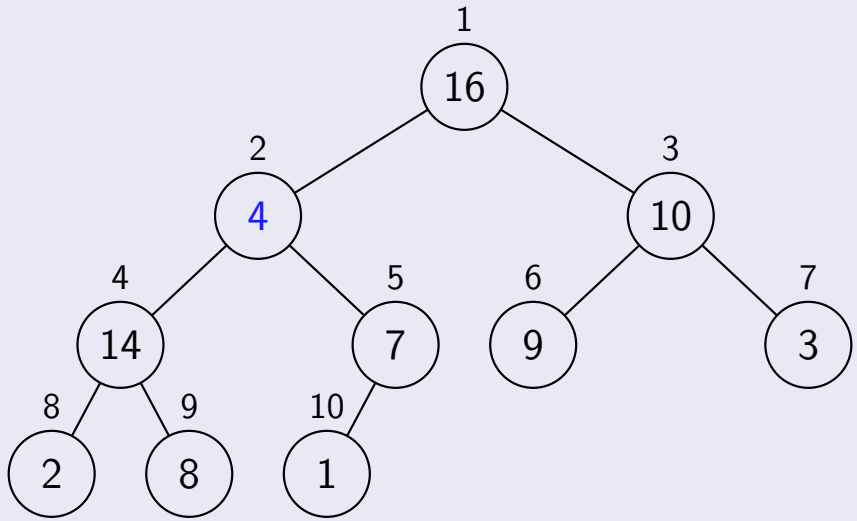
Aula 20 – Heapsort

Norton T. Roman & Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

2023

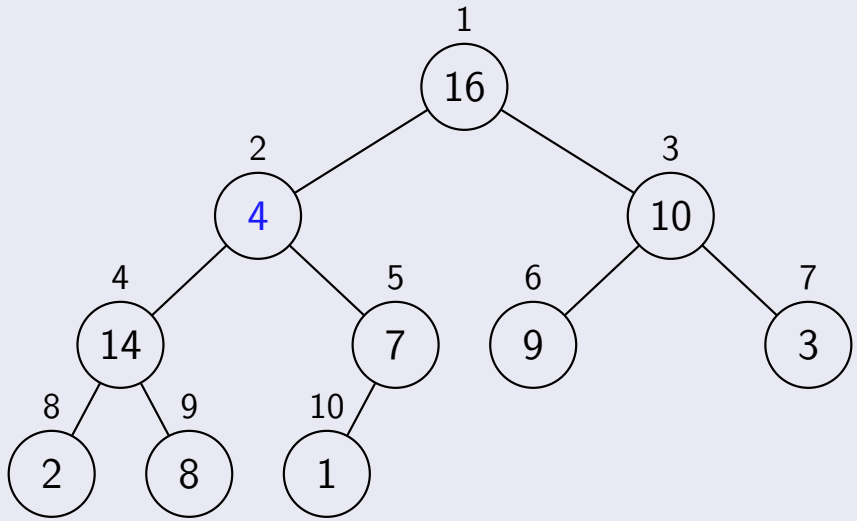
Mantendo a Propriedade do Heap

- Vimos que, no caso de alguma mudança no heap alterar sua propriedade:



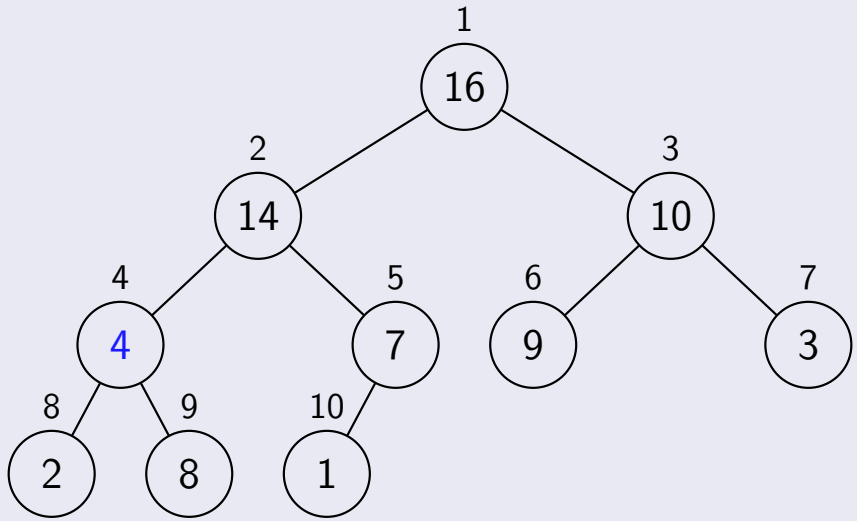
Mantendo a Propriedade do Heap

- Vimos que, no caso de alguma mudança no heap alterar sua propriedade:
  - Trocamos o elemento problemático de lugar com o maior filho



Mantendo a Propriedade do Heap

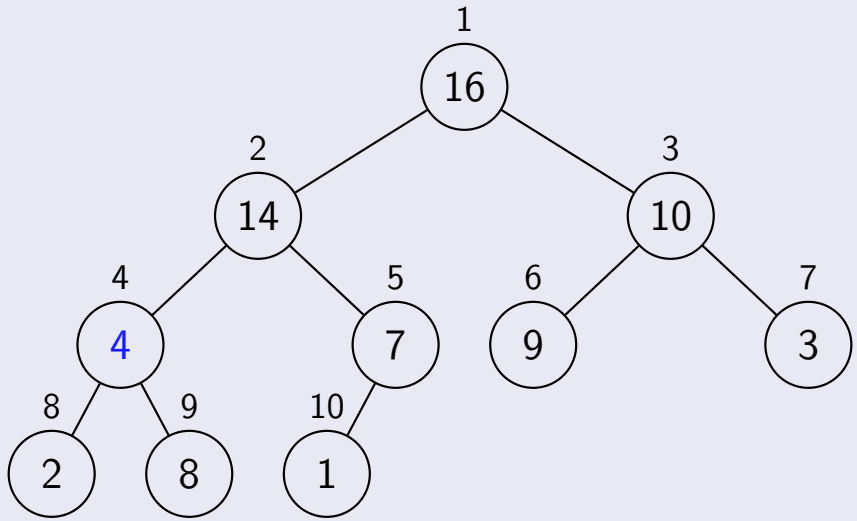
- Vimos que, no caso de alguma mudança no heap alterar sua propriedade:
  - Trocamos o elemento problemático de lugar com o maior filho



Heaps

Mantendo a Propriedade do Heap

- Vimos que, no caso de alguma mudança no heap alterar sua propriedade:
  - Trocamos o elemento problemático de lugar com o maior filho

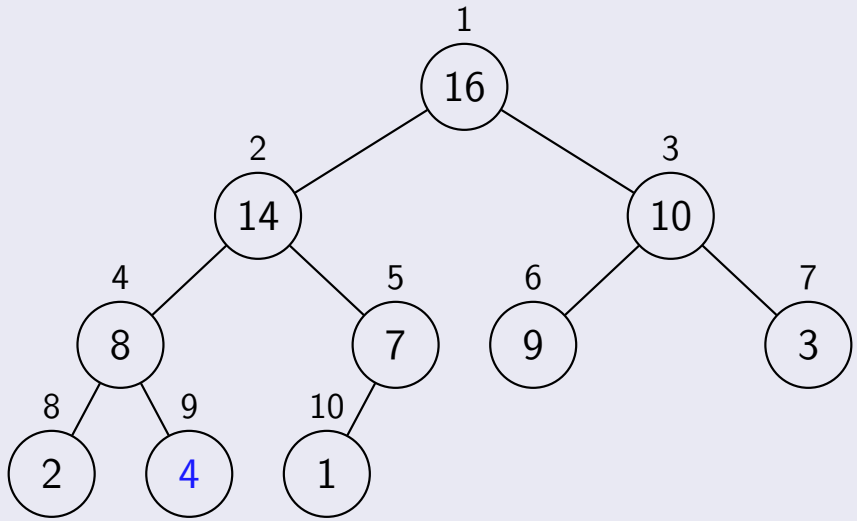


- E repetimos o procedimento enquanto houver violação da propriedade

Heaps

Mantendo a Propriedade do Heap

- Vimos que, no caso de alguma mudança no heap alterar sua propriedade:
  - Trocamos o elemento problemático de lugar com o maior filho

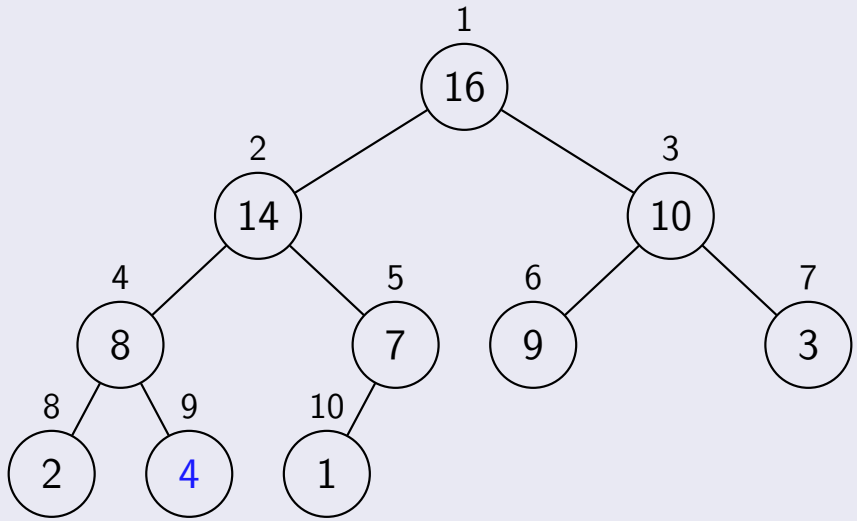


- E repetimos o procedimento enquanto houver violação da propriedade

Heaps

Mantendo a Propriedade do Heap

- Vimos que, no caso de alguma mudança no heap alterar sua propriedade:
  - Trocamos o elemento problemático de lugar com o maior filho



- E repetimos o procedimento enquanto houver violação da propriedade
  - Empurrando assim o elemento problemático para baixo no heap, até seu devido lugar

Heaps

Mantendo a Propriedade do Heap

- Vimos também o procedimento
  `void refazHeapMax(int A[], int i, int compHeap)`
  que fazia essa manutenção no heap em $O(\log n)$

Heaps

Mantendo a Propriedade do Heap

- Vimos também o procedimento
  `void refazHeapMax(int A[], int i, int compHeap)`
  que fazia essa manutenção no heap em $O(log\ n)$
- Como então construímos um heap?

Heaps

Mantendo a Propriedade do Heap

- Vimos também o procedimento
  `void refazHeapMax(int A[], int i, int compHeap)`
  que fazia essa manutenção no heap em $O(\log n)$
- Como então construímos um heap?
- Uma maneira é usar nosso `refazHeapMax` para converter um arranjo $A[1..n]$ em um heap máximo

Heaps

Construindo o Heap

- Note que, quando representamos o heap com o arranjo $A[1..n]$, as folhas ficam nos nós que vão de $\lfloor n/2 \rfloor + 1$ a n

Heaps

Construindo o Heap

- Note que, quando representamos o heap com o arranjo $A[1..n]$, as folhas ficam nos nós que vão de $\lfloor n/2 \rfloor + 1$ a n
  - Ou, alternativamente, nos elementos de índice $\lfloor n/2 \rfloor$ a $n-1$, se tomarmos o arranjo como $A[0..n-1]$

Heaps

Construindo o Heap

- Note que, quando representamos o heap com o arranjo $A[1..n]$, as folhas ficam nos nós que vão de $\lfloor n/2 \rfloor + 1$ a n
  - Ou, alternativamente, nos elementos de índice $\lfloor n/2 \rfloor$ a $n-1$, se tomarmos o arranjo como $A[0..n-1]$
- Mesmo? Suponha o arranjo $A[0..n-1]$

Heaps

Construindo o Heap

- Note que, quando representamos o heap com o arranjo $A[1..n]$, as folhas ficam nos nós que vão de $\lfloor n/2 \rfloor + 1$ a n
  - Ou, alternativamente, nos elementos de índice $\lfloor n/2 \rfloor$ a $n-1$, se tomarmos o arranjo como $A[0..n-1]$
- Mesmo? Suponha o arranjo $A[0..n-1]$
  - Imagine que $A[\lfloor n/2 \rfloor] \ldots A[n-1]$ não contenham apenas folhas

