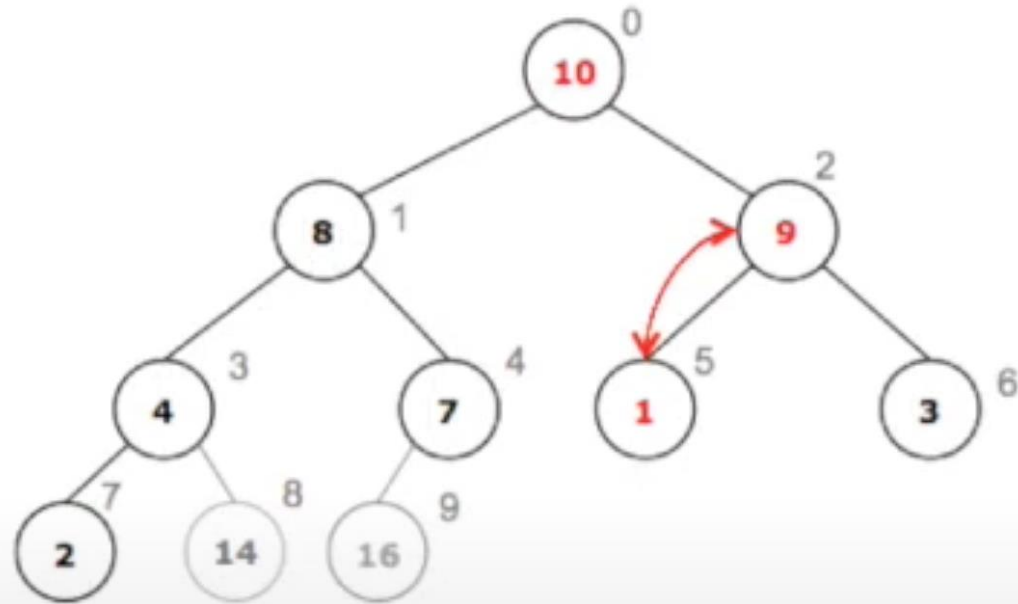
HeapSort



0	1	2	3	4	5	6	7	8	9
10	8	9	4	7	1	3	2	14	16

- Temos novamente um heap!

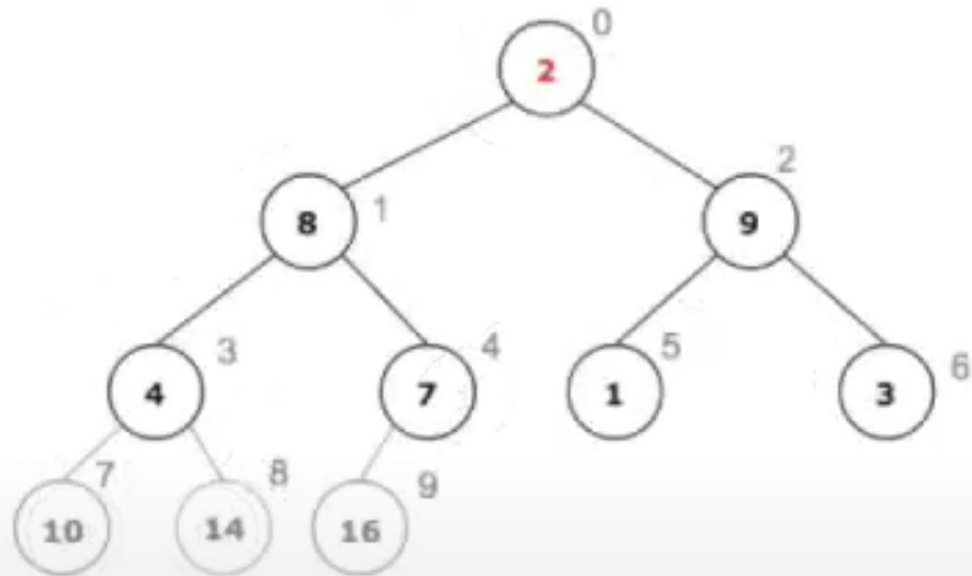
HeapSort



0	1	2	3	4	5	6	7	8	9
10	8	9	4	7	1	3	2	14	16

- **Pegamos** novamente o elemento de **maior valor**, ou seja, a **raiz** e **jogamos para a última posição** do setor do vetor não ordenado.

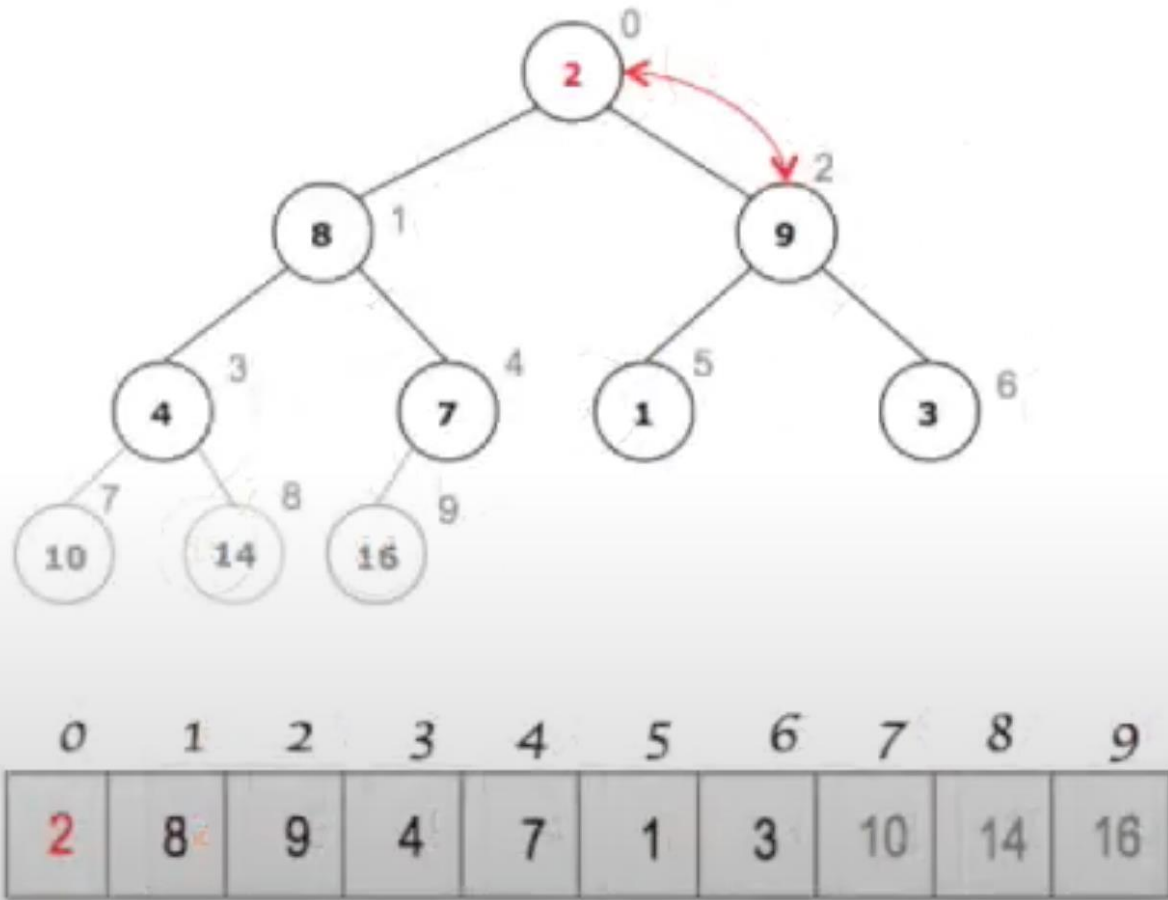
HeapSort



0	1	2	3	4	5	6	7	8	9
2	8	9	4	7	1	3	10	14	16

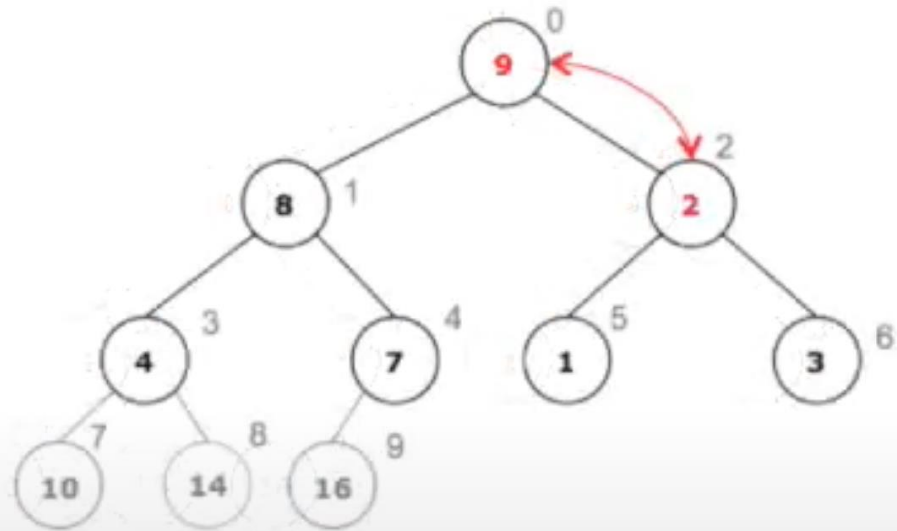
- Agora, precisamos novamente transformar a árvore em um heap

HeapSort



- Começamos, novamente, pela a raiz e trocamos com o maior filho.

HeapSort

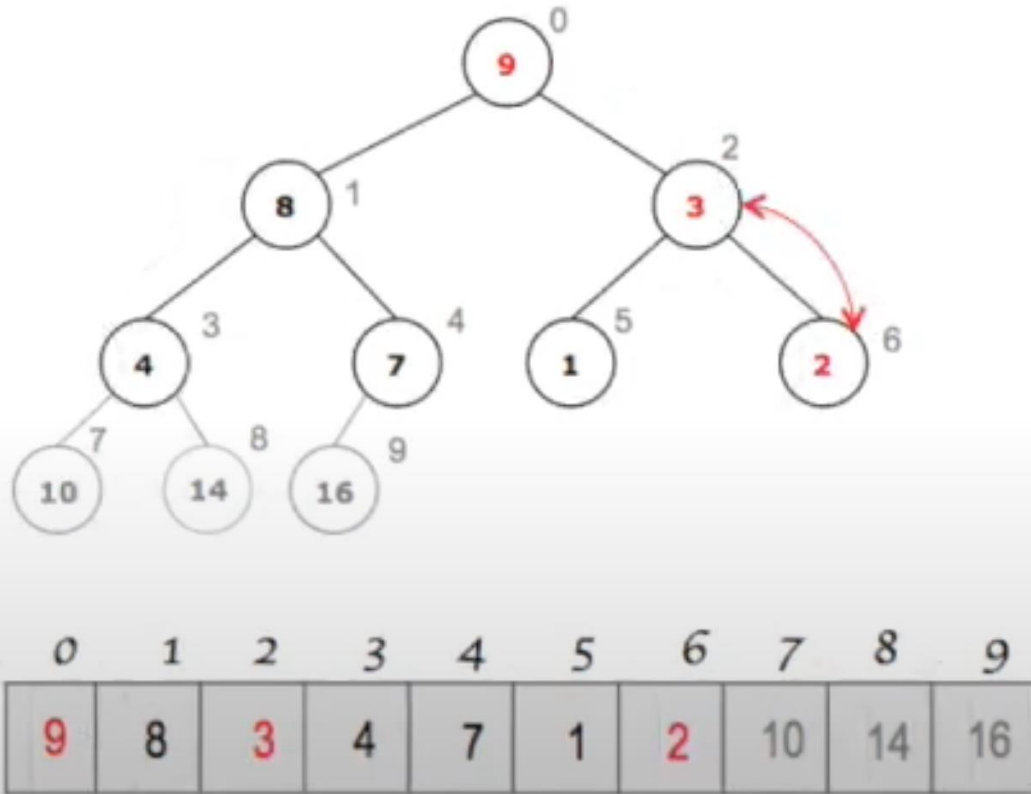


0	1	2	3	4	5	6	7	8	9
9	8	2	4	7	1	3	10	14	16

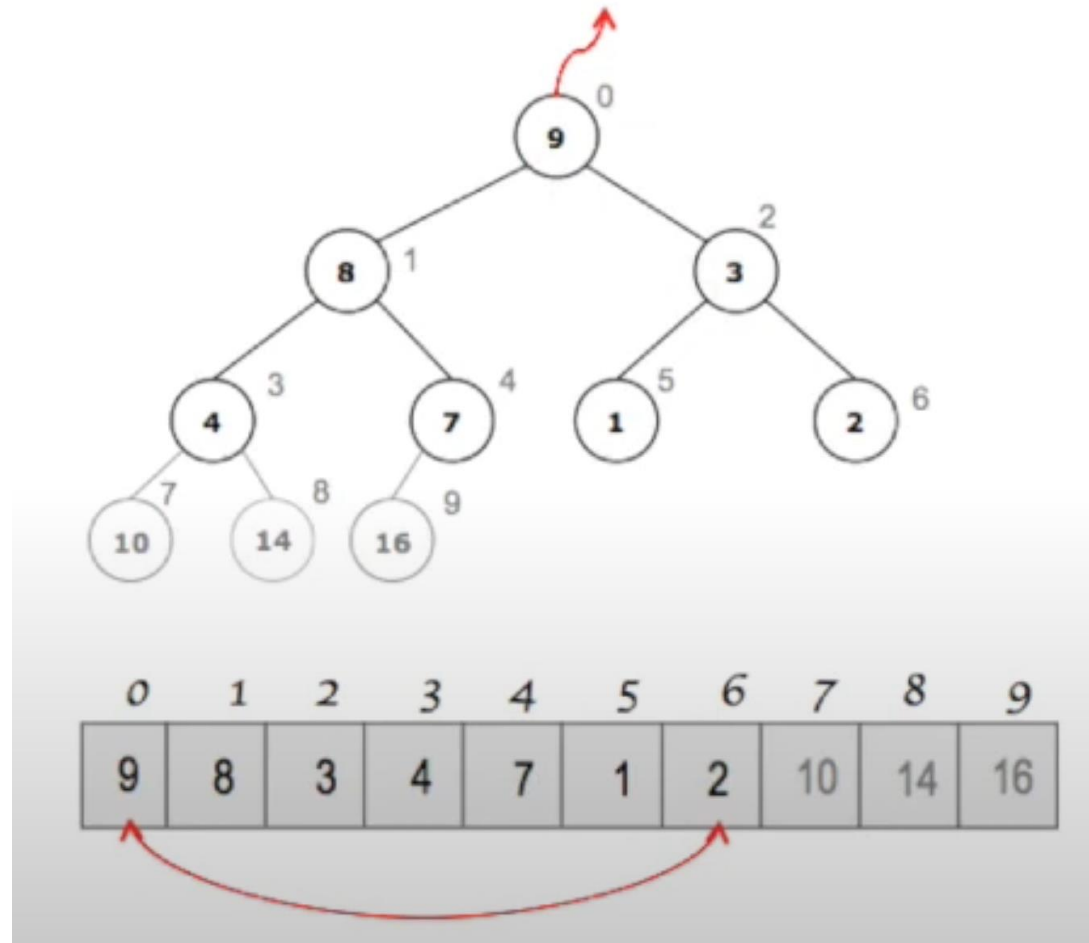
- A sub-arvore da direita não é mais uma Heap

HeapSort

- Troca o 2 com o 3

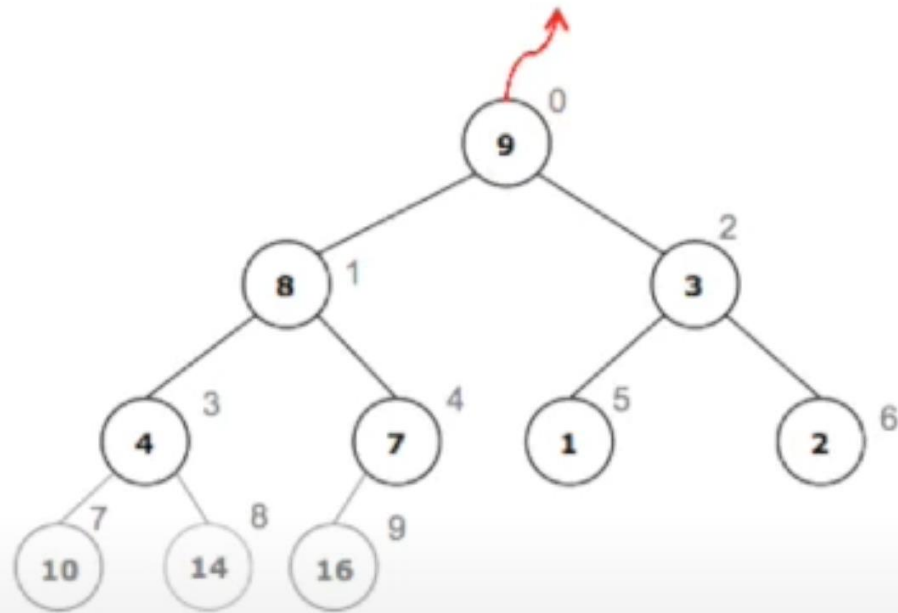


HeapSort



- Temos, novamente, uma Heap máxima

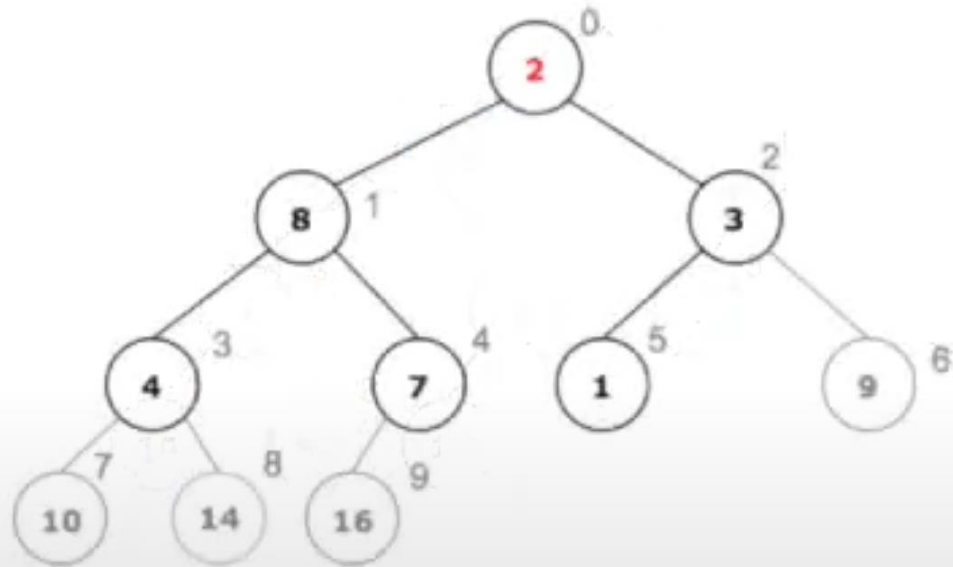
HeapSort



0	1	2	3	4	5	6	7	8	9
9	8	3	4	7	1	2	10	14	16

- Colocamos a raiz que o maior valor na última posição do vetor de não ordenados.

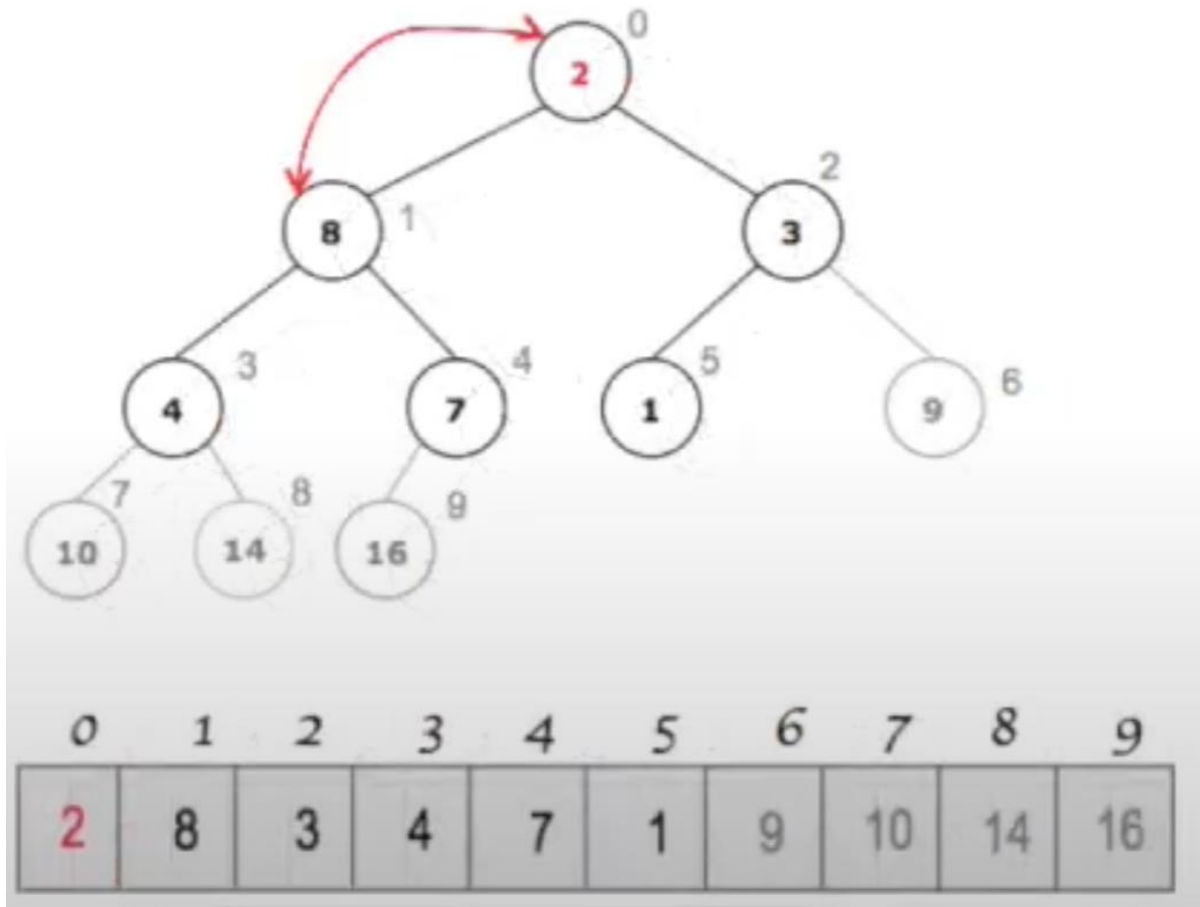
HeapSort



0	1	2	3	4	5	6	7	8	9
2	8	3	4	7	1	9	10	14	16

- Novamente, temos que transformar a árvore em um Heap máximo

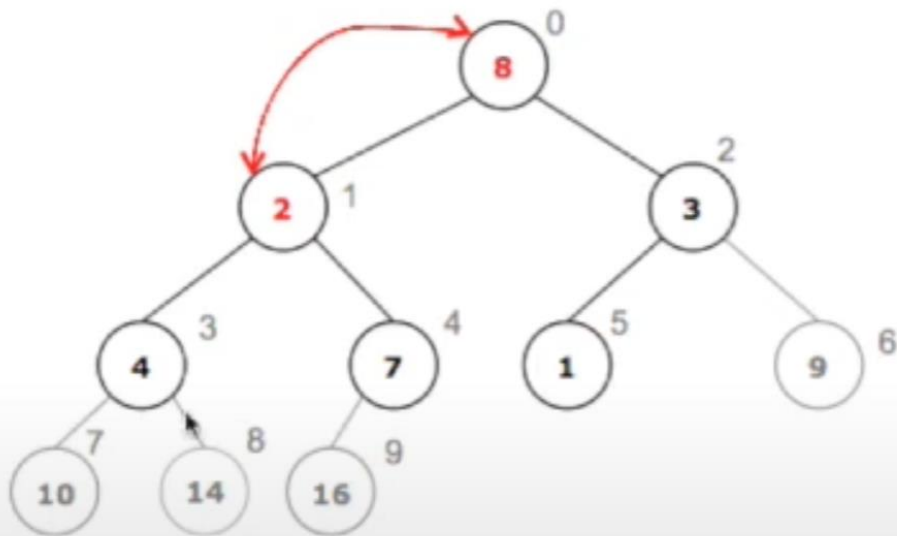
HeapSort



- Maior valor entre os filhos do nó raiz?

HeapSort

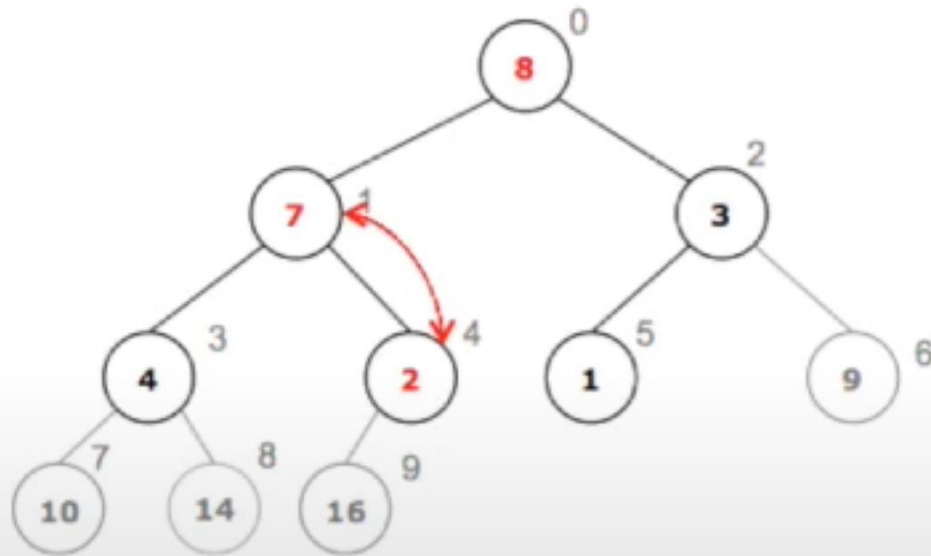
- Maior valor entre os filhos do nó 2?