Heaps

Construindo o Heap

- Note que, quando representamos o heap com o arranjo $A[1..n]$, as folhas ficam nos nós que vão de $\lfloor n/2 \rfloor + 1$ a n
  - Ou, alternativamente, nos elementos de índice $\lfloor n/2 \rfloor$ a $n-1$, se tomarmos o arranjo como $A[0..n-1]$
- Mesmo? Suponha o arranjo $A[0..n-1]$
  - Imagine que $A[\lfloor n/2 \rfloor] \ldots A[n-1]$ não contenham apenas folhas
  - Isso significa que existe (pelo menos) um elemento $A[k]$, $\lfloor n/2 \rfloor \leq k \leq n-1$ que não é folha

Construindo o Heap

- O que significa que existe um $A[k']$, $\lfloor n/2 \rfloor \leq k' \leq n-1$ no heap que é filho desse elemento $A[k]$

Construindo o Heap

- O que significa que existe um $A[k']$, $\lfloor n/2 \rfloor \leq k' \leq n-1$ no heap que é filho desse elemento $A[k]$
- Mas, pela regra do heap, um filho de $A[k]$ estaria nas posições $k' = 2k+1$ ou $k' = 2k+2$

Construindo o Heap

- O que significa que existe um $A[k']$, $\lfloor n/2 \rfloor \leq k' \leq n-1$ no heap que é filho desse elemento $A[k]$
- Mas, pela regra do heap, um filho de $A[k]$ estaria nas posições $k' = 2k+1$ ou $k' = 2k+2$
- Como $k \geq \lfloor n/2 \rfloor$, então esse elemento estaria em $k' \geq 2n/2+1$ ou em $k' \geq 2n/2+2$

Construindo o Heap

- O que significa que existe um $A[k']$, $\lfloor n/2 \rfloor \leq k' \leq n-1$ no heap que é filho desse elemento $A[k]$
- Mas, pela regra do heap, um filho de $A[k]$ estaria nas posições $k' = 2k+1$ ou $k' = 2k+2$
- Como $k \geq \lfloor n/2 \rfloor$, então esse elemento estaria em $k' \geq 2n/2+1$ ou em $k' \geq 2n/2+2$
- Ou seja, $k' \geq n+1$ ou em $k' \geq n+2$. Fora, portanto, do arranjo

Construindo o Heap

- O que significa que existe um $A[k']$, $\lfloor n/2 \rfloor \leq k' \leq n-1$ no heap que é filho desse elemento $A[k]$
- Mas, pela regra do heap, um filho de $A[k]$ estaria nas posições $k' = 2k+1$ ou $k' = 2k+2$
- Como $k \geq \lfloor n/2 \rfloor$, então esse elemento estaria em $k' \geq 2n/2+1$ ou em $k' \geq 2n/2+2$
- Ou seja, $k' \geq n+1$ ou em $k' \geq n+2$. Fora, portanto, do arranjo
- Então sim, as folhas ficam nos nós que vão de $\lfloor n/2 \rfloor + 1$ a n

Heaps

Construindo o Heap

- As folhas de um heap podem ser vistas como heaps de 1 único elemento

Heaps

Construindo o Heap

- As folhas de um heap podem ser vistas como heaps de 1 único elemento
  - Podemos então considerar cada elemento $A[k]$, $\lfloor n/2 \rfloor \leq k \leq n-1$ como um heap

Heaps

Construindo o Heap

- As folhas de um heap podem ser vistas como heaps de 1 único elemento
  - Podemos então considerar cada elemento $A[k]$, $\lfloor n/2 \rfloor \leq k \leq n-1$ como um heap
  - E a partir desses, construir o resto do heap, numa estratégia bottom-up

Heaps

Construindo o Heap

- As folhas de um heap podem ser vistas como heaps de 1 único elemento
  - Podemos então considerar cada elemento $A[k]$, $\lfloor n/2 \rfloor \leq k \leq n-1$ como um heap
  - E a partir desses, construir o resto do heap, numa estratégia bottom-up
- Como?

Heaps

Construindo o Heap

- Iniciamos com esses heaps de 1 elemento ($A[\lfloor n/2 \rfloor]$ até $A[n-1]$)

Heaps

Construindo o Heap

- Iniciamos com esses heaps de 1 elemento ($A[\lfloor n/2 \rfloor]$ até $A[n-1]$)
- Chamamos então `refazHeapMax` a partir da posição $\lfloor n/2 \rfloor - 1$ descendo até 0

Heaps

Construindo o Heap

- Iniciamos com esses heaps de 1 elemento ($A[\lfloor n/2 \rfloor]$ até $A[n-1]$)
- Chamamos então `refazHeapMax` a partir da posição $\lfloor n/2 \rfloor - 1$ descendo até 0
  - Como se tivéssemos adicionado um elemento ao heap, em seu início, e agora tivéssemos que manter sua propriedade

Heaps

Construindo o Heap

- Iniciamos com esses heaps de 1 elemento ($A[\lfloor n/2 \rfloor]$ até $A[n-1]$)
- Chamamos então `refazHeapMax` a partir da posição $\lfloor n/2 \rfloor - 1$ descendo até 0
  - Como se tivéssemos adicionado um elemento ao heap, em seu início, e agora tivéssemos que manter sua propriedade
- Repetimos então a partir da posição $\lfloor n/2 \rfloor - 2$ etc, até a posição 0

Heaps

Construindo o Heap

- Iniciamos com esses heaps de 1 elemento ($A[\lfloor n/2 \rfloor]$ até $A[n-1]$)
- Chamamos então `refazHeapMax` a partir da posição $\lfloor n/2 \rfloor - 1$ descendo até 0
  - Como se tivéssemos adicionado um elemento ao heap, em seu início, e agora tivéssemos que manter sua propriedade
- Repetimos então a partir da posição $\lfloor n/2 \rfloor - 2$ etc, até a posição 0
  - Ou seja, um a um adicionamos os demais elementos

Heaps

Construindo o Heap

```
void constroiHeapMax(int A[], int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A, i, n);
}
```

Heaps

Construindo o Heap

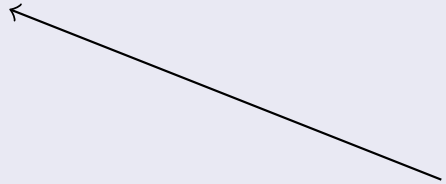
```
void constroiHeapMax(int A[], int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A, i, n);
}
```

Para cada elemento, da metade até a primeira posição

Heaps

Construindo o Heap

```
void constroiHeapMax(int A[], int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A, i, n);
}
```



Garanto a propriedade do heap a partir desse elemento

Heaps

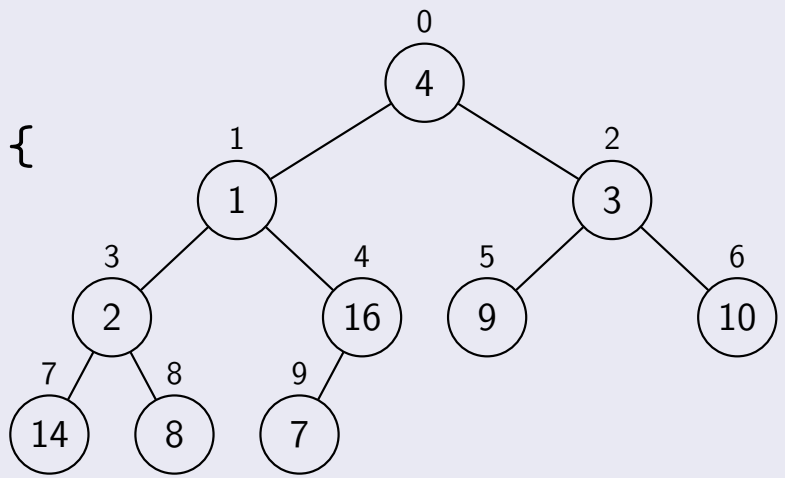
Construindo o Heap

```
void constroiHeapMax(int A[], int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A,i,n);
}
```

Heaps

Construindo o Heap

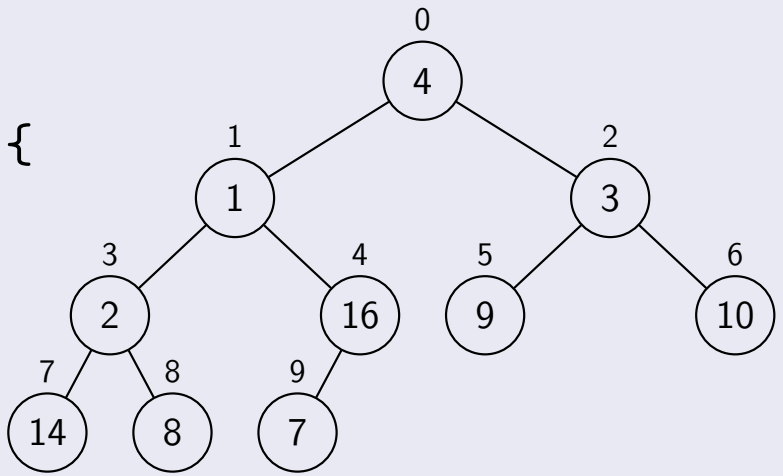
```
void constroiHeapMax(int A[], int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A,i,n);
}
```



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

Construindo o Heap

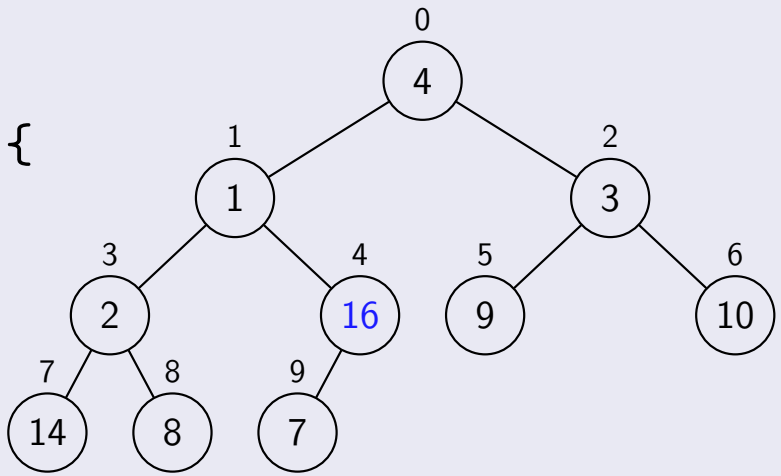
```
void constroiHeapMax(int A[], int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A,i,n);
}
```



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

Construindo o Heap

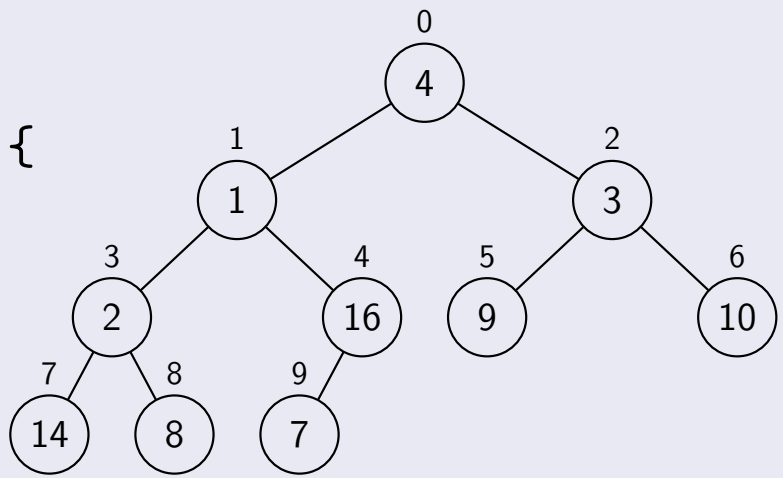
```
void constroiHeapMax(int A[], int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A,i,n);
}
```



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

Construindo o Heap

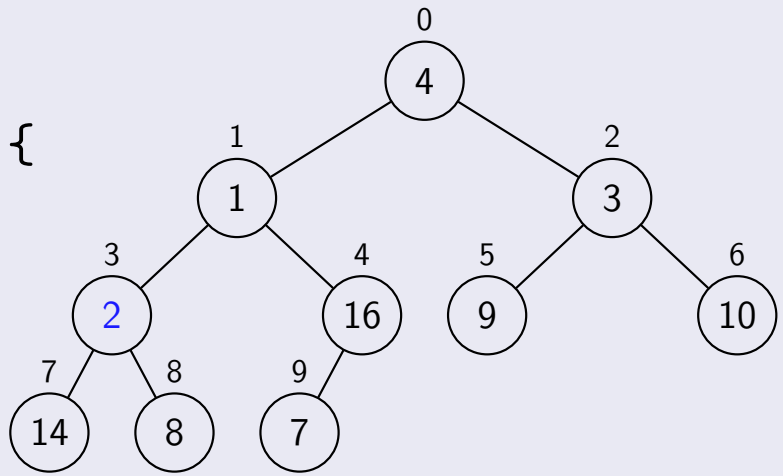
```
void constroiHeapMax(int A[], int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A,i,n);
}
```