0	1	2	3	4	5	6	7	8	9
8	2	3	4	7	1	9	10	14	16

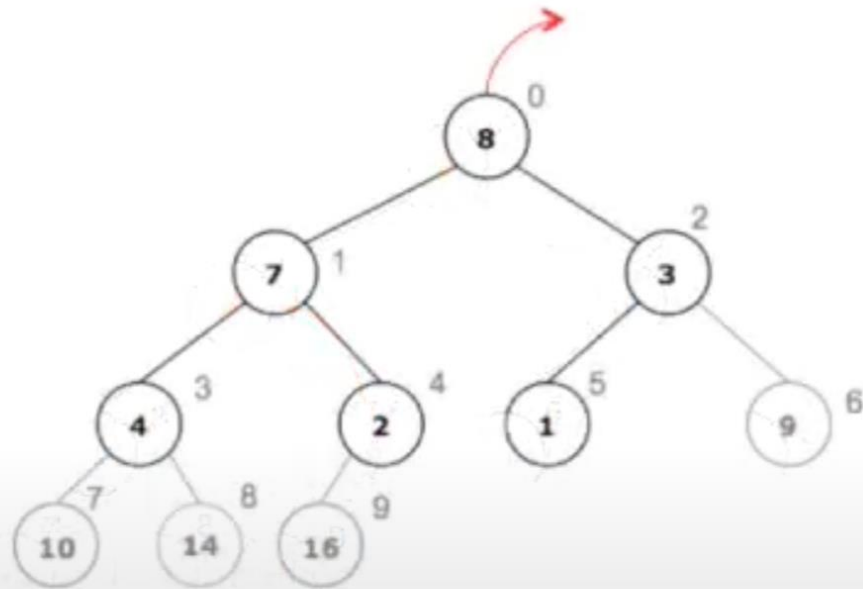
HeapSort

- Temos um Heap máximo



0	1	2	3	4	5	6	7	8	9
8	7	3	4	2	1	9	10	14	16

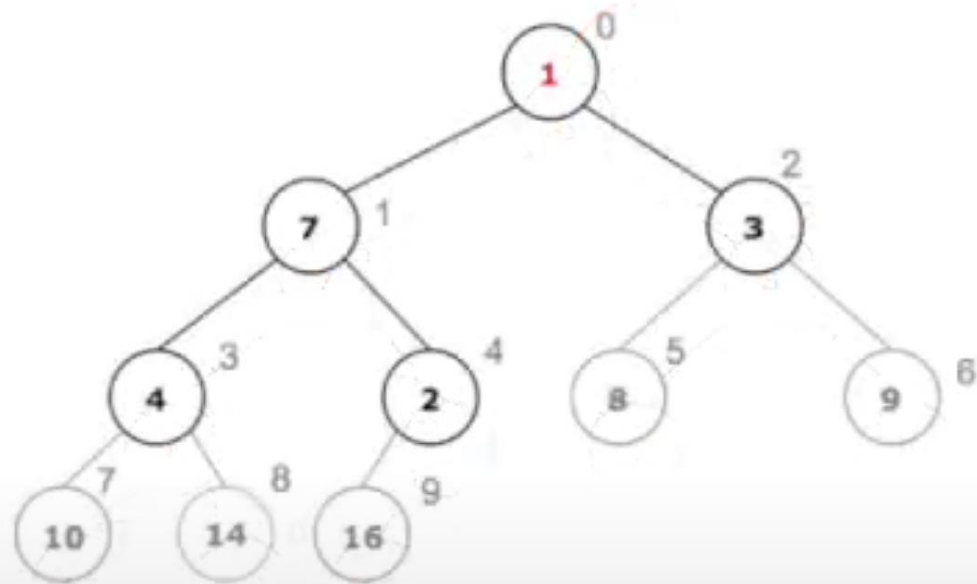
HeapSort



0	1	2	3	4	5	6	7	8	9
8	7	3	4	2	1	9	10	14	16

- Troca o 8 para a ultima posição do array de não ordenados

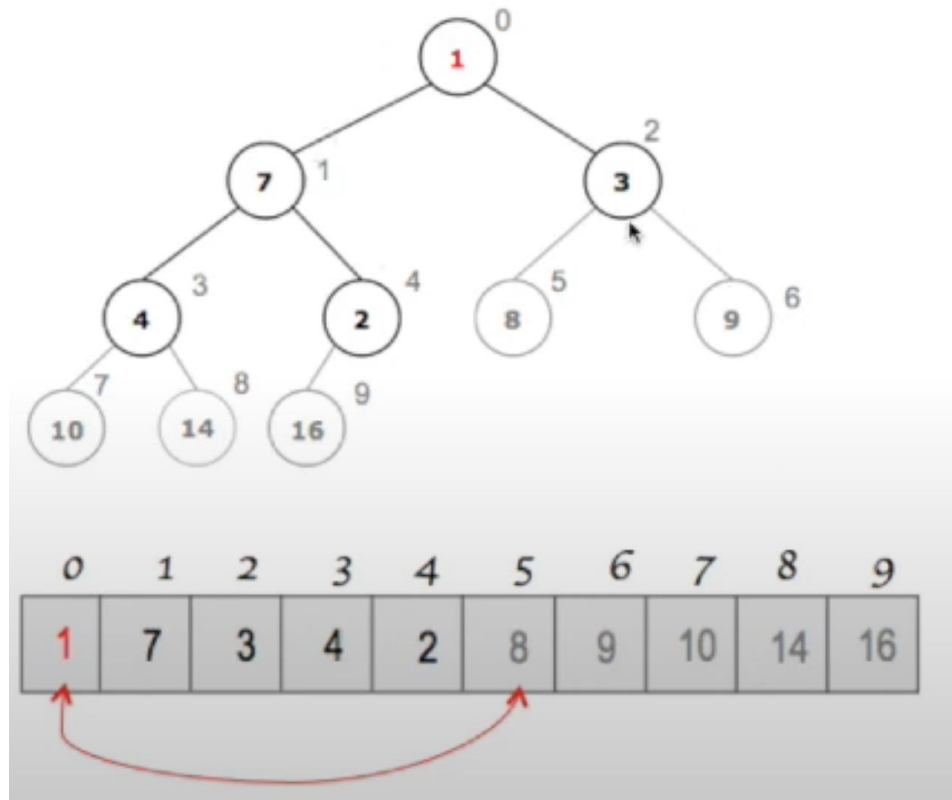
HeapSort



0	1	2	3	4	5	6	7	8	9
1	7	3	4	2	8	9	10	14	16

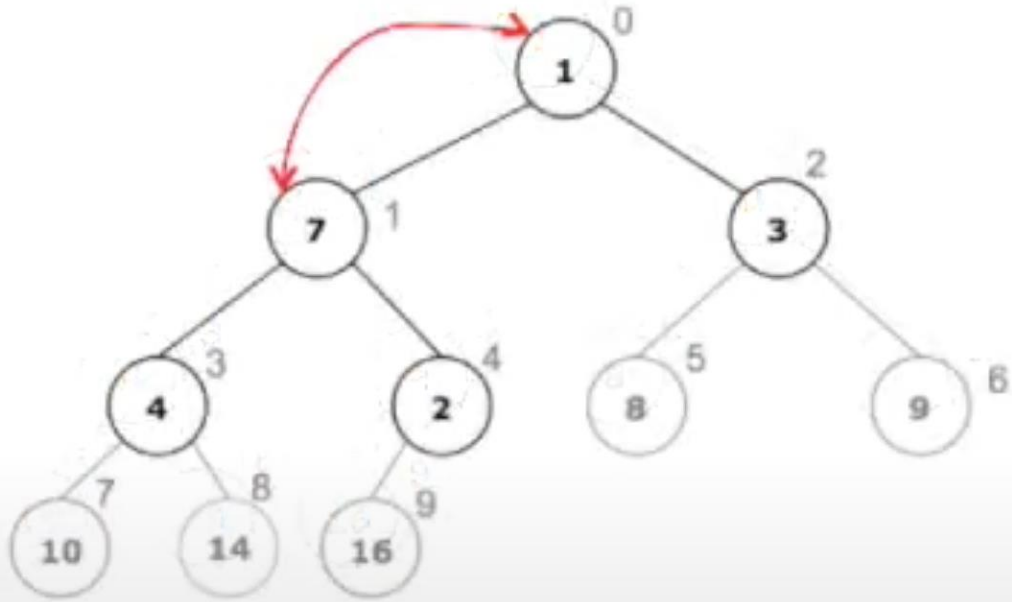
- Troca o 8 para a ultima posição do array de não ordenados

HeapSort



- Novamente, temos que transforma a árvore em uma heap

HeapSort

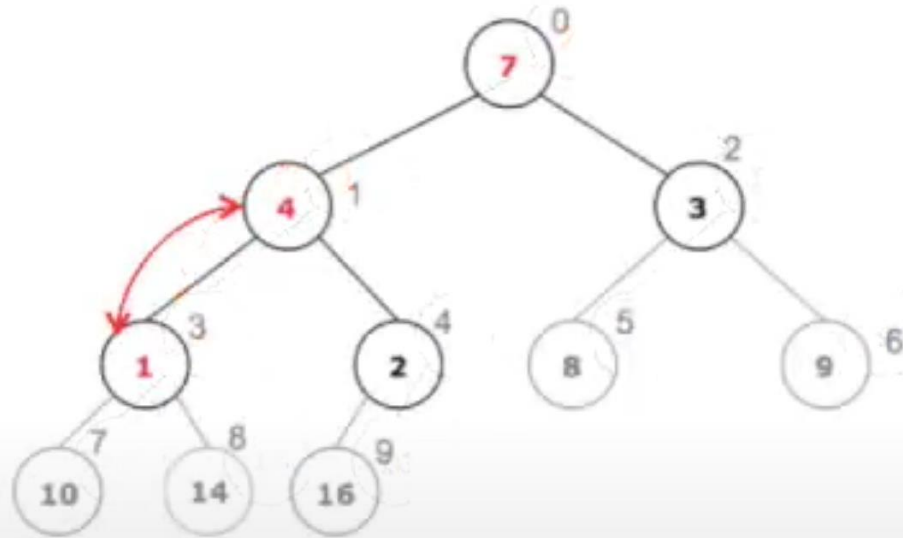


0	1	2	3	4	5	6	7	8	9
1	7	3	4	2	8	9	10	14	16

- Novamente, temos que transforma a árvore em uma heap

HeapSort

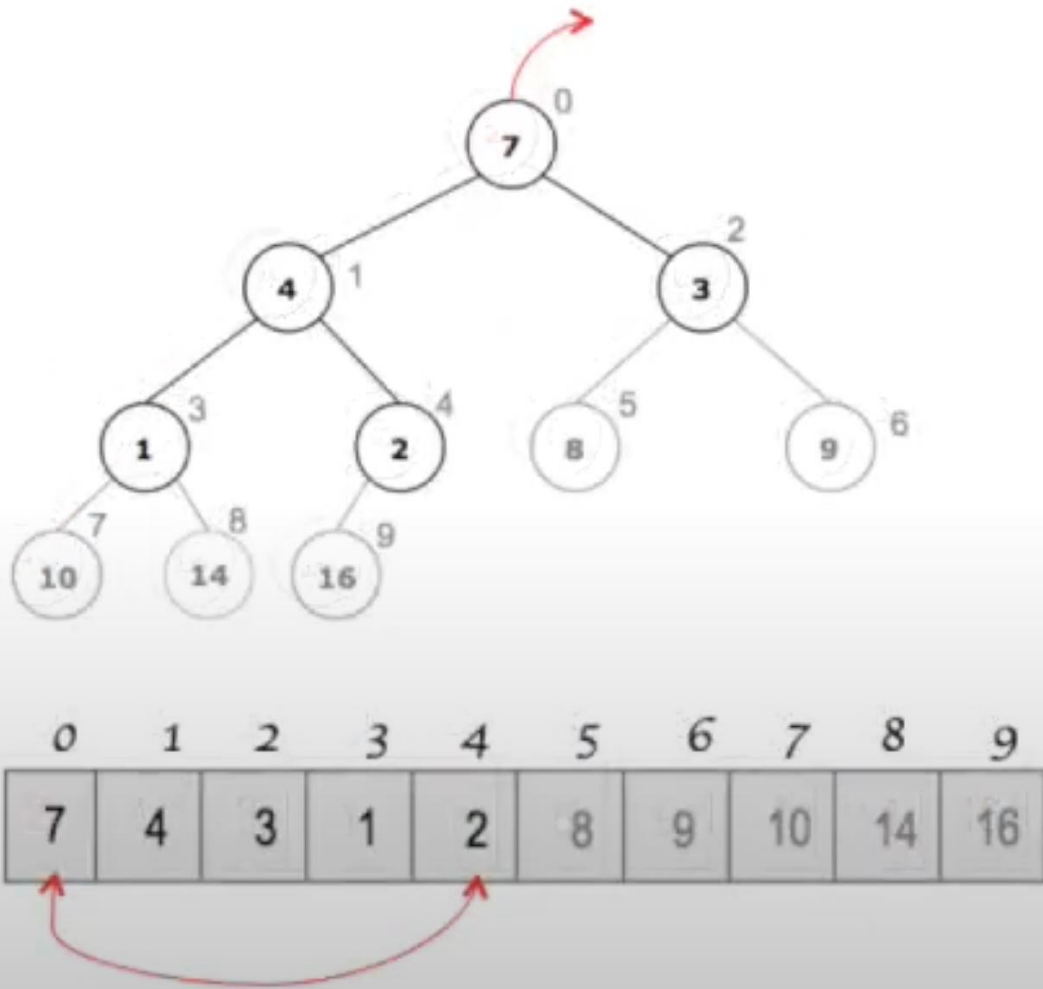
- Ajusta a sub-arvore



0	1	2	3	4	5	6	7	8	9
7	4	3	1	2	8	9	10	14	16

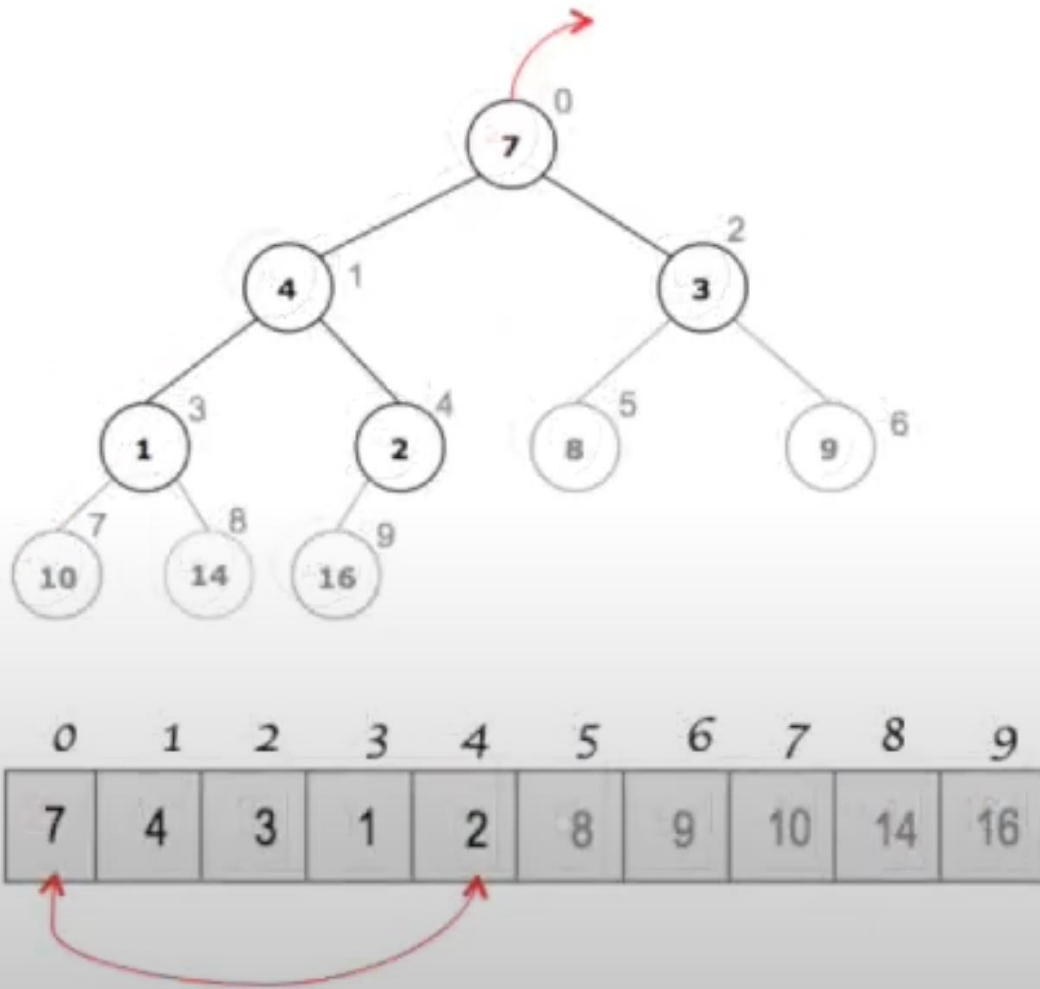
HeapSort

- Temos uma heap novamente!