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

Construindo o Heap

```
void constroiHeapMax(int A[], int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A,i,n);
}
```

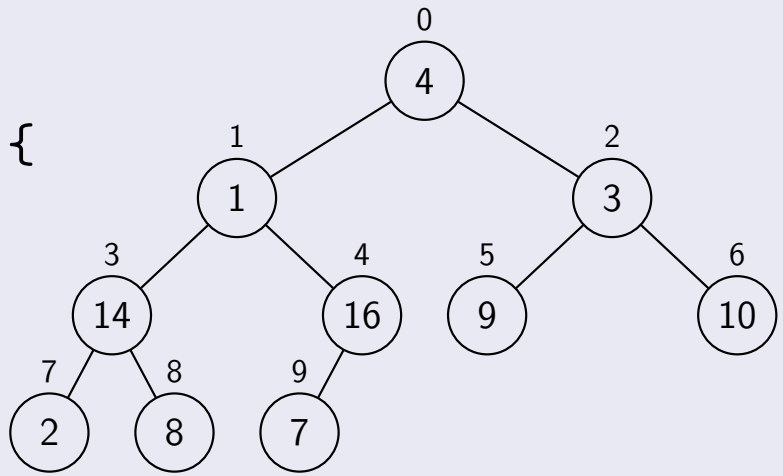


4	1	3	2	16	9	10	14	8	7
0	1	2	3	4	5	6	7	8	9

i

Construindo o Heap

```
void constroiHeapMax(int A[], int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A,i,n);
}
```



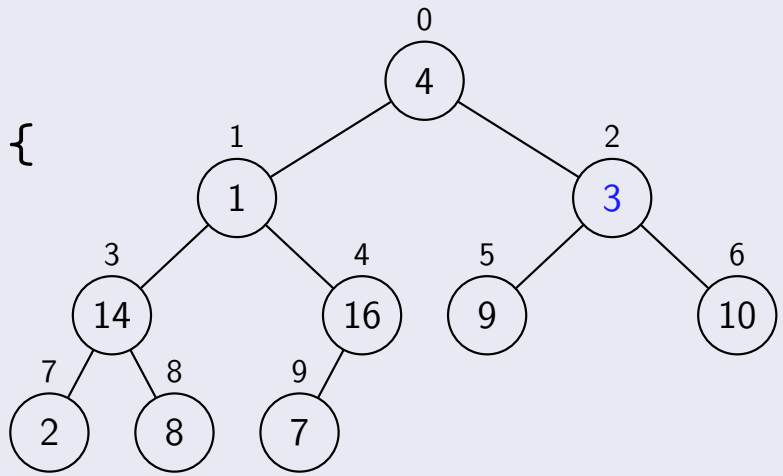
4	1	3	14	16	9	10	2	8	7
0	1	2	3	4	5	6	7	8	9

i

Heaps

Construindo o Heap

```
void constroiHeapMax(int A[], int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A,i,n);
}
```

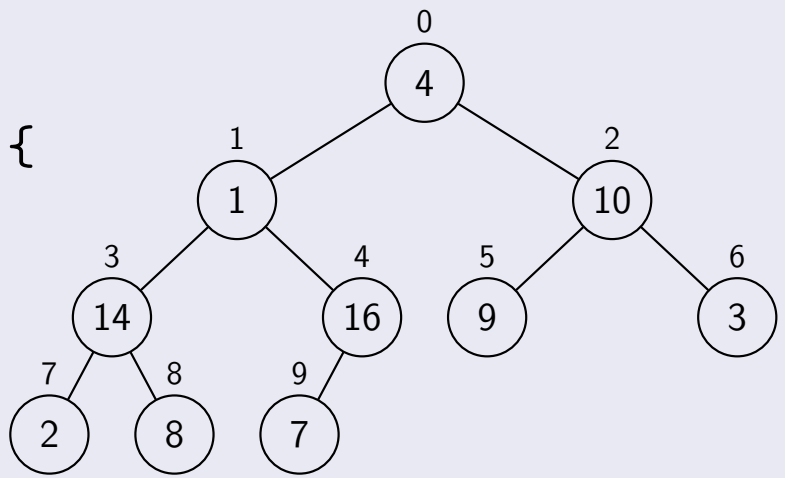


4	1	3	14	16	9	10	2	8	7
0	1	2	3	4	5	6	7	8	9
		i							

Heaps

Construindo o Heap

```
void constroiHeapMax(int A[], int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A,i,n);
}
```

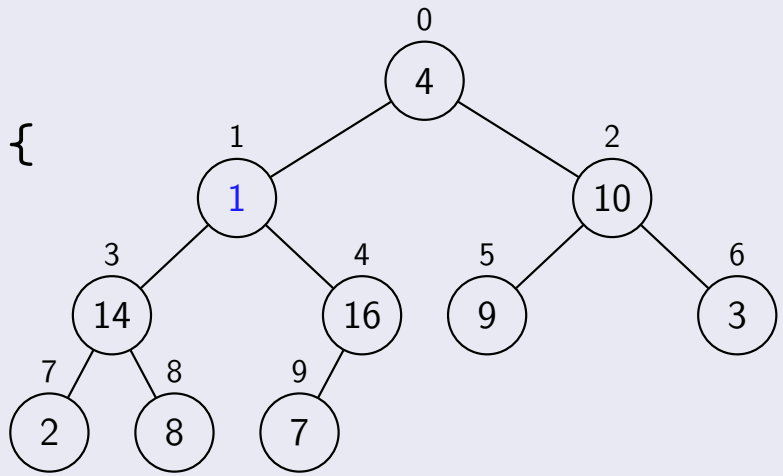


4	1	10	14	16	9	3	2	8	7
0	1	2	3	4	5	6	7	8	9
		i							

Heaps

Construindo o Heap

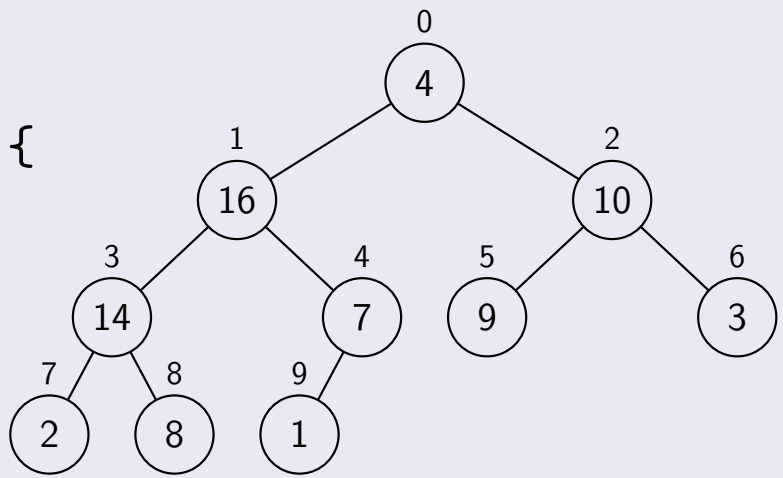
```
void constroiHeapMax(int A[], int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A,i,n);
}
```



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

Construindo o Heap

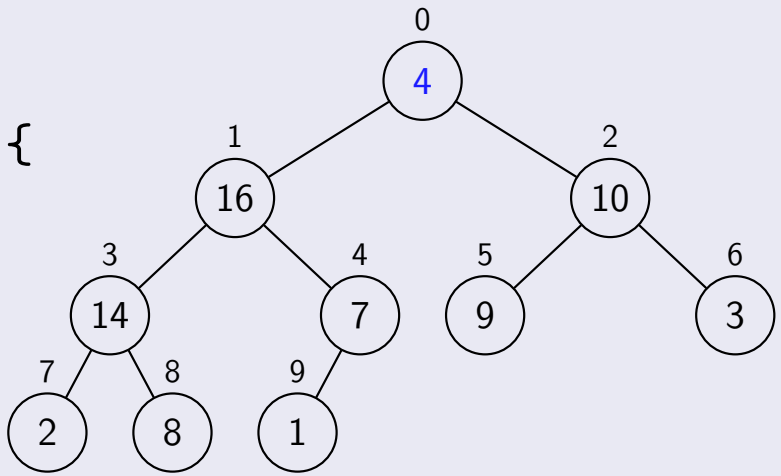
```
void constroiHeapMax(int A[], int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A,i,n);
}
```



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

Construindo o Heap

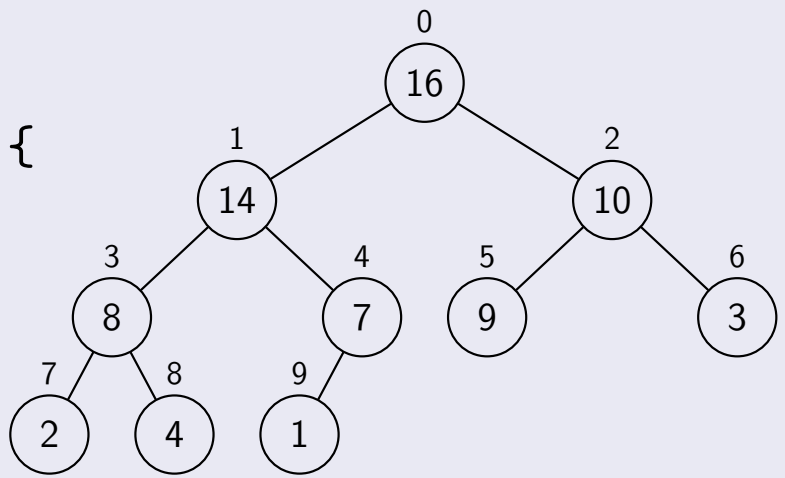
```
void constroiHeapMax(int A[], int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A,i,n);
}
```



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

Construindo o Heap

```
void constroiHeapMax(int A[], int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A,i,n);
}
```



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

Heaps

Construindo o Heap

- E qual a complexidade desse código?

```
void constroiHeapMax(int A[],
                     int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A,i,compHeap);
}
```

Heaps

Construindo o Heap

- E qual a complexidade desse código?
  - $O(log\ n)$

```
void constroiHeapMax(int A[],
                     int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A,i,compHeap);
}
```

Heaps

Construindo o Heap

- E qual a complexidade desse código?
  - $O(log\ n) \times O(n)$

```
void constroiHeapMax(int A[],
                     int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A,i,compHeap);
}
```

Heaps

Construindo o Heap

- E qual a complexidade desse código?
  - $O(log\ n) \times O(n)$
- Portanto, $O(n\ log\ n)$

```
void constroiHeapMax(int A[],
                     int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A,i,compHeap);
}
```

Heaps

Construindo o Heap

- E qual a complexidade desse código?
  - $O(log\ n) \times O(n)$
- Portanto, $O(n\ log\ n)$
- Mas isso pode melhorar...

```
void constroiHeapMax(int A[],
                     int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A,i,compHeap);
}
```

Heaps

Construindo o Heap

- E qual a complexidade desse código?
  - $O(log\ n) \times O(n)$
- Portanto, $O(n\ log\ n)$
- Mas isso pode melhorar...
- De fato, pode-se demonstrar que um limite mais apertado seria $O(n)$

```
void constroiHeapMax(int A[],
                     int n) {
  int i;
  for(i = n/2 - 1; i >= 0; i--)
    refazHeapMax(A,i,compHeap);
}
```

Heaps

Construindo o Heap [versão recursiva]

```
void constroiHeapMaxRec(int A[], int i, int n) {
  if (i<n/2){
    constroiHeapMaxRec(A, esquerda(i), n);
    constroiHeapMaxRec(A, direita(i), n);
    refazHeapMax(A, i, n);
  }
}
```

Construindo o Heap [versão recursiva]

```
void constroiHeapMaxRec(int A[], int i, int n) {
  if (i<n/2){
    constroiHeapMaxRec(A, esquerda(i), n);
    constroiHeapMaxRec(A, direita(i), n);
    refazHeapMax(A, i, n);
  }
}
```

$$T(n) = \begin{cases} & \text{se } n \leq 1 \\ & \text{para } n \geq 2 \end{cases}$$

Heaps

Construindo o Heap [versão recursiva]

```
void constroiHeapMaxRec(int A[], int i, int n) {
  if (i<n/2){
    constroiHeapMaxRec(A, esquerda(i), n);
    constroiHeapMaxRec(A, direita(i), n);
    refazHeapMax(A, i, n);
  }
}
```

$$T(n) = \begin{cases} O(1) & \text{se } n \leq 1 \\ & \text{para } n \geq 2 \end{cases}$$