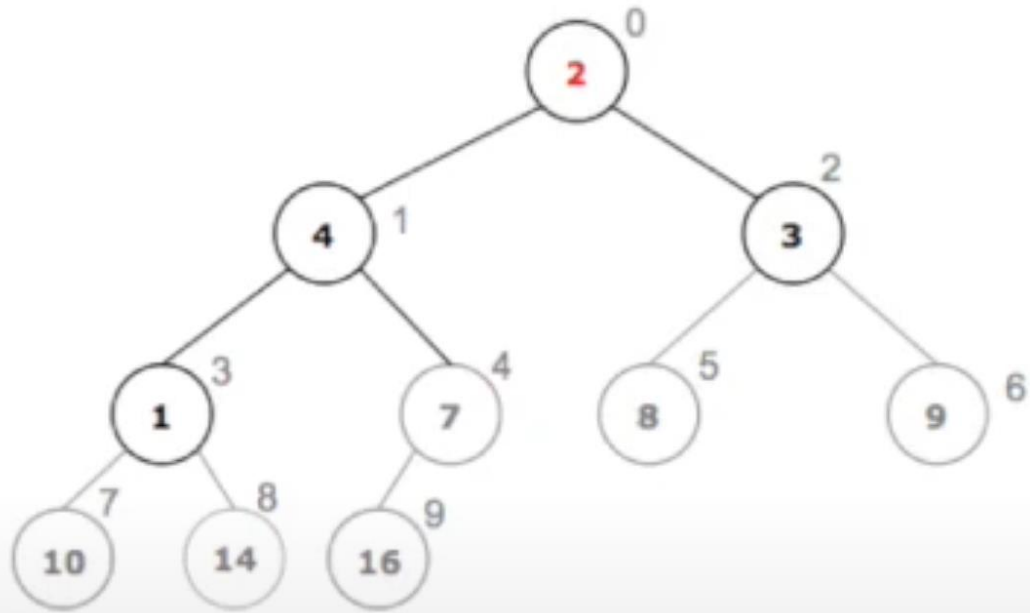


HeapSort

- Troca a raiz com o último elemento do array de não ordenados



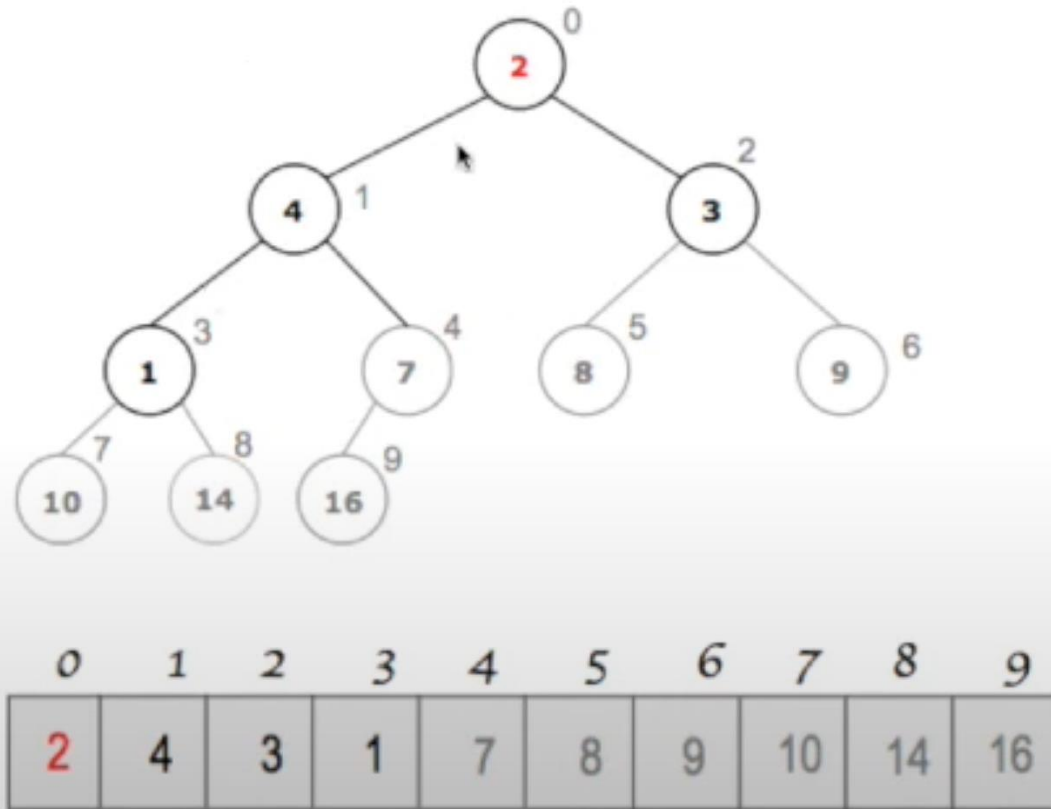
HeapSort



0	1	2	3	4	5	6	7	8	9
2	4	3	1	7	8	9	10	14	16

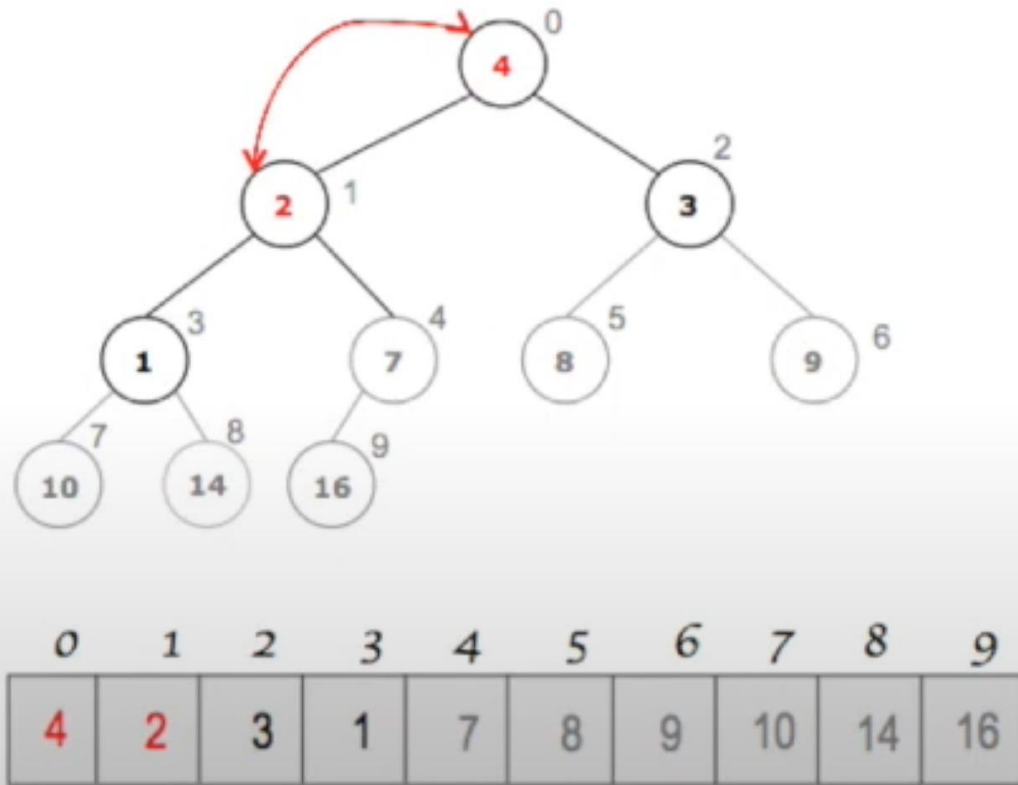
- Troca a raiz com o último elemento do array de não ordenados

HeapSort



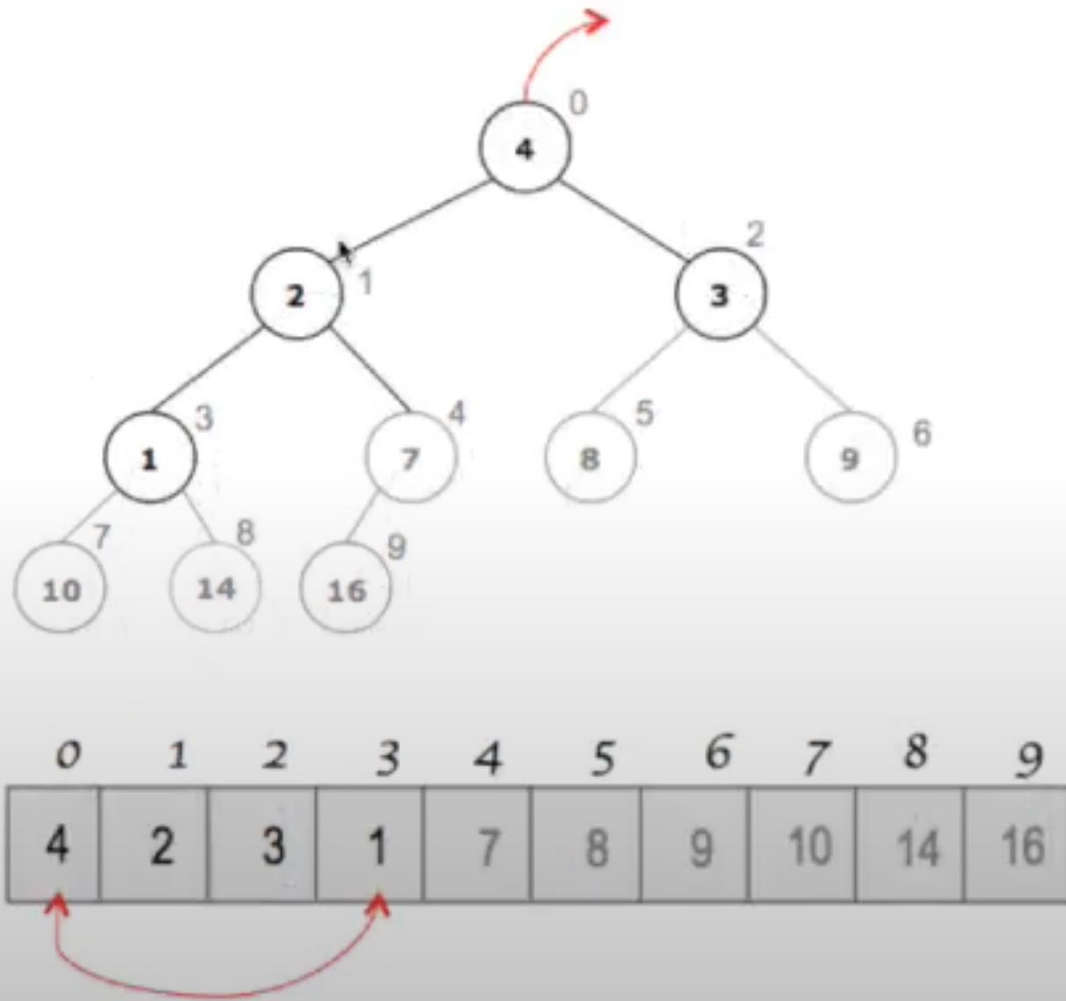
- Transformar em Heap novamente.

HeapSort



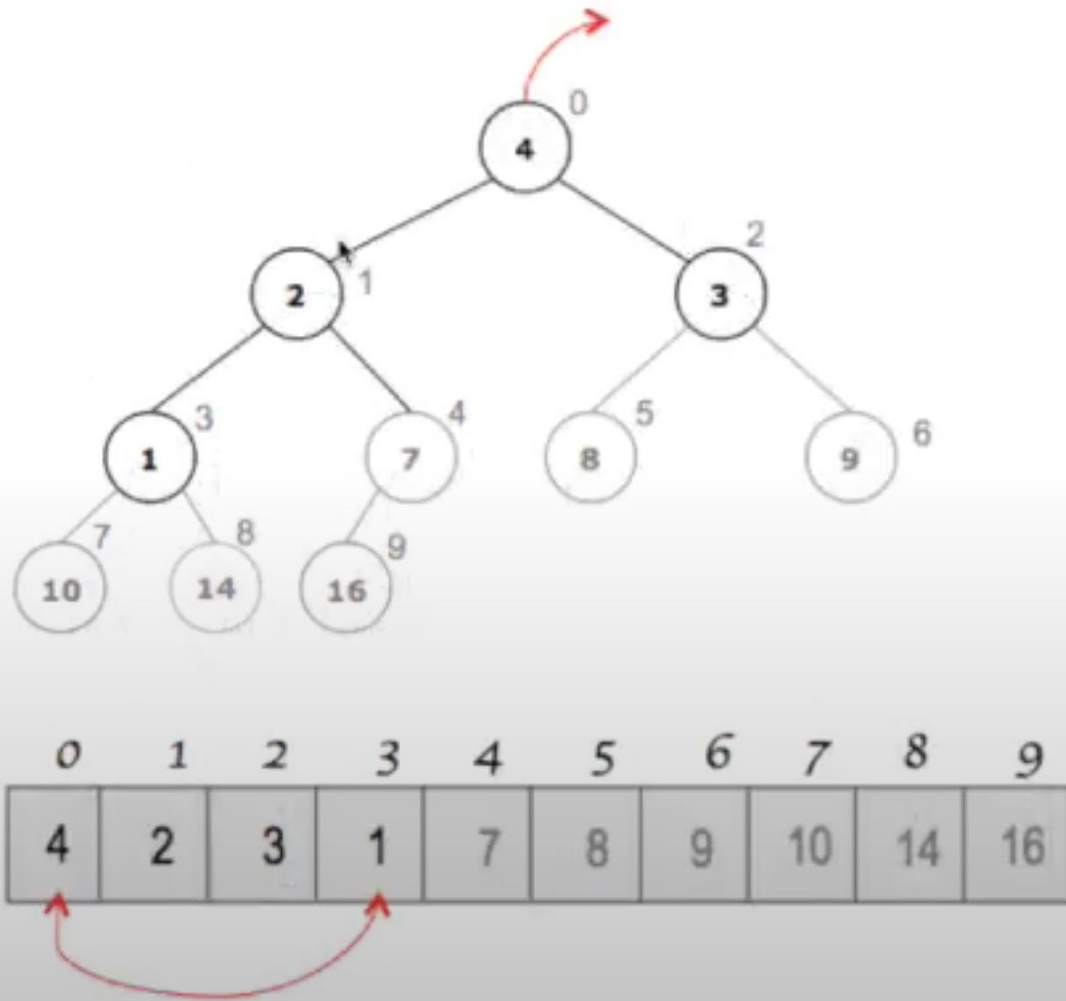
- Transformar em Heap novamente.

HeapSort



- Já temos uma heap novamente

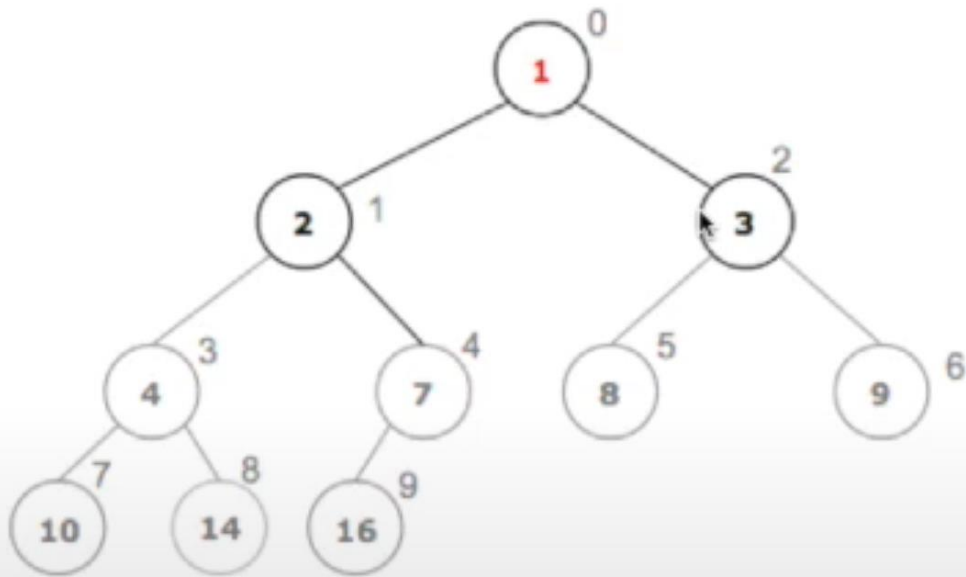
HeapSort



- Já temos uma heap novamente
- Trocamos a raiz pelo último elemento do vetor de não ordenados

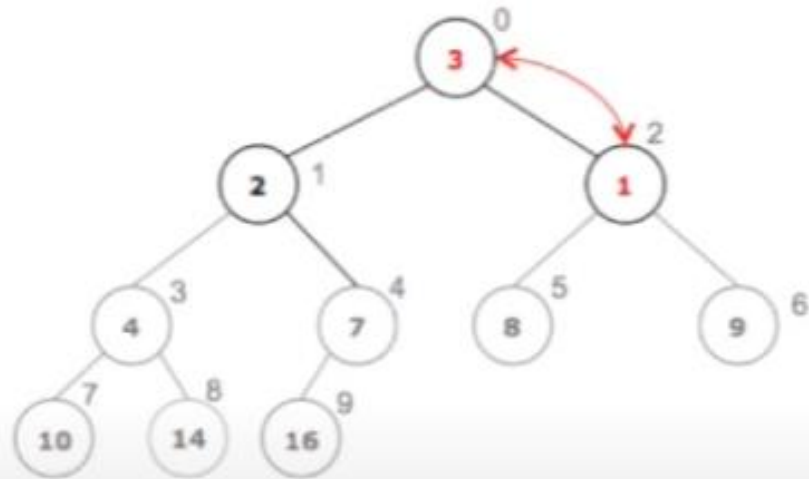
HeapSort

- Transformar em heap novamente!



0	1	2	3	4	5	6	7	8	9
1	2	3	4	7	8	9	10	14	16

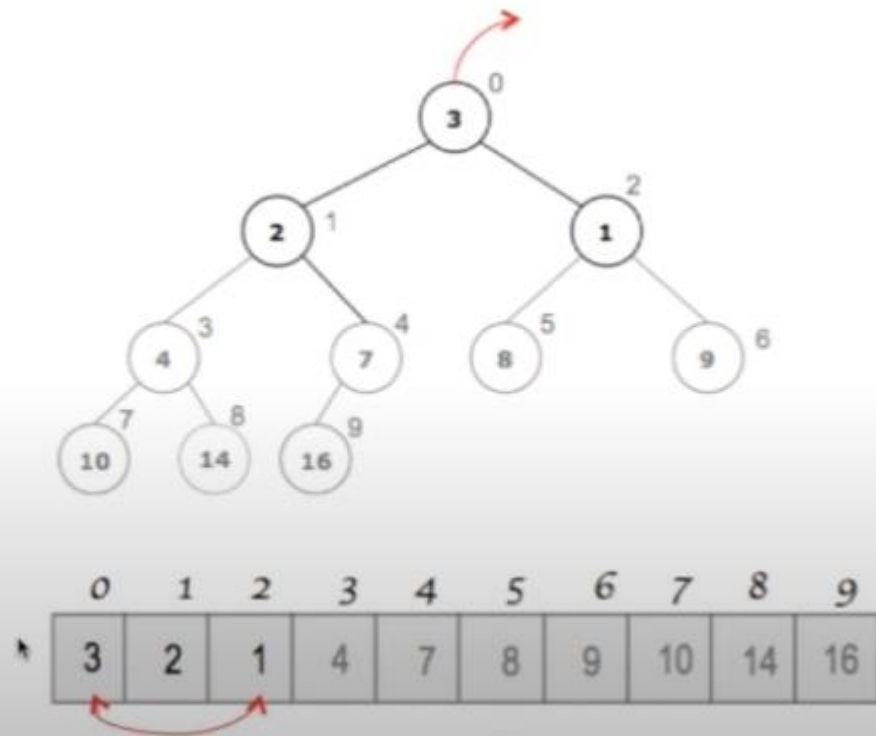
HeapSort



0	1	2	3	4	5	6	7	8	9
3	2	1	4	7	8	9	10	14	16

- Agora, transformar em um heap máximo novamente

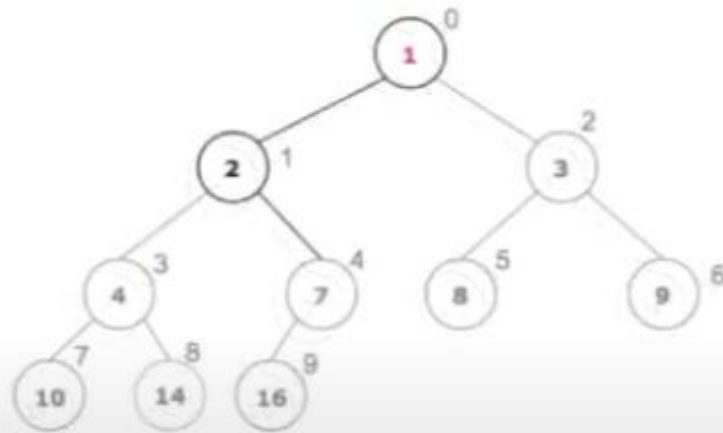
HeapSort



- Temos, novamente, Heap máximo, colocamos o nó raiz ao final da lista de desordenados.