Construindo o Heap [versão recursiva]

```
void constroiHeapMaxRec(int A[], int i, int n) {
  if (i<n/2){
    constroiHeapMaxRec(A, esquerda(i), n);
    constroiHeapMaxRec(A, direita(i), n);
    refazHeapMax(A, i, n);
  }
}
```

$$T(n) = \begin{cases} O(1) & \text{se } n \leq 1 \\ T(2n/3) + T(n/3) & \text{para } n \geq 2 \end{cases}$$

Construindo o Heap [versão recursiva]

```
void constroiHeapMaxRec(int A[], int i, int n) {
  if (i<n/2){
    constroiHeapMaxRec(A, esquerda(i), n);
    constroiHeapMaxRec(A, direita(i), n);
    refazHeapMax(A, i, n);
  }
}
```

$$T(n) = \begin{cases} O(1) & \text{se } n \leq 1 \\ T(2n/3) + T(n/3) + O(log\ n) & \text{para } n \geq 2 \end{cases}$$

Heaps

Construindo o Heap [versão recursiva]

```
void constroiHeapMaxRec(int A[], int i, int n) {
  if (i<n/2){
    constroiHeapMaxRec(A, esquerda(i), n);
    constroiHeapMaxRec(A, direita(i), n);
    refazHeapMax(A, i, n);
  }
}
```

$$T(n) = \begin{cases} O(1) & \text{se } n \leq 1 \\ T(2n/3) + T(n/3) + O(log\ n) & \text{para } n \geq 2 \end{cases}$$

Se considerarmos uma árvore binária completa:

$$T(n) = \begin{cases} O(1) & \text{se } n \leq 1 \\ 2 * T(n/2) + O(log\ n) & \text{para } n \geq 2 \end{cases}$$

HeapSort

Projeto por Indução Fraca

Quarta Alternativa

Projeto por Indução Fraca

Quarta Alternativa

- **Base**: $n = 1$. Um único elemento está ordenado

Projeto por Indução Fraca

Quarta Alternativa

- **Base**: $n = 1$. Um único elemento está ordenado
- **H.I.**: Sei ordenar um conjunto de $n - 1 \geq 1$ valores

Projeto por Indução Fraca

Quarta Alternativa

- **Base**: $n = 1$. Um único elemento está ordenado
- **H.I.**: Sei ordenar um conjunto de $n - 1 \geq 1$ valores
- **Passo**:

Projeto por Indução Fraca

Quarta Alternativa

- **Base**: $n = 1$. Um único elemento está ordenado
- **H.I.**: Sei ordenar um conjunto de $n - 1 \geq 1$ valores
- **Passo**:
  - Seja S um conjunto de $n \geq 2$ valores, organizados na forma de um heap máximo

Projeto por Indução Fraca

Quarta Alternativa

- **Base**: $n = 1$. Um único elemento está ordenado
- **H.I.**: Sei ordenar um conjunto de $n - 1 \geq 1$ valores
- **Passo**:
  - Seja S um conjunto de $n \geq 2$ valores, organizados na forma de um heap máximo
  - Sendo S um heap máximo, o maior elemento está em S_1. Trocamos esse elemento por S_n, restaurando o heap em $S - S_n$. Por hipótese de indução, sei ordenar a sequência $S - S_n$, e assim obtemos S ordenado

Projeto por Indução Fraca

Quarta Alternativa

- **Base**: $n = 1$. Um único elemento está ordenado
- **H.I.**: Sei ordenar um conjunto de $n - 1 \geq 1$ valores
- **Passo**:
  - Seja S um conjunto de $n \geq 2$ valores, organizados na forma de um heap máximo
  - Sendo S um heap máximo, o maior elemento está em S_1. Trocamos esse elemento por S_n, restaurando o heap em $S - S_n$. Por hipótese de indução, sei ordenar a sequência $S - S_n$, e assim obtemos S ordenado
- (*HeapSort*)

HeapSort

Projeto

- O projeto do Heapsort é essencialmente o mesmo da ordenação por Seleção

HeapSort

Projeto

- O projeto do Heapsort é essencialmente o mesmo da ordenação por Seleção
  - Selecionamos e posicionamos o maior (ou menor) elemento do conjunto

HeapSort

Projeto

- O projeto do Heapsort é essencialmente o mesmo da ordenação por Seleção
  - Selecionamos e posicionamos o maior (ou menor) elemento do conjunto
  - Aplicamos a hipótese de indução para ordenar os elementos restantes

HeapSort

Projeto

- O projeto do Heapsort é essencialmente o mesmo da ordenação por Seleção
  - Selecionamos e posicionamos o maior (ou menor) elemento do conjunto
  - Aplicamos a hipótese de indução para ordenar os elementos restantes
- A diferença é que no HeapSort utilizamos um heap para selecionar o maior (ou menor) elemento de forma eficiente

HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

Transforma o arranjo em um heap

HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

Coloca o maior elemento ao final do arranjo a ser ordenado

HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

Verifica se a mudança feita não violou a propriedade no heap restante, de $n - 1$ elementos

HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

Repete a operação para o heap de $n-1$ elementos ($T(n-1)$)

HeapSort

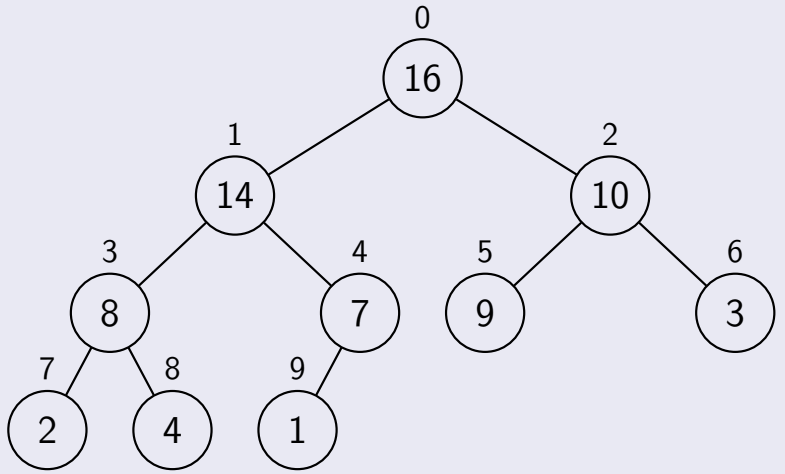
Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

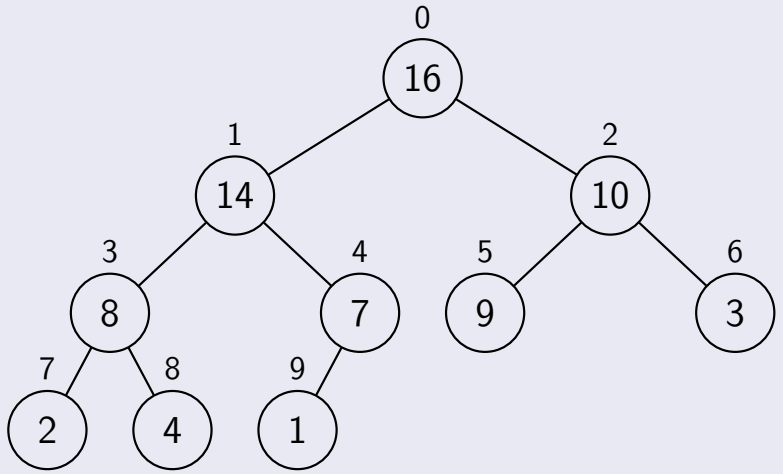


16	14	10	8	7	9	3	2	4	1
0	1	2	3	4	5	6	7	8	9

HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

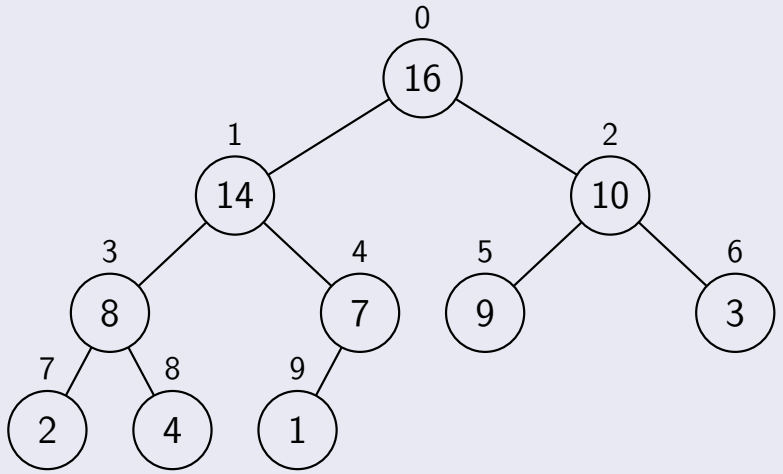


16	14	10	8	7	9	3	2	4	1
0	1	2	3	4	5	6	7	8	9

HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

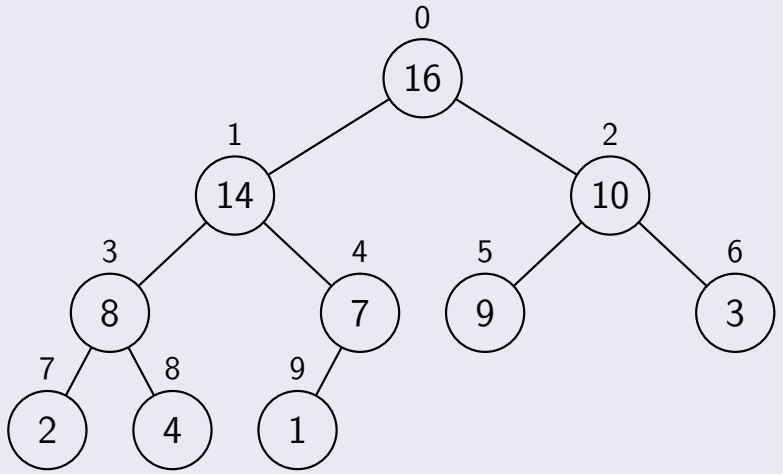


16	14	10	8	7	9	3	2	4	1
0	1	2	3	4	5	6	7	8	9

HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

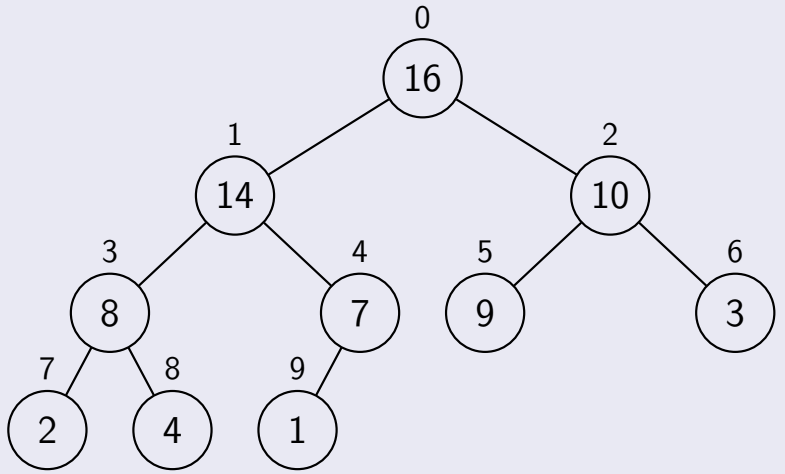


16	14	10	8	7	9	3	2	4	1
0	1	2	3	4	5	6	7	8	9

HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



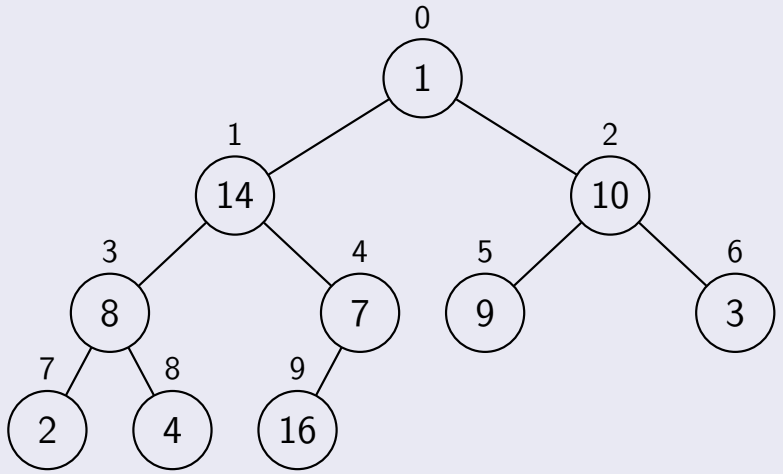
16	14	10	8	7	9	3	2	4	1
0	1	2	3	4	5	6	7	8	9