HeapSort

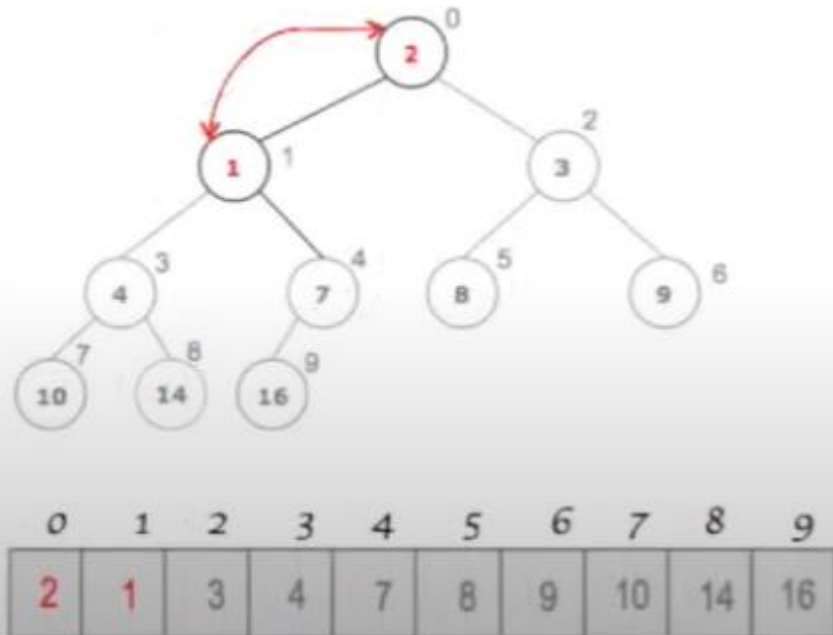
- Transformar em heap, mais uma vez



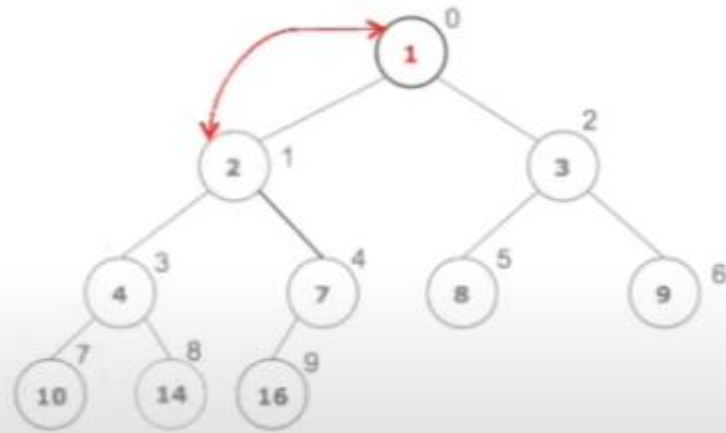
0	1	2	3	4	5	6	7	8	9
1	2	3	4	7	8	9	10	14	16

HeapSort

- Transformamos a árvore em um heap



HeapSort



0	1	2	3	4	5	6	7	8	9
1	2	3	4	7	8	9	10	14	16

- Colocamos a raiz na última posição no vetor dos vetores desordenados no array

The image features decorative blue lines in the corners. On the left, a vertical line descends from the top, with a small circle at its base. On the right, a horizontal line extends from the top, with a small circle at its end. At the bottom, there are two horizontal lines: one on the left with a small circle at its start, and one on the right with a small circle at its end. All lines are a vibrant blue color.

Heap – Algoritmo

HeapSort - Algoritmo

// Algoritmo – livro do Comen

```
public static void heapSort(int a[]){  
    for(int k = a.length/2-1; k >= 0; k--){  
        maxHeapfy(a, k, a.length);  
    }
```

```
}
```

```
for(int n = a.length-1; n >= 1; n--){
```

```
    int temp = a[0];
```

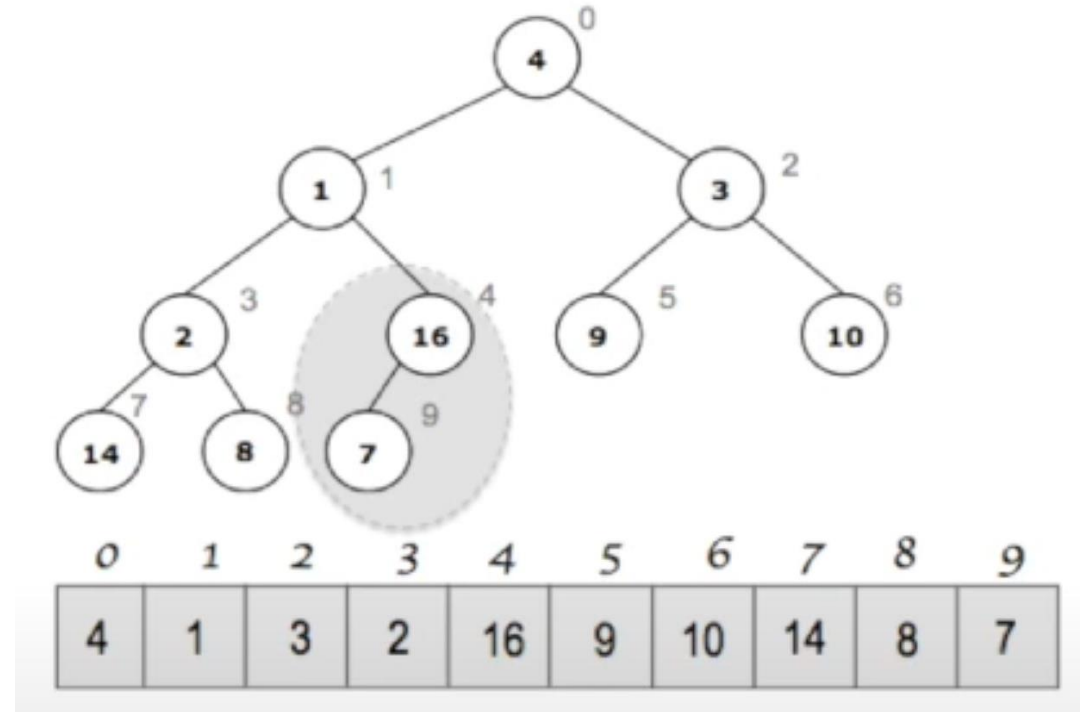
```
    a[0] = a[n];
```

```
    a[n] = temp;
```

```
    maxHeapfy(a, 0, n); // torna o array uma árvore heap
```

```
}
```

```
}
```



HeapSort - Algoritmo

// Algoritmo – livro do Comen

```
public static void heapSort(int a[]){  
    for(int k = a.length/2-1; k >= 0; k--){  
        maxHeapfy(a, k, a.length);  
    }
```

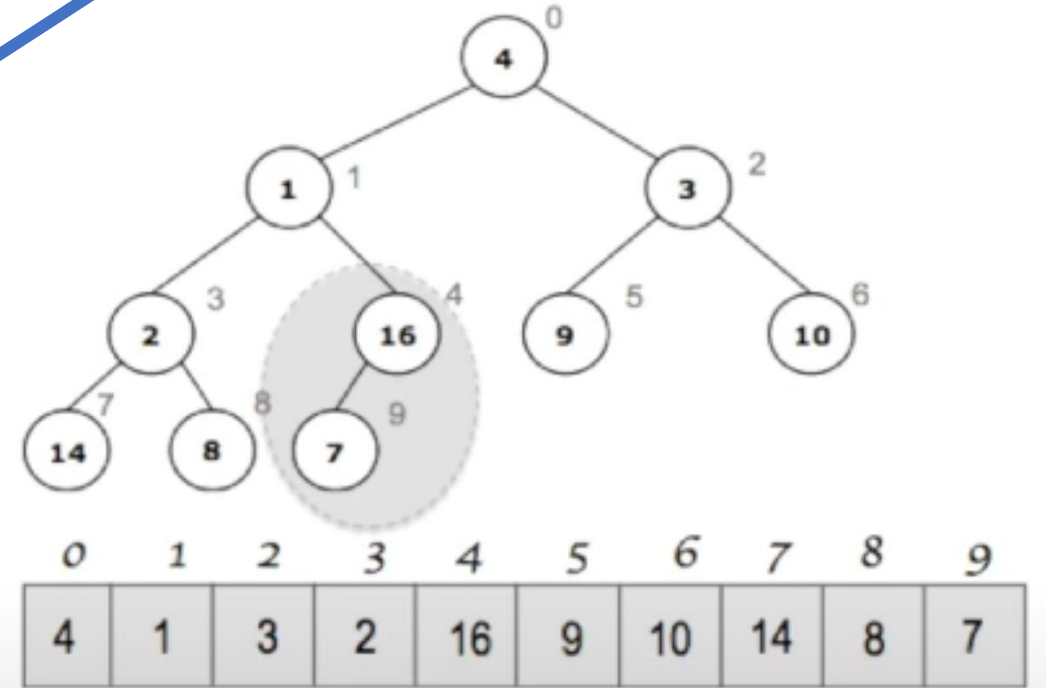
```
}
```

```
for(int n = a.length-1; n >= 1; n--){  
    int temp = a[0];  
    a[0] = a[n];  
    a[n] = temp;  
    maxHeapfy(a, 0, n); // torna o array uma árvore heap
```

```
}
```

```
}
```

Responsável por transformar a
árvore em uma árvore heap



HeapSort - Algoritmo

$a.length = 10$

$k = 10 / 2 - 1 \Rightarrow \text{int } k = 4$

// percebam que os demais nós do array são filhos

// Algoritmo – livro do Comen

```
public static void heapSort(int a[]){  
    for(int k = a.length/2-1; k >= 0; k--){  
        maxHeapfy(a, k, a.length);  
    }
```

```
}
```

```
for(int n = a.length-1; n >= 1; n--){
```

```
    int temp = a[0];
```

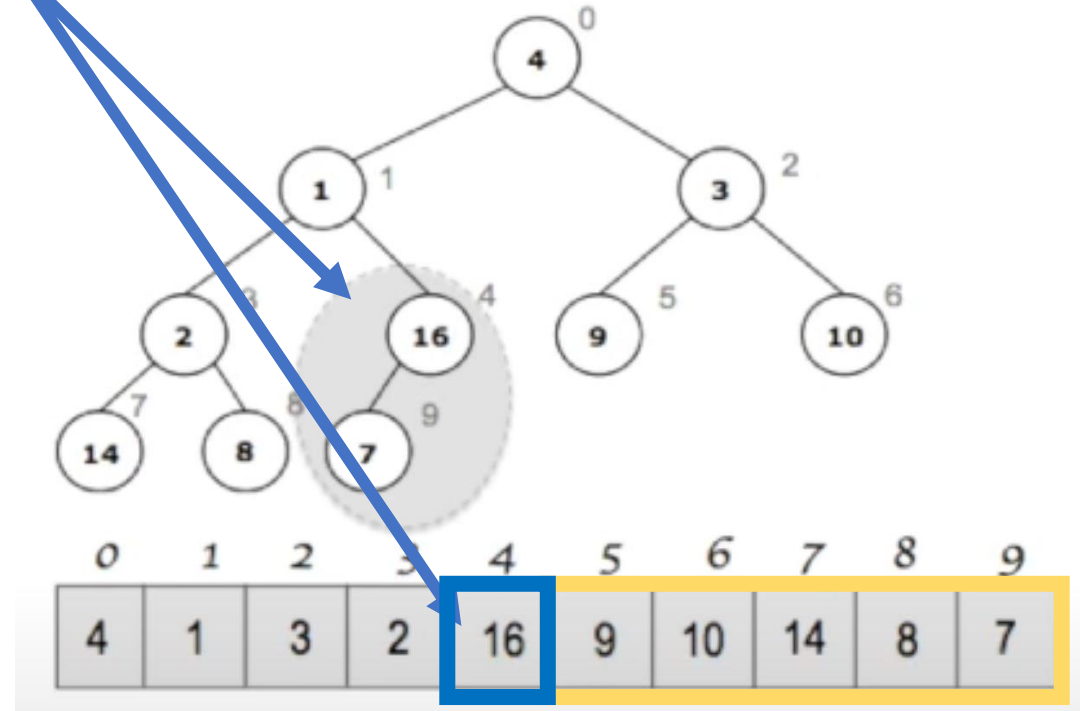
```
    a[0] = a[n];
```

```
    a[n] = temp;
```

```
    maxHeapfy(a, 0, n); // torna o array uma árvore heap
```

```
}
```

```
}
```



HeapSort - Algoritmo

// Algoritmo – livro do Comen

```
public static void heapSort(int a[]){  
    for(int k = a.length/2-1; k >= 0; k--){  
        maxHeapfy(a, k, a.length);  
    }
```

```
}
```

```
for(int n = a.length-1; n >= 1; n--){
```

```
    int temp = a[0];
```