i

HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

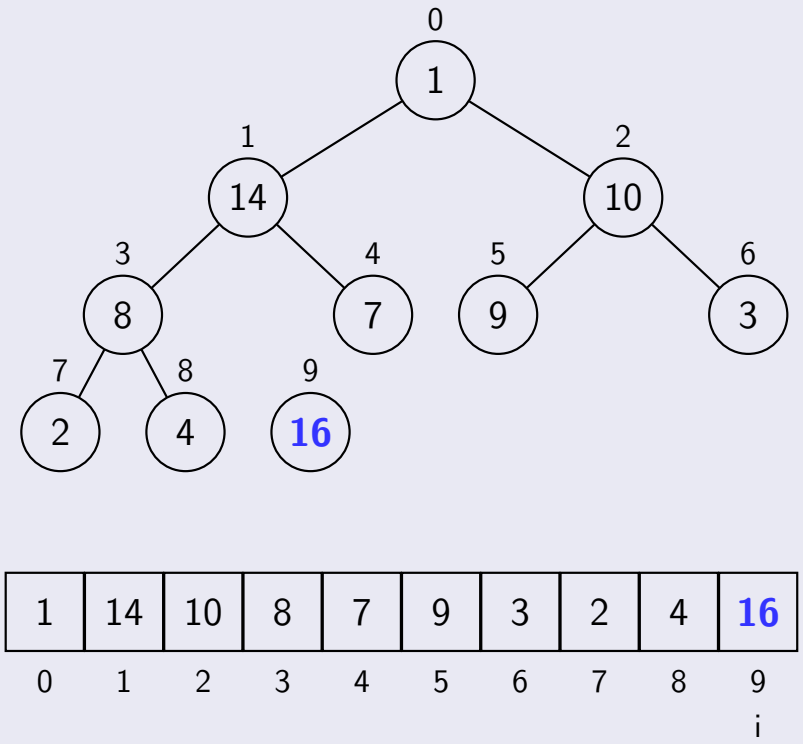


1	14	10	8	7	9	3	2	4	16
0	1	2	3	4	5	6	7	8	9
									i

HeapSort

Código

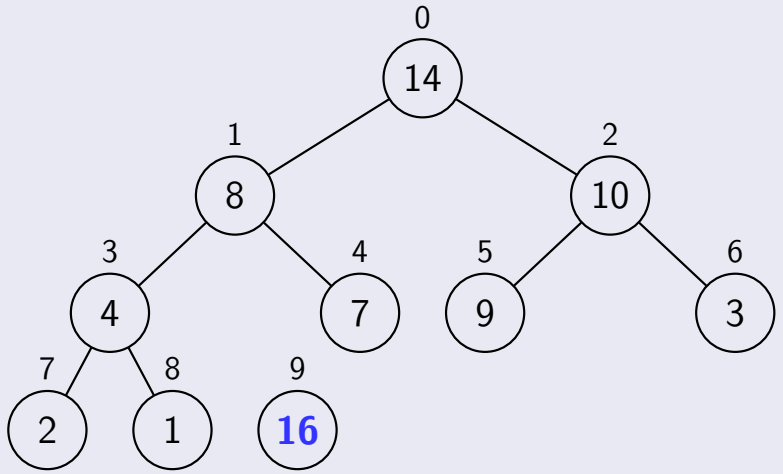
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

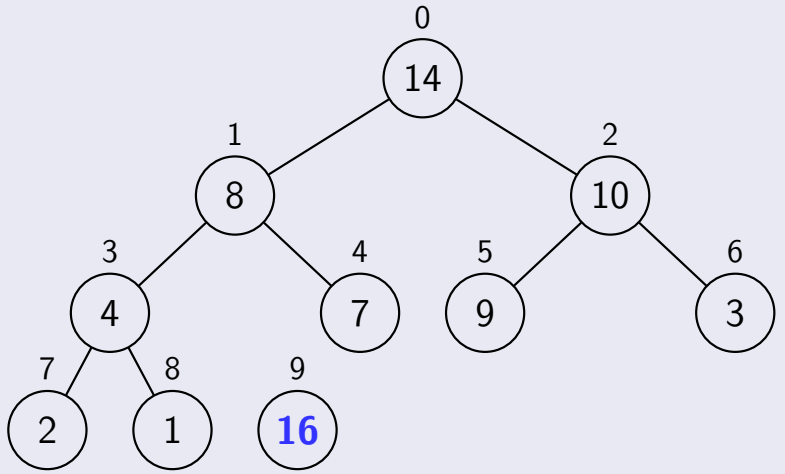


14	8	10	4	7	9	3	2	1	**16**
0	1	2	3	4	5	6	7	8	9
									i

HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

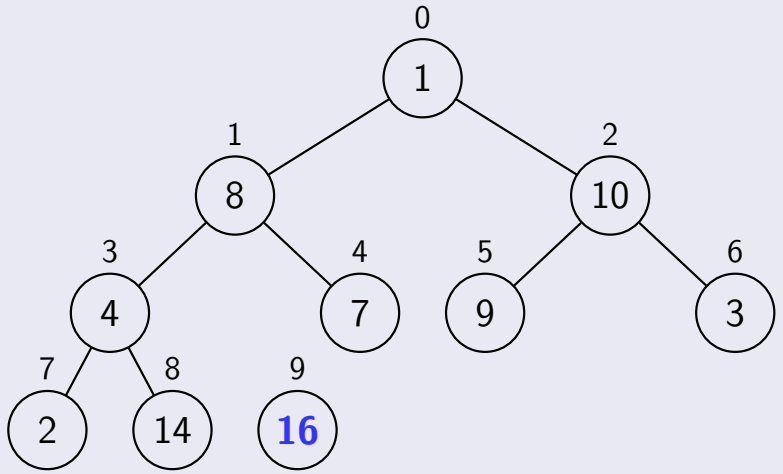


14	8	10	4	7	9	3	2	1	16
0	1	2	3	4	5	6	7	8	9
								i	

HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

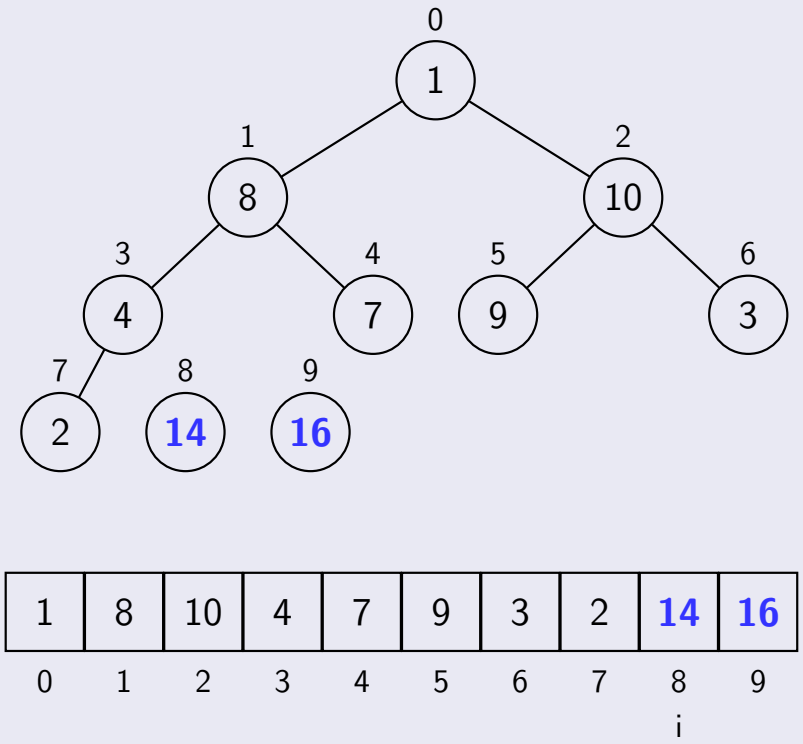


1	8	10	4	7	9	3	2	14	16
0	1	2	3	4	5	6	7	8	9
								i	

HeapSort

Código

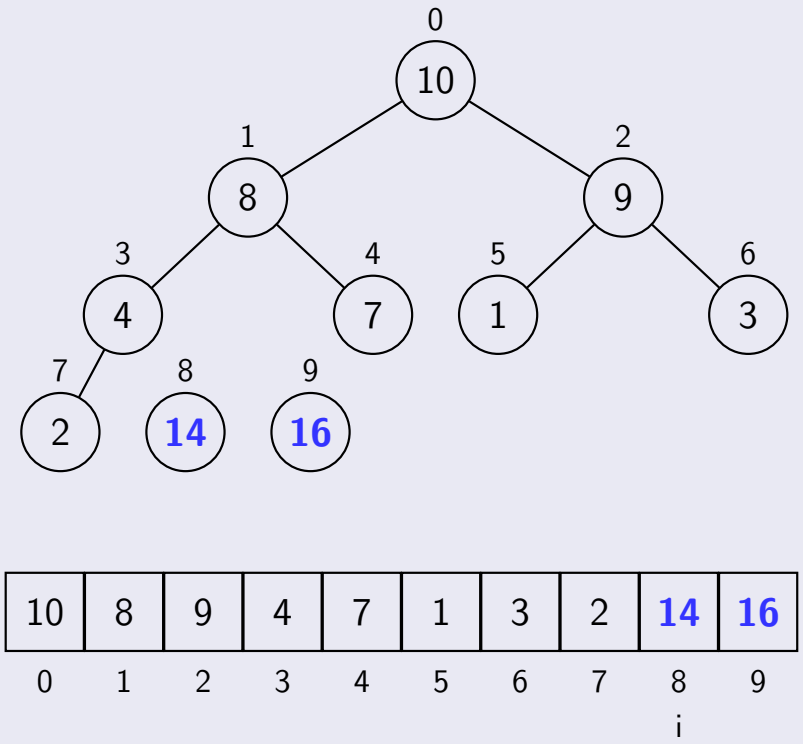
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

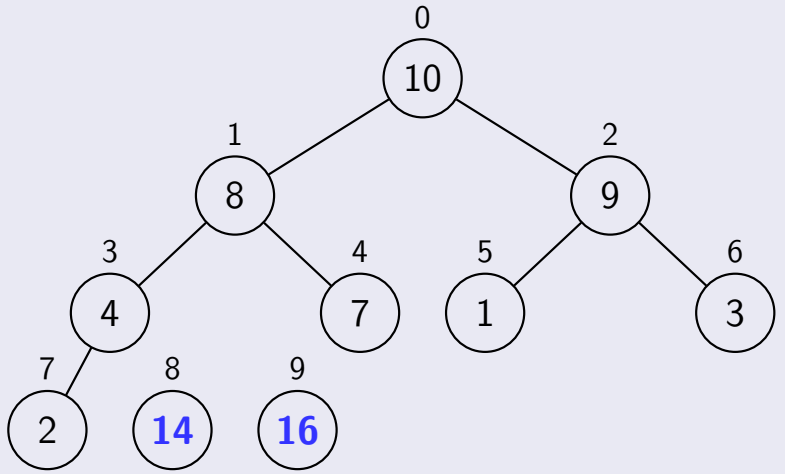
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