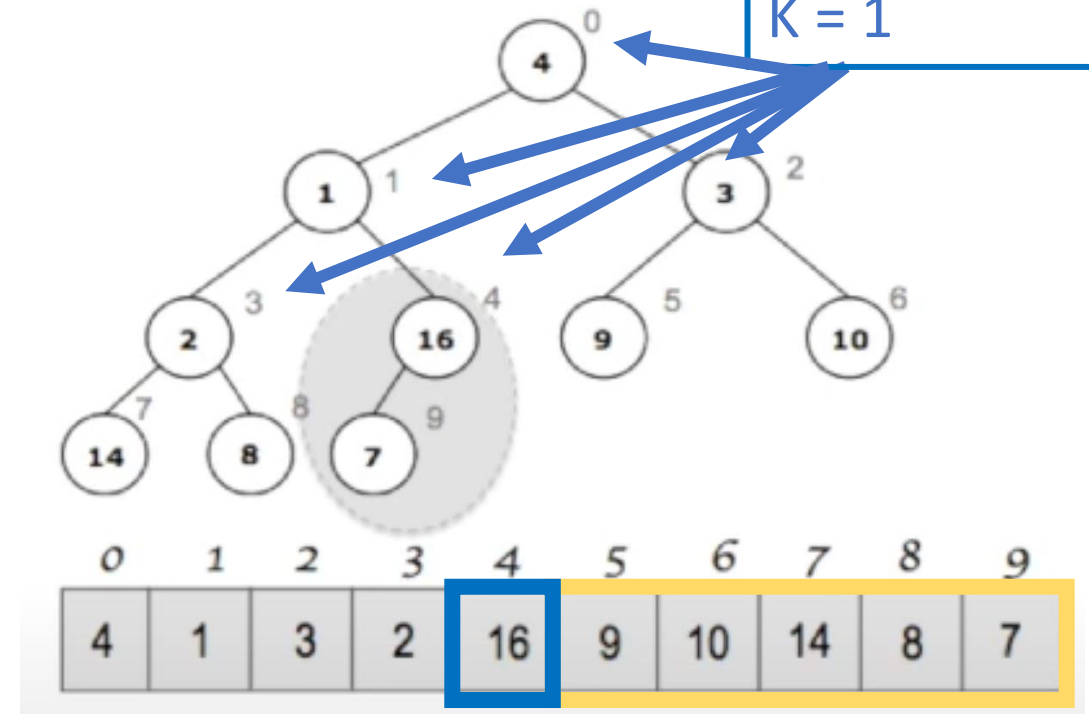
```
    a[0] = a[n];
```

```
    a[n] = temp;
```

```
    maxHeapfy(a, 0, n); // torna o array uma árvore heap
```

```
}
```

```
}
```



HeapSort - Algoritmo

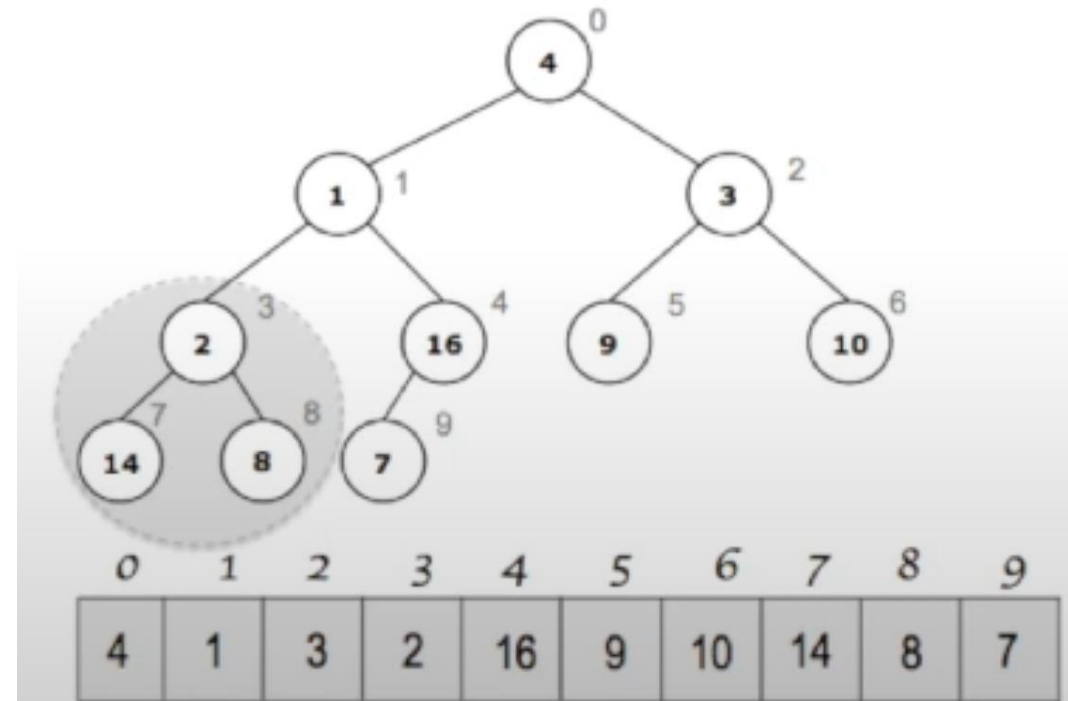
// Algoritmo – livro do Comen

```
public static void maxHeapfy(int a[], int i, int n){  
    int maior = i;  
    int l = 2*i+1; // posição do filho a esquerda de i  
    int r = 2*i+2; // posição do filho a direita de i  
    if(l < n && a[l]>a[i]) maior = l;  
    if(r < n && a[r]>a[maior]) maior = r;  
    if(maior != i){  
        int temp = a[i];  
        a[i] = a[maior];  
        a[maior] = temp;  
        maxHeapfy(a, maior, n);  
    }  
}
```

HeapSort - Algoritmo

// Algoritmo – livro do Comen

```
public static void maxHeapfy(int a[], int i, int n){  
    int maior = i;  
    int l = 2*i+1; // posição do filho a esquerda de i  
    int r = 2*i+2; // posição do filho a direita de i  
    if(l < n && a[l]>a[i]) maior = l;  
    if(r < n && a[r]>a[maior]) maior = r;  
    if(maior != i){  
        int temp = a[i];  
        a[i] = a[maior];  
        a[maior] = temp;  
        maxHeapfy(a, maior, n);  
    }  
}
```



HeapSort - Algoritmo

// Algoritmo – livro do Comen

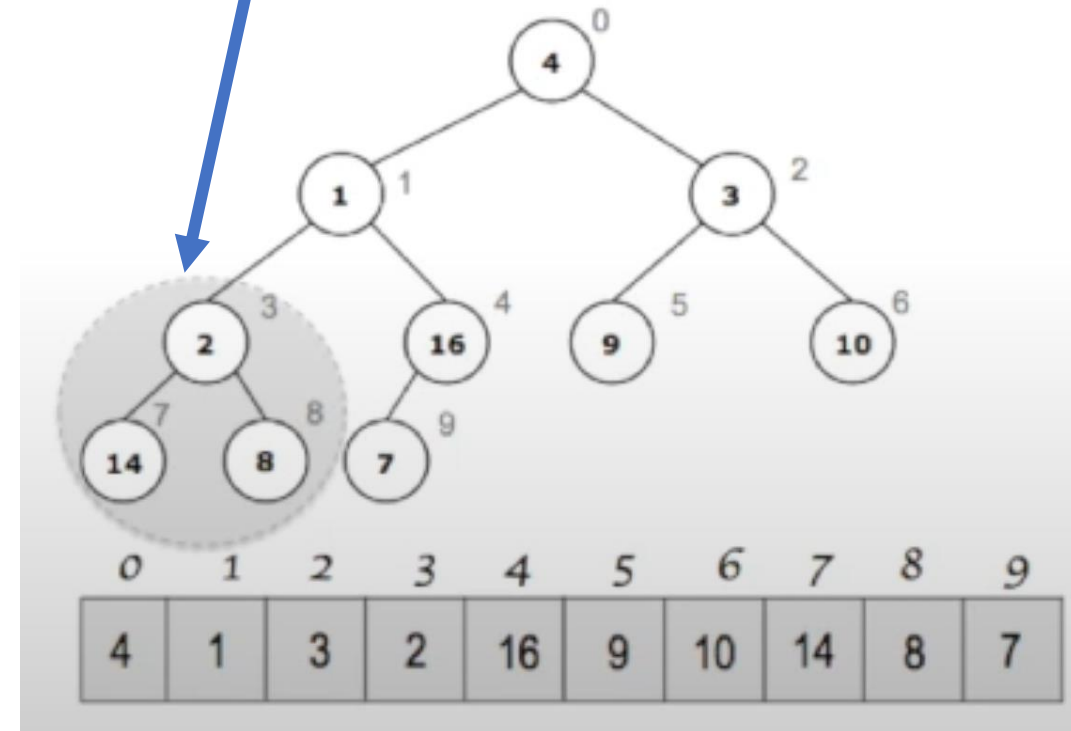
```
public static void maxHeapfy(int a[], int i, int n){  
    int maior = i;  
    int l = 2*i+1; // posição do filho a esquerda de i  
    int r = 2*i+2; // posição do filho a direita de i  
    if(l < n && a[l]>a[i]) maior = l;  
    if(r < n && a[r]>a[maior]) maior = r;  
    if(maior != i){  
        int temp = a[i];  
        a[i] = a[maior];  
        a[maior] = temp;  
        maxHeapfy(a, maior, n);  
    }  
}
```

$i = 3$

$\text{maior} = 3$

$l = 2 * 3 + 1 = 7$

$r = 2 * 3 + 2 = 8$



HeapSort - Algoritmo

$i = 3$
 $\text{maior} = 3$
 $l = 7$
 $r = 8$

$l = 7 > n = 10$

// Algoritmo – livro do Comen

```
public static void maxHeapfy(int a[], int i, int n){
```

```
    int maior = i;
```

```
    int l = 2*i+1; // posição do filho a esquerda de i
```

```
    int r = 2*i+2; // posição do filho a direita de i
```

```
    if(l < n && a[l] > a[i]) maior = l;
```

```
    if(r < n && a[r] > a[maior]) maior = r;
```

```
    if(maior != i){
```

```
        int temp = a[i];
```

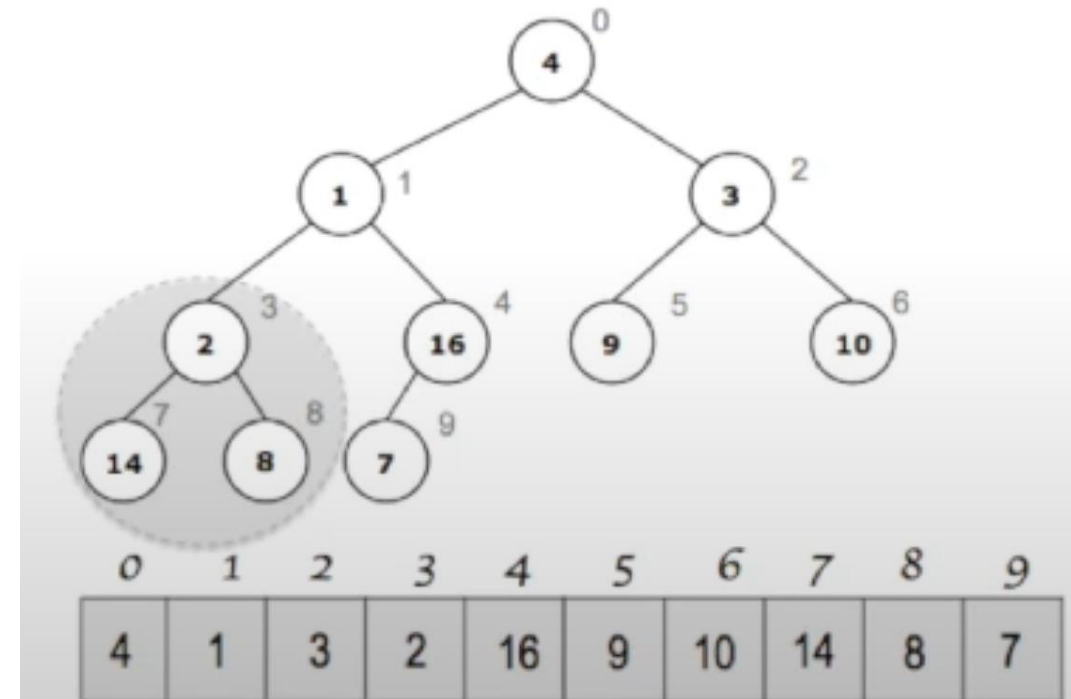
```
        a[i] = a[maior];
```

```
        a[maior] = temp;
```

```
        maxHeapfy(a, maior, n);
```

```
    }
```

```
}
```



HeapSort - Algoritmo

$i = 3$
 $\text{maior} = 3$
 $l = 7$
 $r = 8$

$a[7] > a[3]$
14 2

// Algoritmo – livro do Comen

```
public static void maxHeapfy(int a[], int i, int n){
```

```
    int maior = i;
```

```
    int l = 2*i+1; // posição do filho a esquerda de i
```

```
    int r = 2*i+2; // posição do filho a direita de i
```

```
    if(l < n && a[l]>a[i]) maior = l;
```

```
    if(r < n && a[r]>a[maior]) maior = r;
```

```
    if(maior != i){
```

```
        int temp = a[i];
```

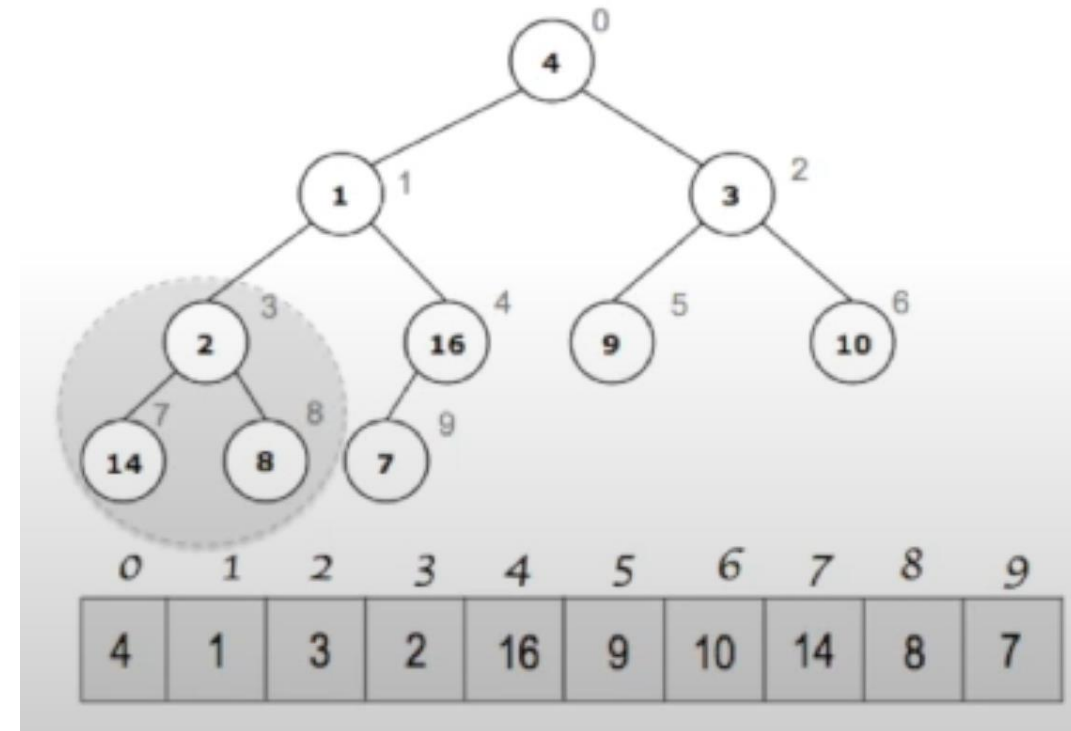
```
        a[i] = a[maior];
```

```
        a[maior] = temp;
```

```
        maxHeapfy(a, maior, n);
```

```
    }
```

```
}
```



HeapSort - Algoritmo

i = 3
maior = 3
l = 7
r = 8

a[7] > a[3]
14 2

// Algoritmo – livro do Comen

```
public static void maxHeapfy(int a[], int i, int n){
```

```
    int maior = i;
```

```
    int l = 2*i+1; // posição do filho a esquerda de i
```

```
    int r = 2*i+2; // posição do filho a direita de i
```

```
    if(l < n && a[l]>a[i]) maior = l;
```

```
    if(r < n && a[r]>a[maior]) maior = r;
```

```
    if(maior != i){
```

```
        int temp = a[i];
```

```
        a[i] = a[maior];
```