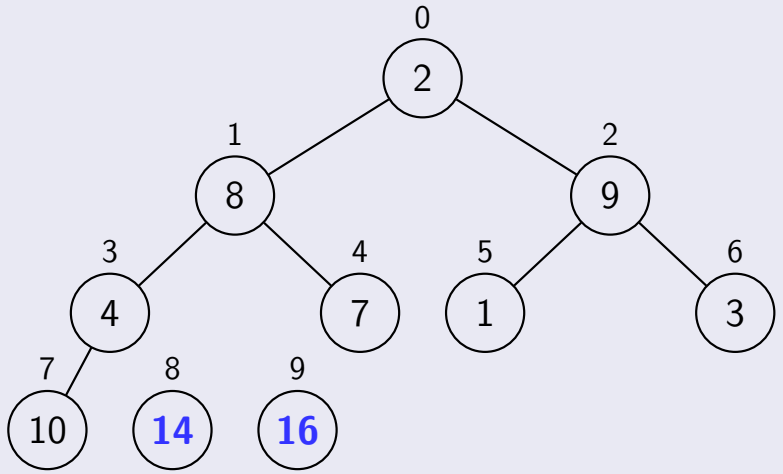


10	8	9	4	7	1	3	2	14	16
0	1	2	3	4	5	6	7	8	9
							i		

HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

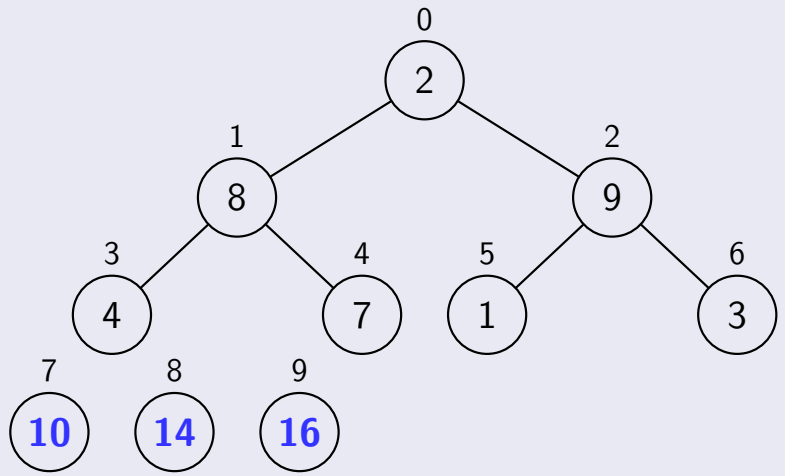


2	8	9	4	7	1	3	10	14	16
0	1	2	3	4	5	6	7	8	9
							i		

HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

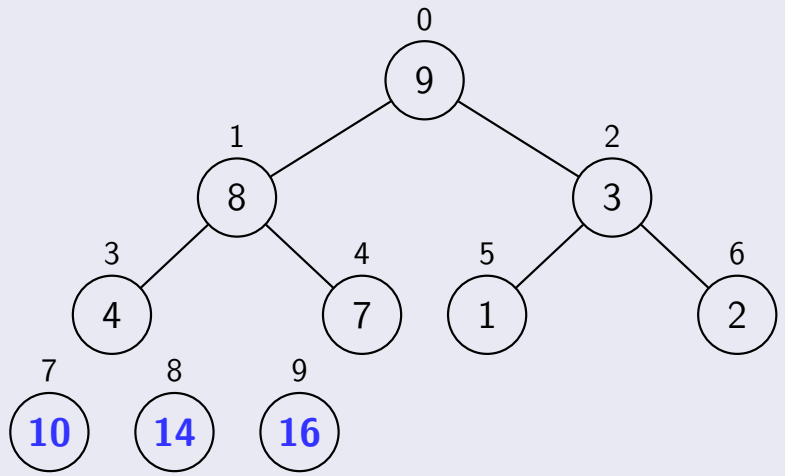


2	8	9	4	7	1	3	10	14	16
0	1	2	3	4	5	6	7	8	9
							i		

HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

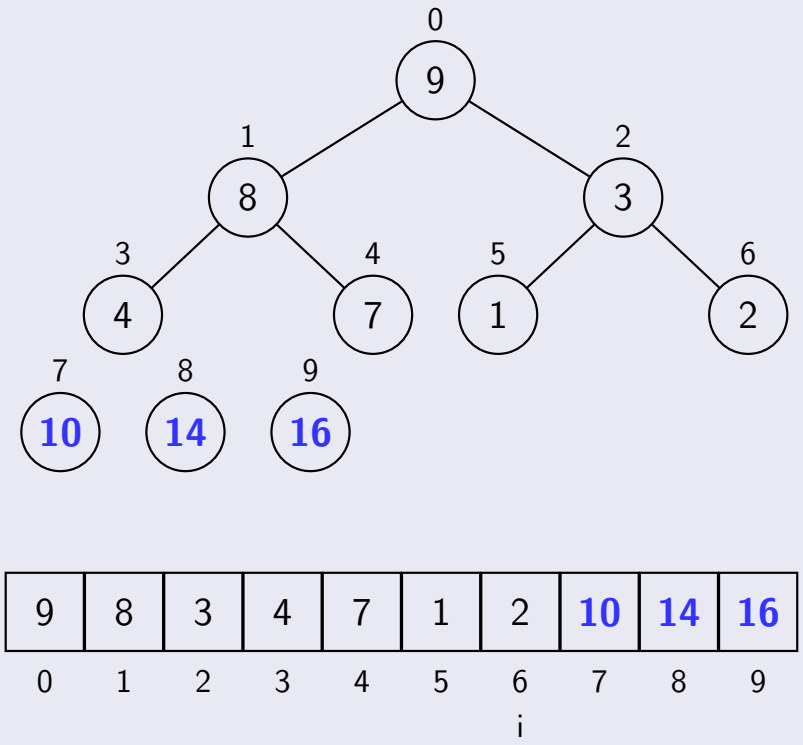


9	8	3	4	7	1	2	10	14	16
0	1	2	3	4	5	6	7	8	9
							i		

HeapSort

Código

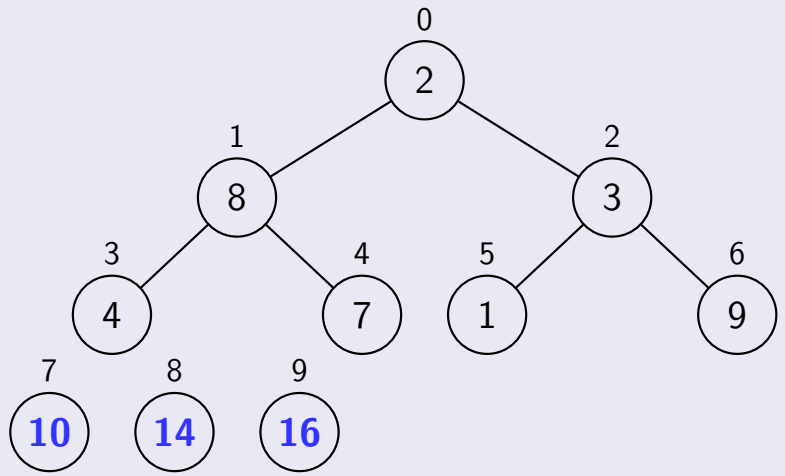
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

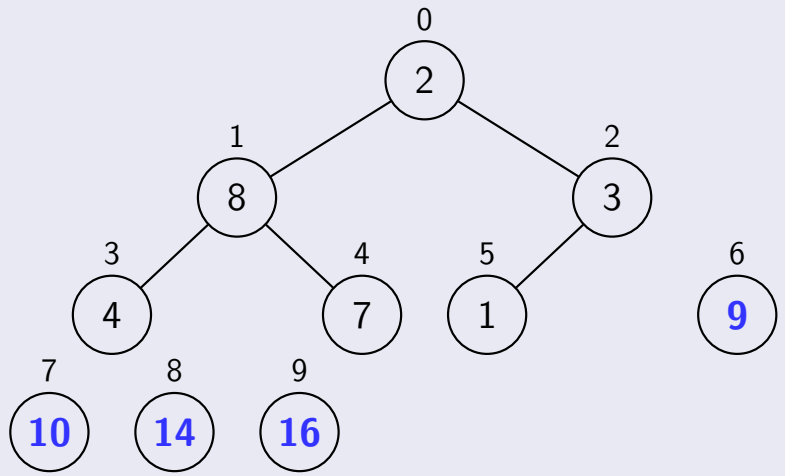


2	8	3	4	7	1	9	10	14	16
0	1	2	3	4	5	6	7	8	9
						i			

HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

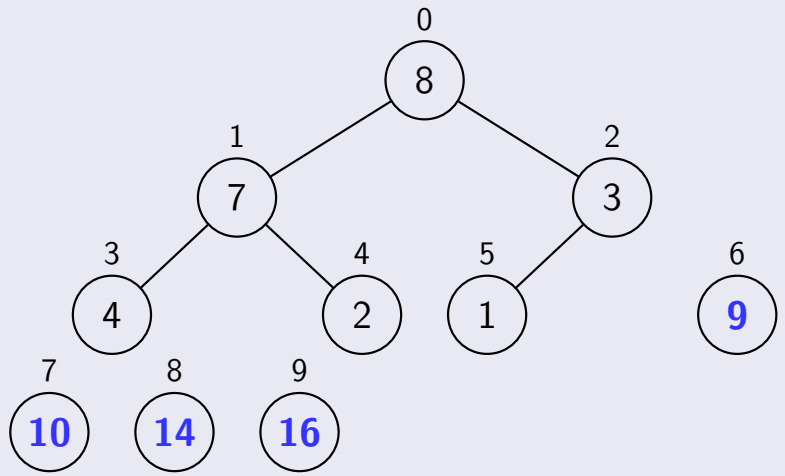


2	8	3	4	7	1	9	10	14	16
0	1	2	3	4	5	6	7	8	9
						i			

HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



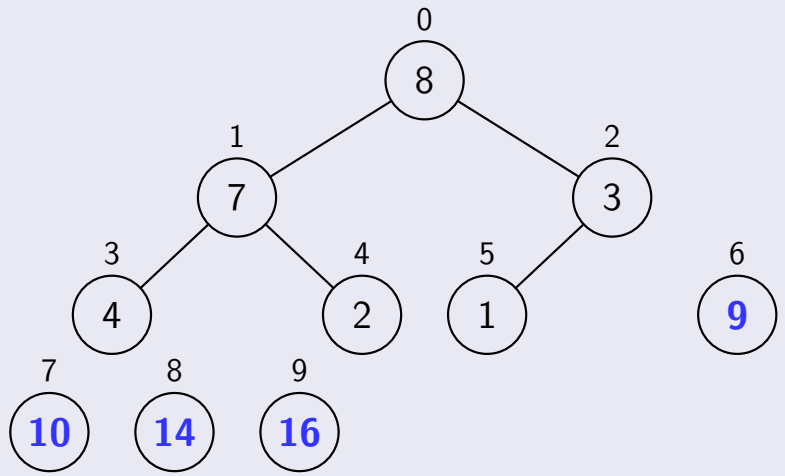
8	7	3	4	2	1	9	10	14	16
0	1	2	3	4	5	6	7	8	9

i

HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

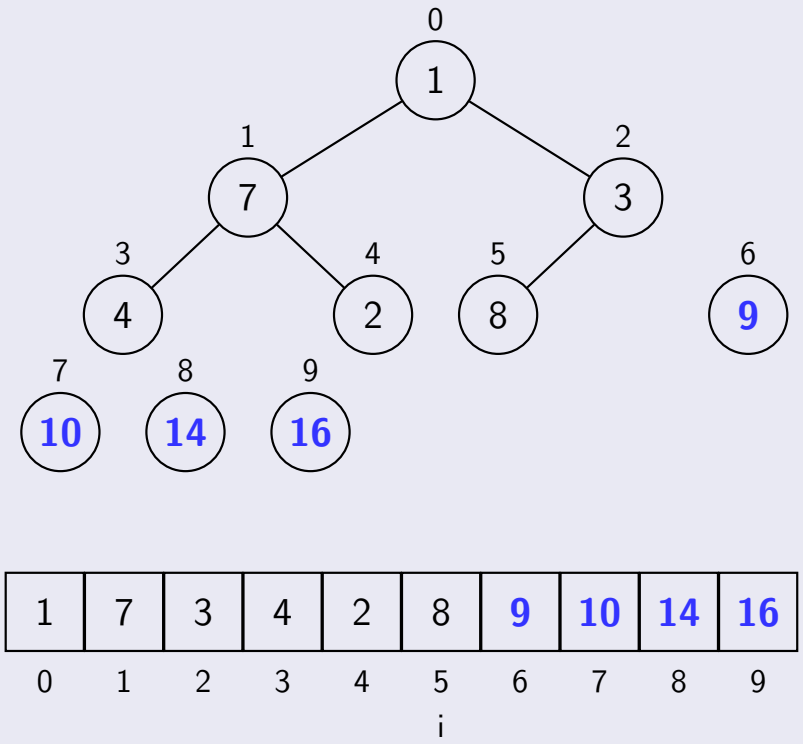


8	7	3	4	2	1	9	10	14	16
0	1	2	3	4	5	6	7	8	9
					i				

HeapSort

Código

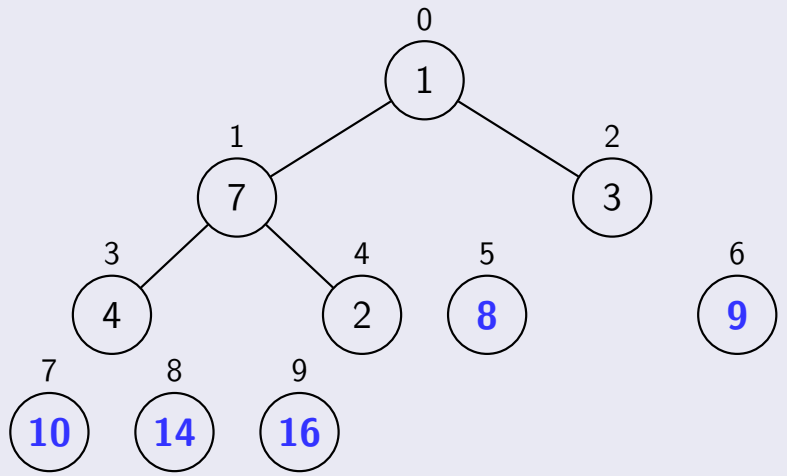
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



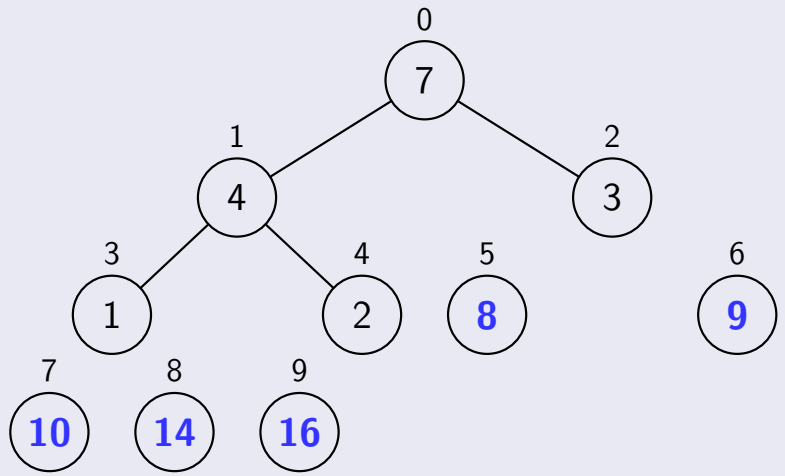
1	7	3	4	2	8	9	10	14	16
0	1	2	3	4	5	6	7	8	9

i

HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



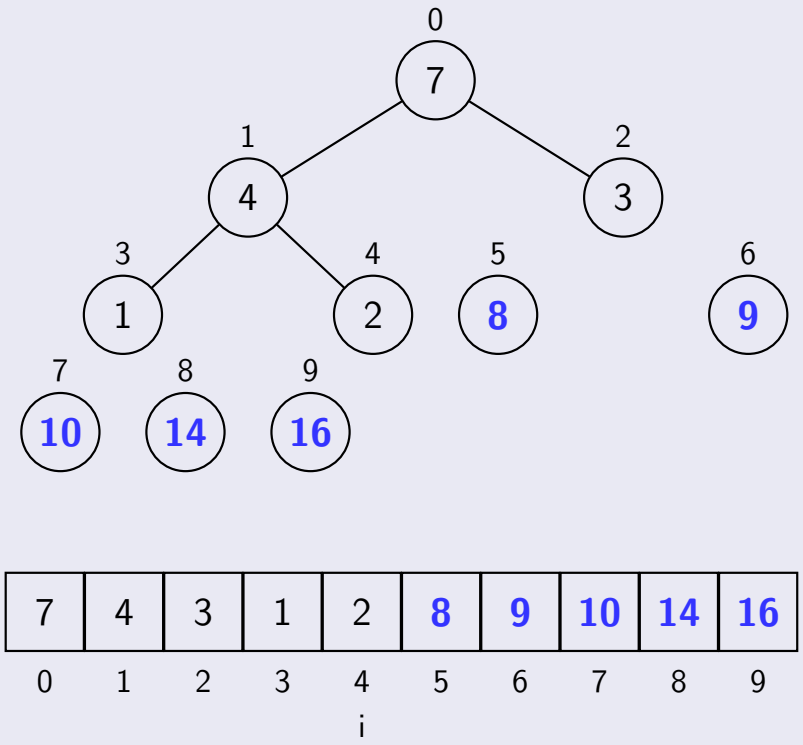
7	4	3	1	2	8	9	10	14	16
0	1	2	3	4	5	6	7	8	9

i

HeapSort

Código

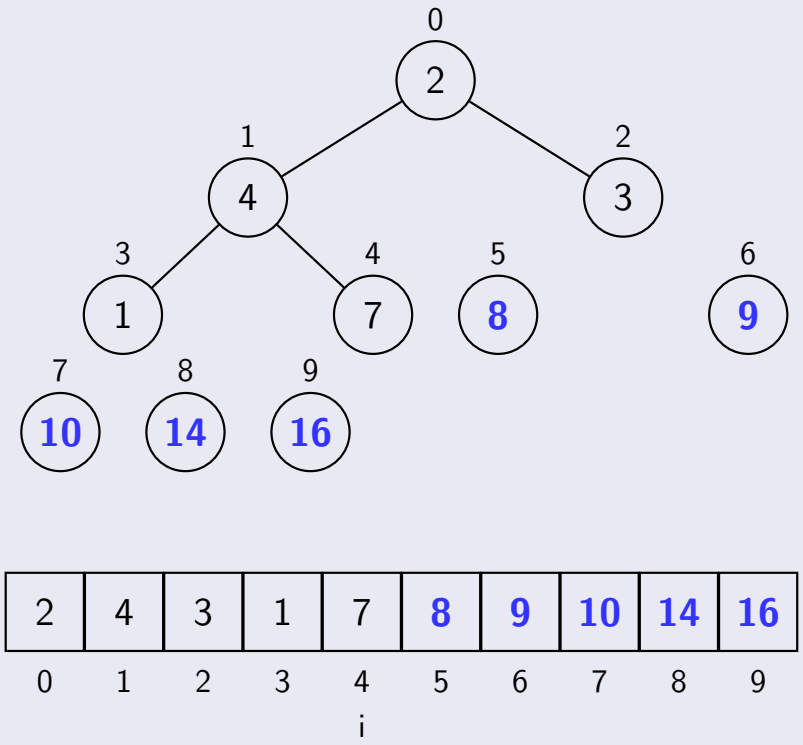
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

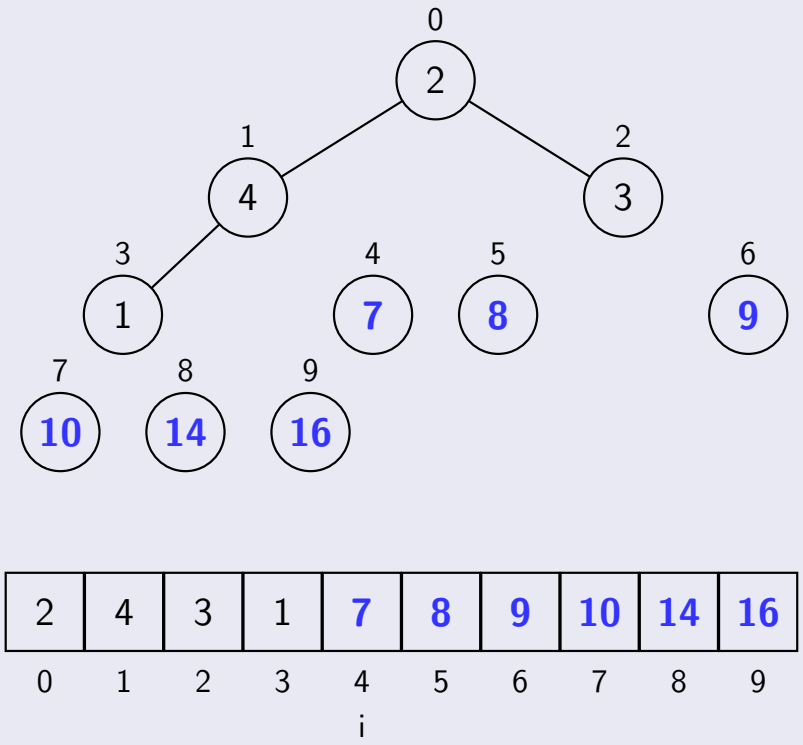
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

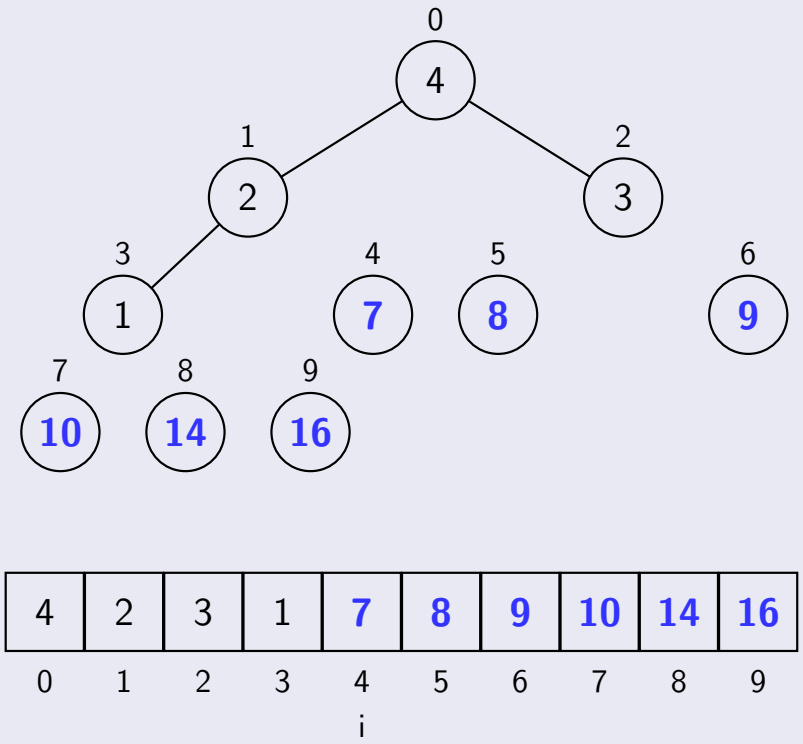
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

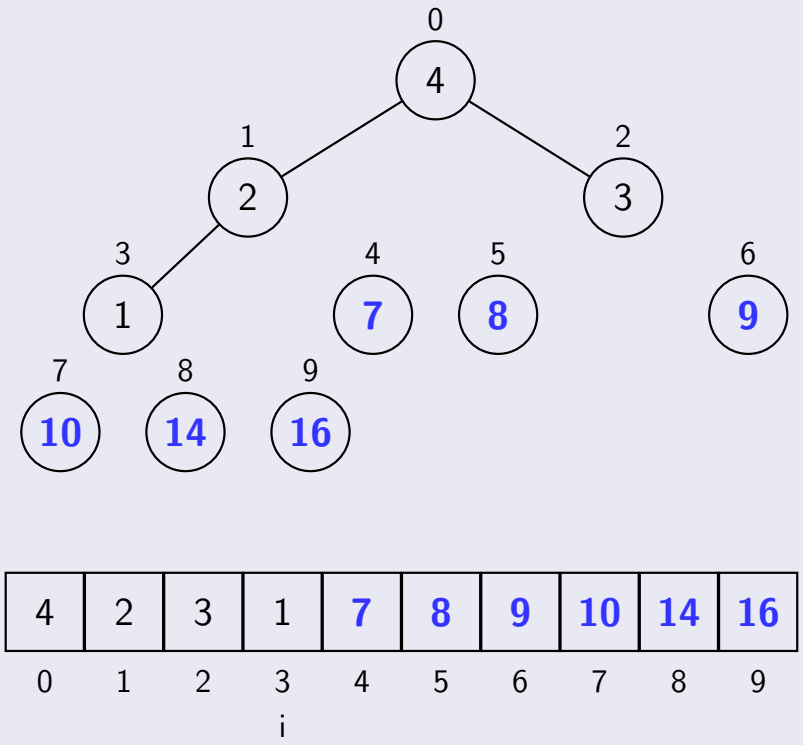
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