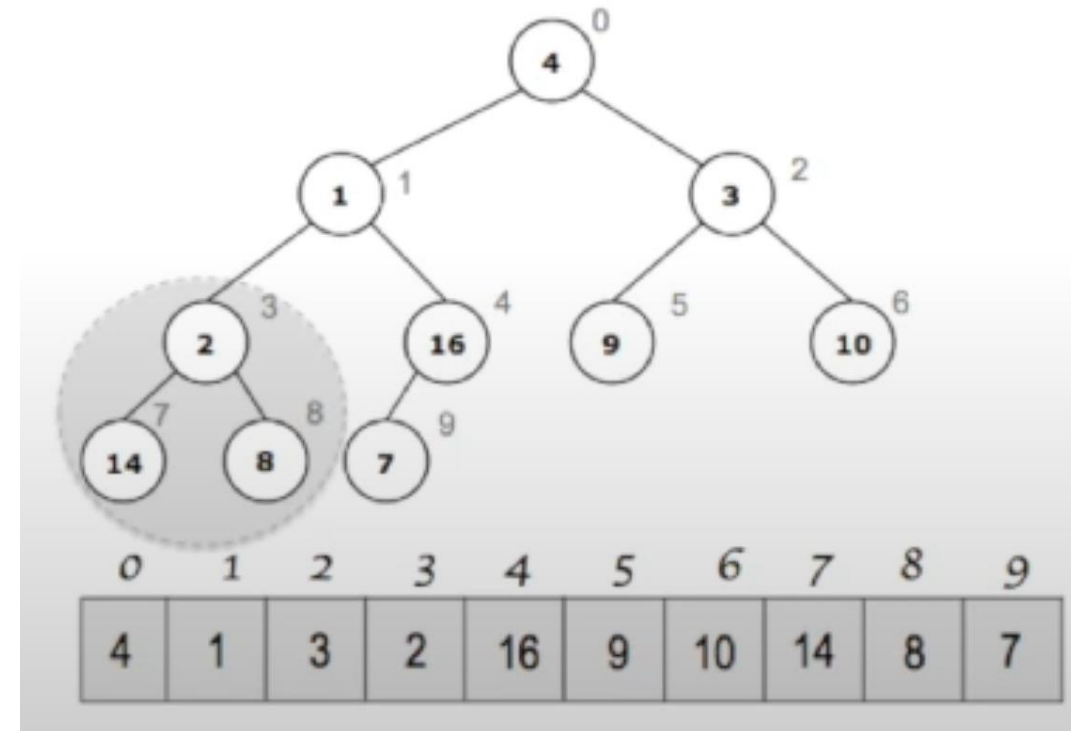
```
        a[maior] = temp;
```

```
        maxHeapfy(a, maior, n);
```

```
    }
```

```
}
```

maior = 7 // valor 14



HeapSort - Algoritmo

// Algoritmo – livro do Comen

```
public static void maxHeapfy(int a[], int i, int n){
```

```
    int maior = i;
```

```
    int l = 2*i+1; // posição do filho a esquerda de i
```

```
    int r = 2*i+2; // posição do filho a direita de i
```

```
    if(l < n && a[l]>a[i]) maior = l;
```

```
    if(r < n && a[r]>a[maior]) maior = r;
```

```
    if(maior != i){
```

```
        int temp = a[i];
```

```
        a[i] = a[maior];
```

```
        a[maior] = temp;
```

```
        maxHeapfy(a, maior, n);
```

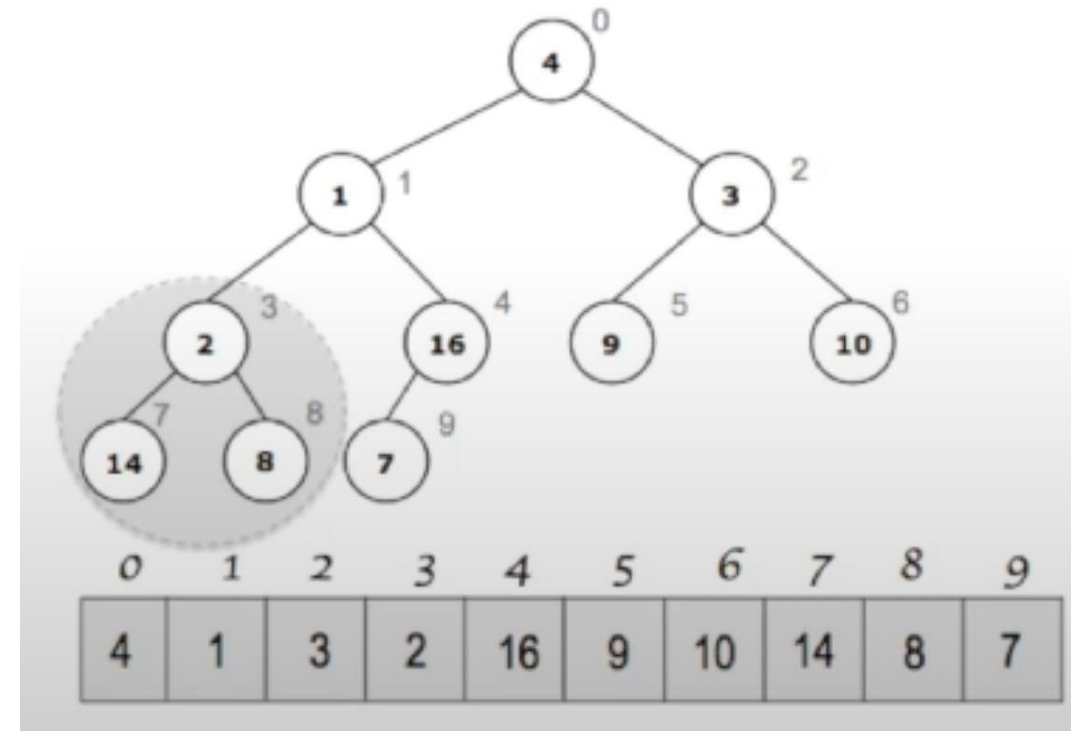
```
    }
```

```
}
```

i = 3
maior = 7
l = 7
r = 8

a[7] > a[3]
14 2

maior = 7 // valor 14



HeapSort - Algoritmo

$i = 3$
maior = 7
 $l = 7$
 $r = 8$

// Algoritmo – livro do Comen

```
public static void maxHeapfy(int a[], int i, int n){
```

```
    int maior = i;
```

```
    int l = 2*i+1; // posição do filho a esquerda de i
```

```
    int r = 2*i+2; // posição do filho a direita de i
```

```
    if(l < n && a[l]>a[i]) maior = l;
```

```
    if(r < n && a[r]>a[maior]) maior = r;
```

```
    if(maior != i){
```

```
        int temp = a[i];
```

```
        a[i] = a[maior];
```

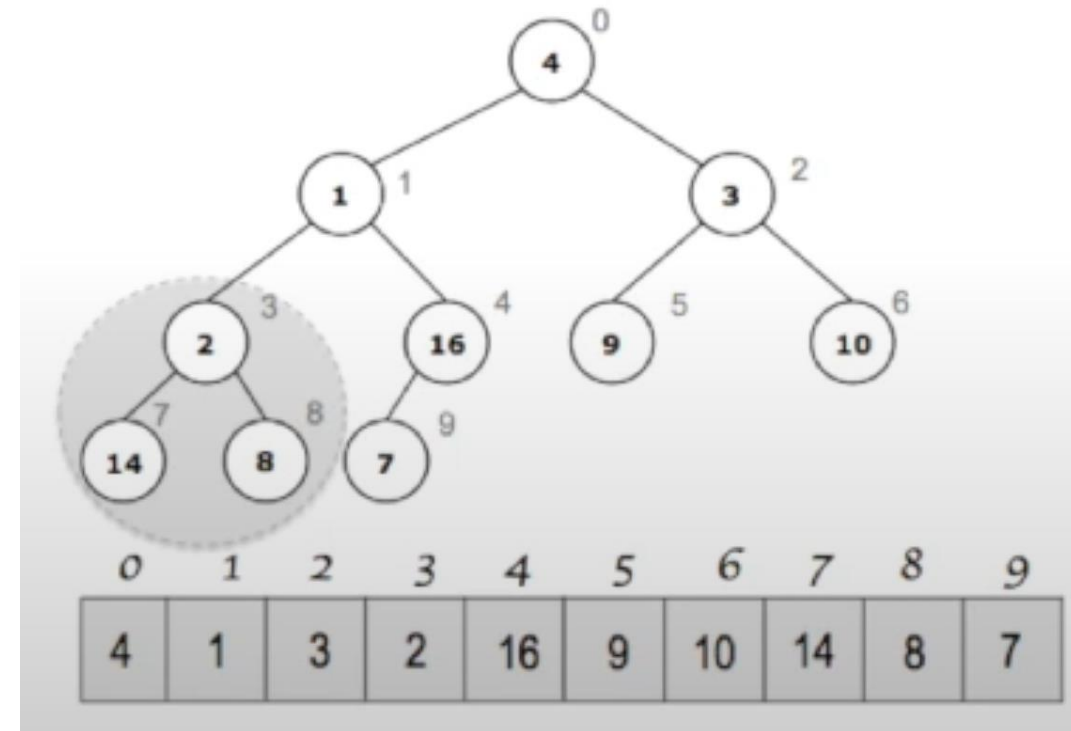
```
        a[maior] = temp;
```

```
        maxHeapfy(a, maior, n);
```

```
    }
```

```
}
```

$a[r] > a[\text{maior}]$
8 14



HeapSort - Algoritmo

$i = 3$
 $\text{maior} = 7$
 $l = 7$
 $r = 8$

$\text{maior} \neq r$

// teste para verificar se o maior
continuou sendo o pai, do
contrário faz as trocas

// Algoritmo – livro do Comen

```
public static void maxHeapfy(int a[], int i, int n){
```

```
    int maior = i;
```

```
    int l = 2*i+1; // posição do filho a esquerda de i
```

```
    int r = 2*i+2; // posição do filho a direita de i
```

```
    if(l < n && a[l] > a[i]) maior = l;
```

```
    if(r < n && a[r] > a[maior]) maior = r;
```

```
    if(maior != i){
```

```
        int temp = a[i];
```

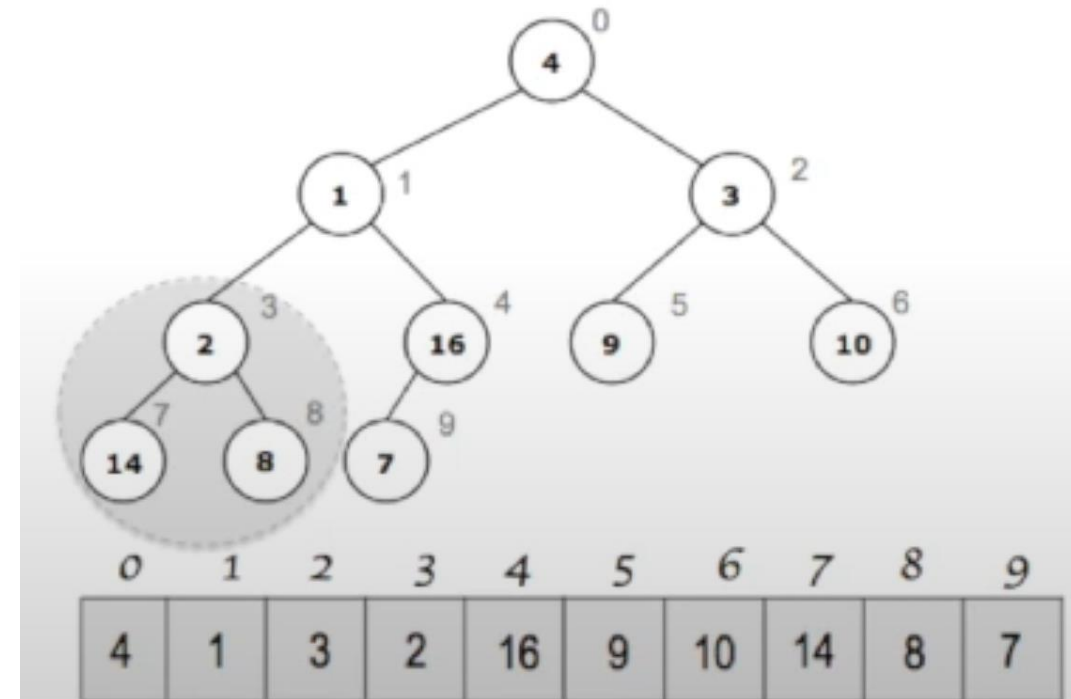
```
        a[i] = a[maior];
```

```
        a[maior] = temp;
```

```
        maxHeapfy(a, maior, n);
```

```
    }
```

```
}
```



HeapSort - Algoritmo

// Algoritmo – livro do Comen

```
public static void maxHeapfy(int a[], int i, int n){
```

```
    int maior = i;
```

```
    int l = 2*i+1; // posição do filho a esquerda de i
```

```
    int r = 2*i+2; // posição do filho a direita de i
```

```
    if(l < n && a[l]>a[i]) maior = l;
```

```
    if(r < n && a[r]>a[maior]) maior = r;
```

```
    if(maior != i){
```

```
        int temp = a[i];
```

```
        a[i] = a[maior];
```

```
        a[maior] = temp;
```

```
        maxHeapfy(a, maior, n);
```

```
    }
```

```
}
```

i = 3

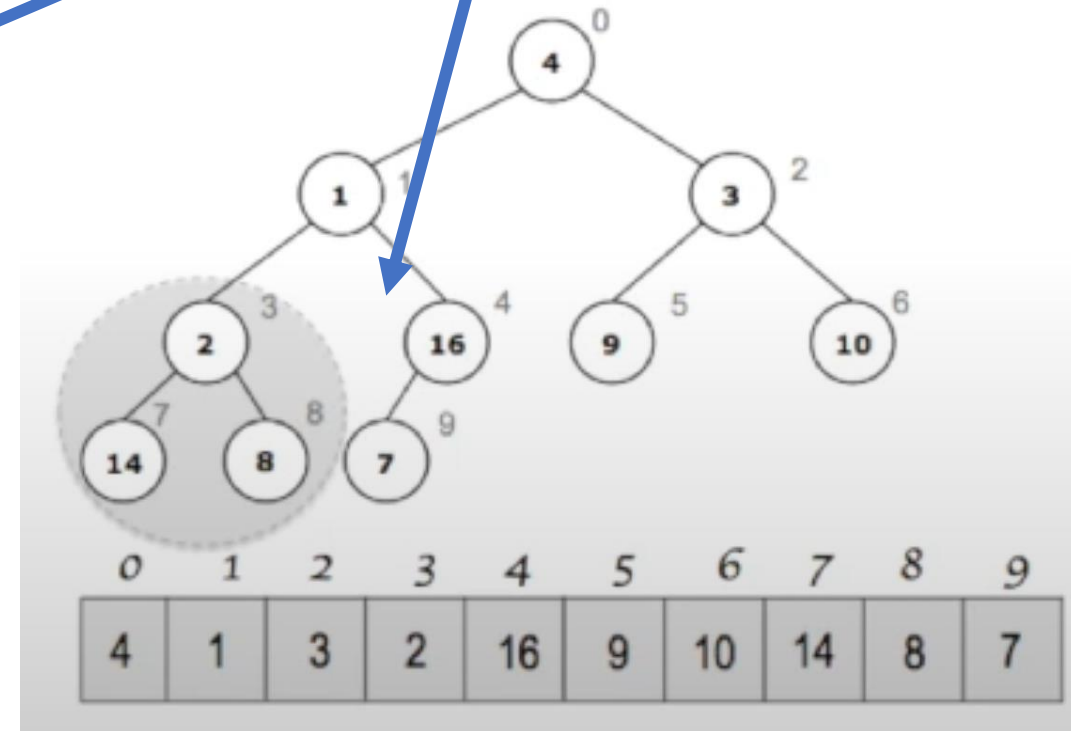
maior = 7

l = 7

r = 8

maior != r

// teste para verificar se o maior
continuou sendo o pai, do
contrário faz as trocas



HeapSort - Algoritmo

$i = 3$
 $\text{maior} = 7$
 $l = 7$
 $r = 8$

Chama heapfy para verificar a sub-árvore do 7, como não há, nada é feito

// Algoritmo – livro do Comen

```
public static void maxHeapfy(int a[], int i, int n){
```

```
    int maior = i;
```

```
    int l = 2*i+1; // posição do filho a esquerda de i
```

```
    int r = 2*i+2; // posição do filho a direita de i
```

```
    if(l < n && a[l]>a[i]) maior = l;
```

```
    if(r < n && a[r]>a[maior]) maior = r;
```

```
    if(maior != i){
```

```
        int temp = a[i];
```

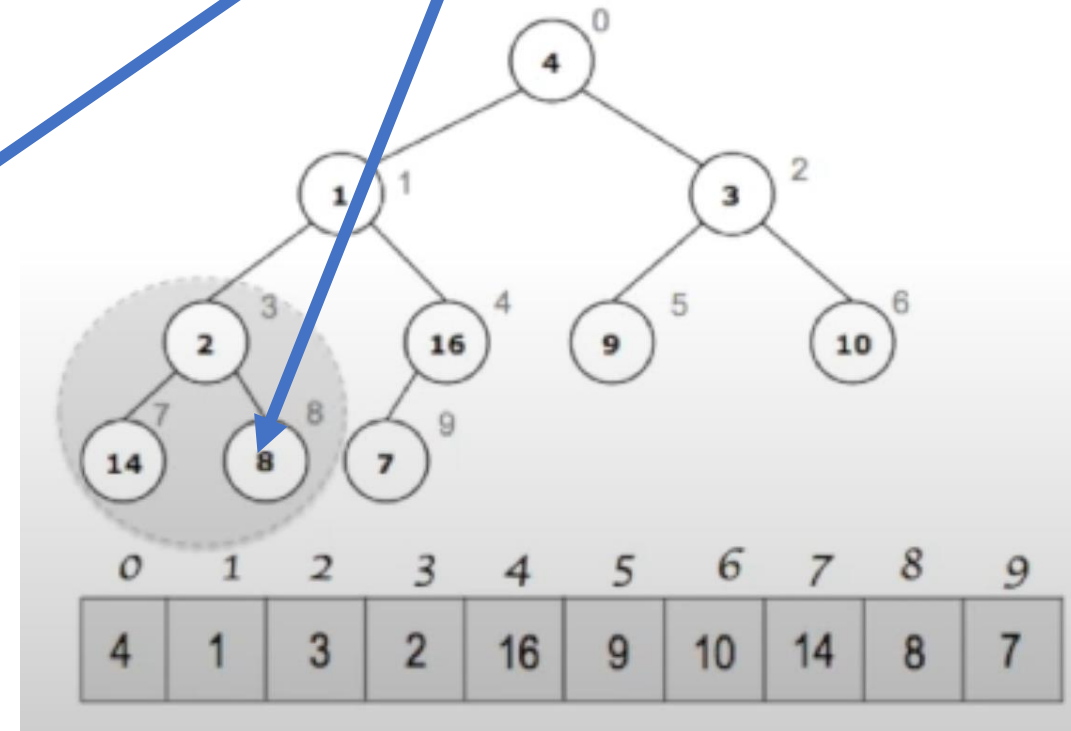
```
        a[i] = a[maior];
```

```
        a[maior] = temp;
```

```
        maxHeapfy(a, maior, n);
```

```
    }
```

```
}
```



HeapSort - Algoritmo

// Algoritmo – livro do Comen

```
public static void heapSort(int a[]){  
    for(int k = a.length/2-1; k >= 0; k--){  
        maxHeapfy(a, k, a.length);  
    }
```

```
    for(int n = a.length-1; n >= 1; n--){  
        int temp = a[0];  
        a[0] = a[n];  
        a[n] = temp;  
        maxHeapfy(a, 0, n); // torna o array uma árvore heap
```

```
    }
```

```
}
```

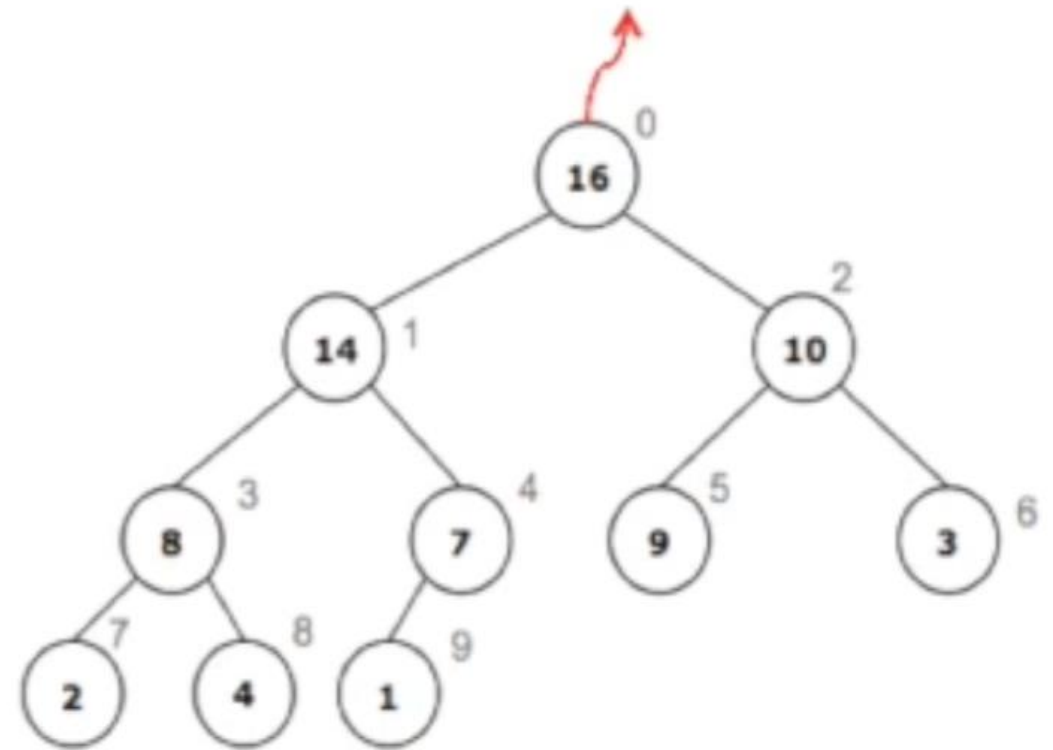
Quando terminar esse laço temos todos os elementos na posição correta (árvore heap)

HeapSort - Algoritmo

// Algoritmo – livro do Comen

```
public static void heapSort(int a[]){  
    for(int k = a.length/2-1; k >= 0; k--){  
        maxHeapfy(a, k, a.length);  
    }  
    for(int n = a.length-1; n >= 1; n--){  
        int temp = a[0];  
        a[0] = a[n];  
        a[n] = temp;  
        maxHeapfy(a, 0, n); // torna o  
    }  
}
```

Quando terminar esse laço temos todos os elementos na posição correta (árvore heap)



HeapSort - Algoritmo

// Algoritmo – livro do Comen

```
public static void heapSort(int a[]){  
    for(int k = a.length/2-1; k >= 0; k--){  
        maxHeapfy(a, k, a.length);  
    }
```

```
    for(int n = a.length-1; n >= 1; n--){
```

```
        int temp = a[n];
```

```
        a[0] = a[n];
```

```
        a[n] = temp;
```

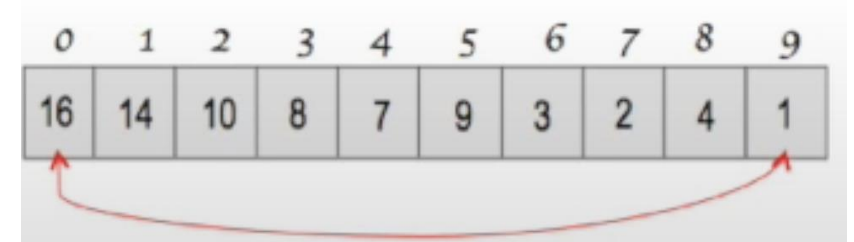
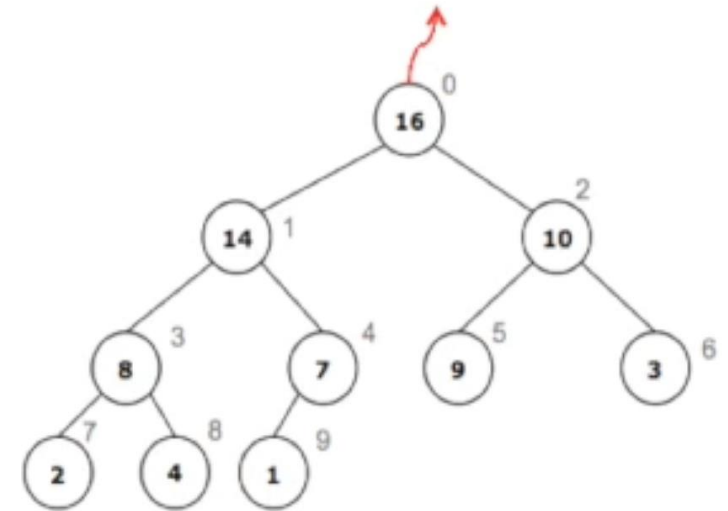
```
        maxHeapfy(a, 0, n);
```

```
        // torna o array uma árvore heap
```

```
    }
```

```
}
```

Troca os elementos

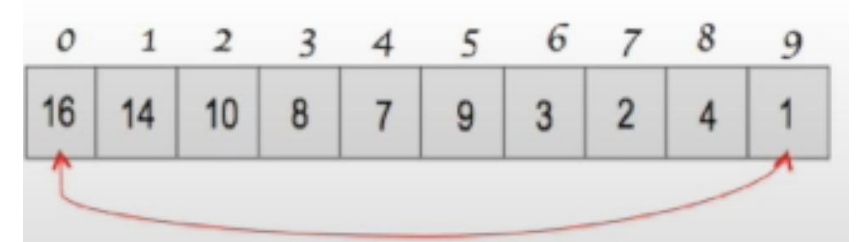
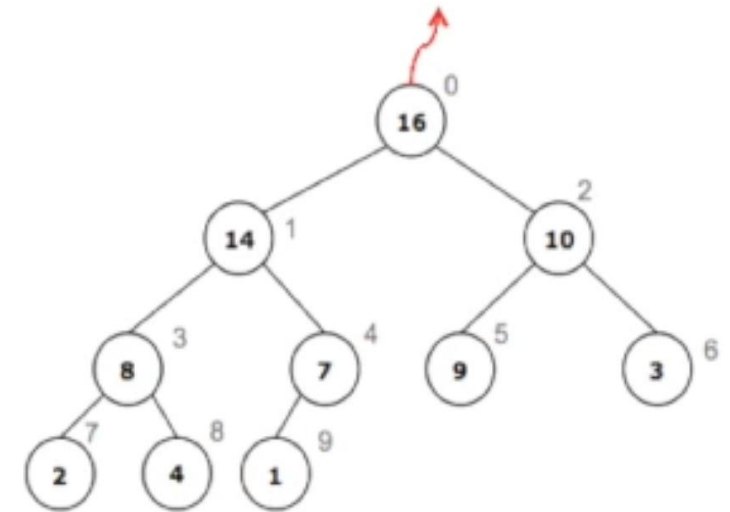


HeapSort - Algoritmo

// Algoritmo – livro do Comen

```
public static void heapSort(int a[]){  
    for(int k = a.length/2-1; k >= 0; k--){  
        maxHeapfy(a, k, a.length);  
    }  
    for(int n = a.length-1; n >= 1; n--){  
        int temp = a[0];  
        a[0] = a[n];  
        a[n] = temp;  
        maxHeapfy(a, 0, n);  
        // torna o array uma árvore heap  
    }  
}
```

Perceba que o $N - 1 = 9$ Neste caso vai virar 9



HeapSort - Algoritmo

// Algoritmo – livro do Comen

```
public static void heapSort(int a[]){  
    for(int k = a.length/2-1; k >= 0; k--){  
        maxHeapfy(a, k, a.length);  
    }
```

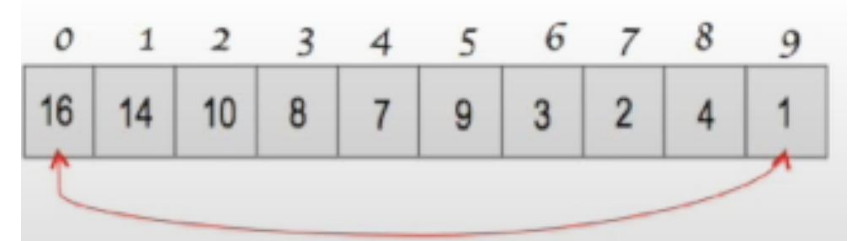
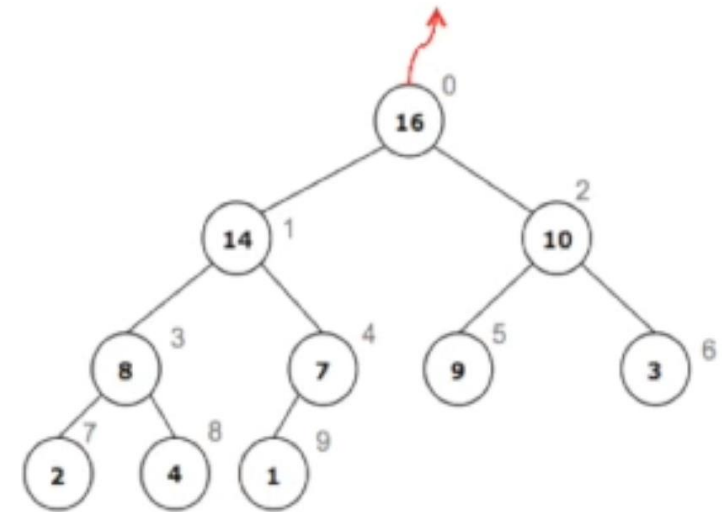
```
    for(int n = a.length-1; n >= 1; n--){  
        int temp = a[0];  
        a[0] = a[n];  
        a[n] = temp;  
        maxHeapfy(a, 0, n);  
        // torna o array uma árvore heap
```

```
    }
```

```
}
```

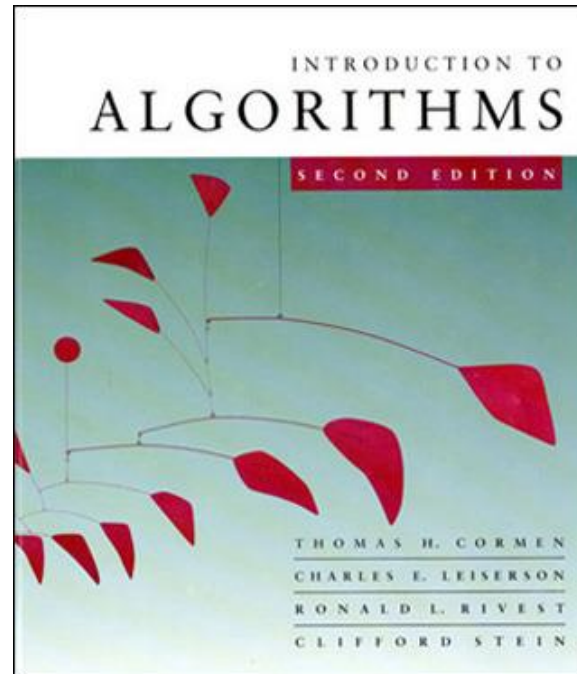
Neste caso vai virar 9

E chamar heap somente para a parte a ser ordenada



Leitura importante

livro “Algorithms” de Cormen et al.



Classificação e Pesquisa de Dados

Cristiano Santos

cristiano.santos@amf.edu.br