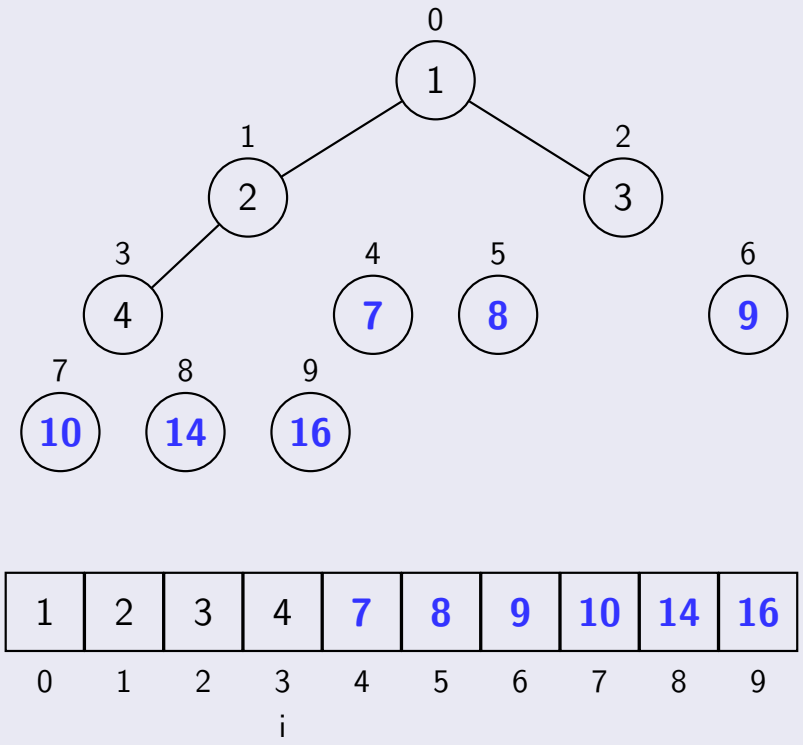
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

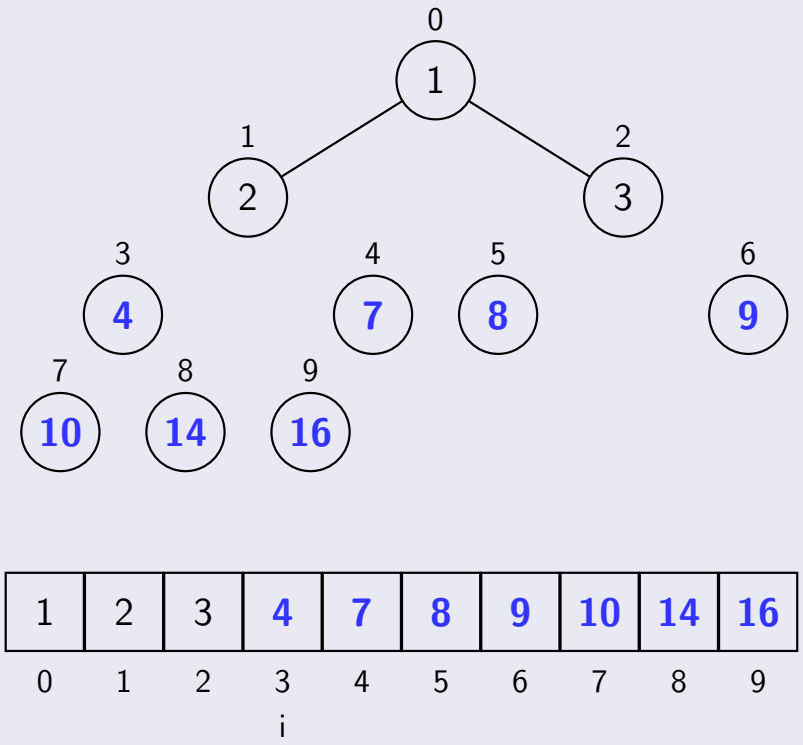
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

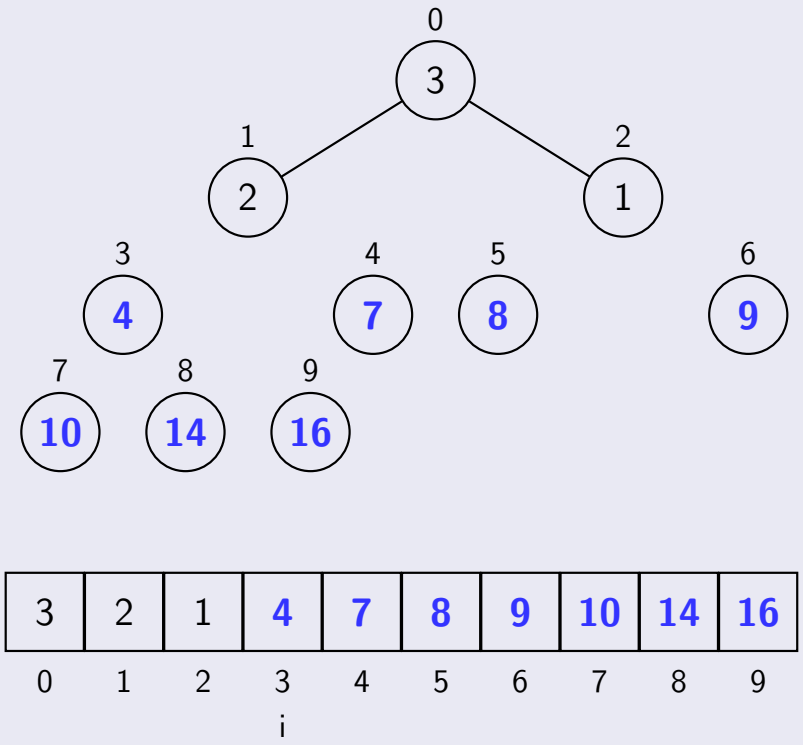
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

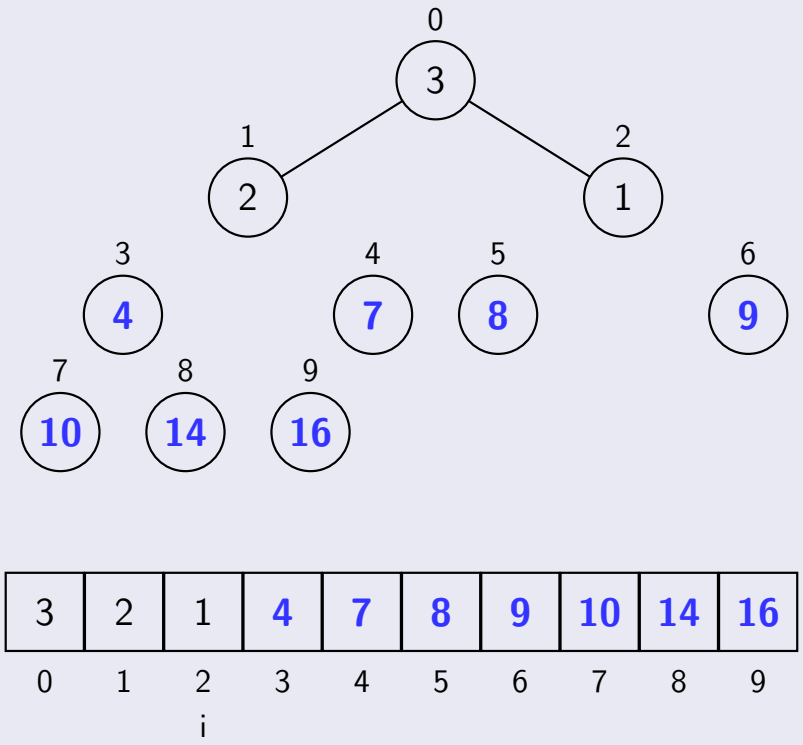
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

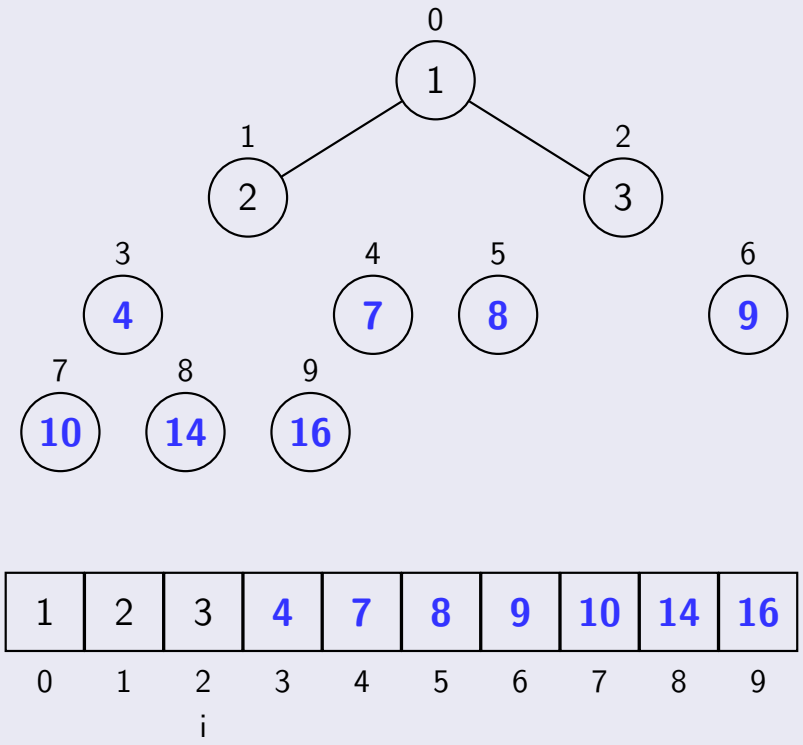
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

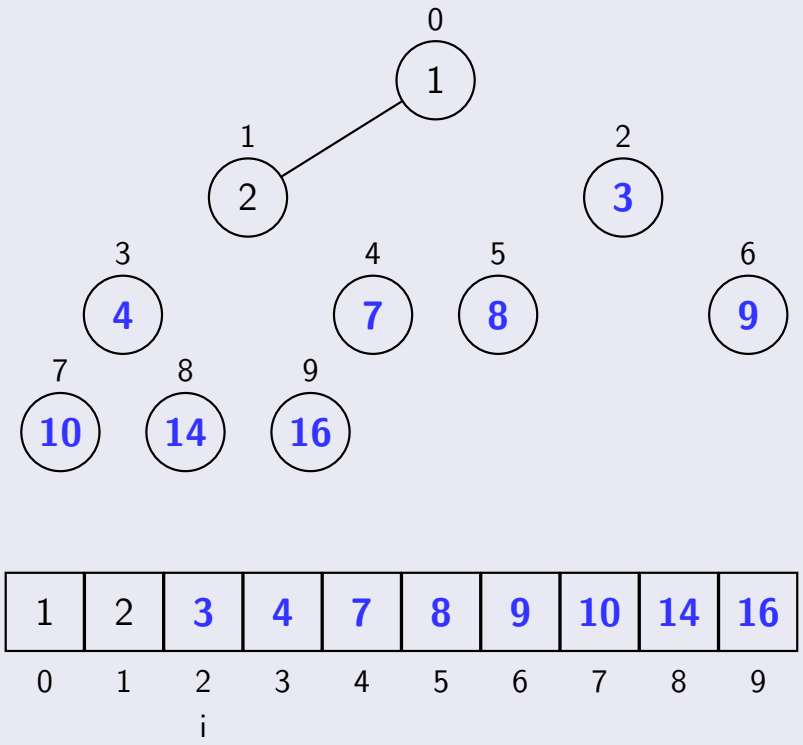
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

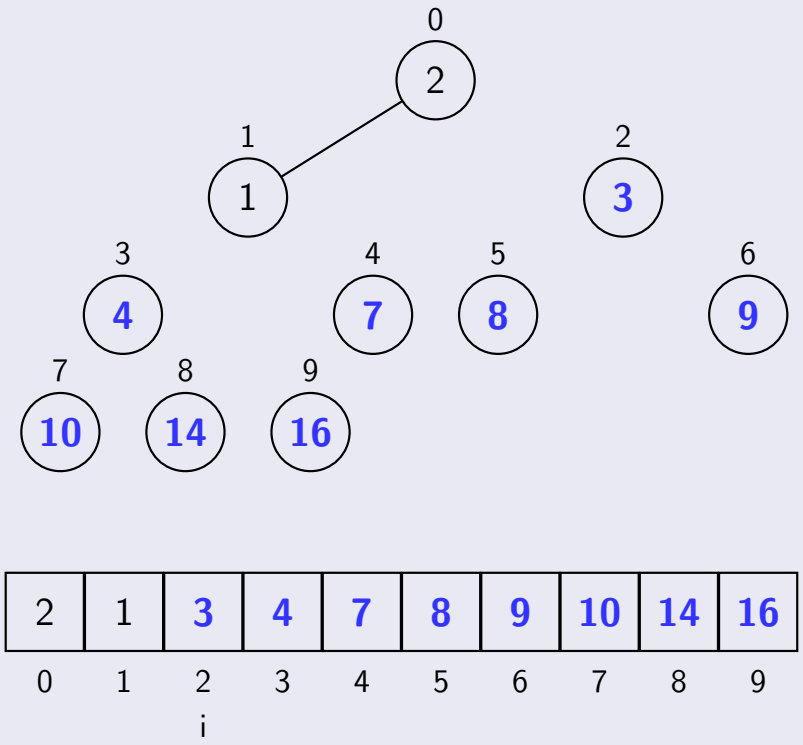
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

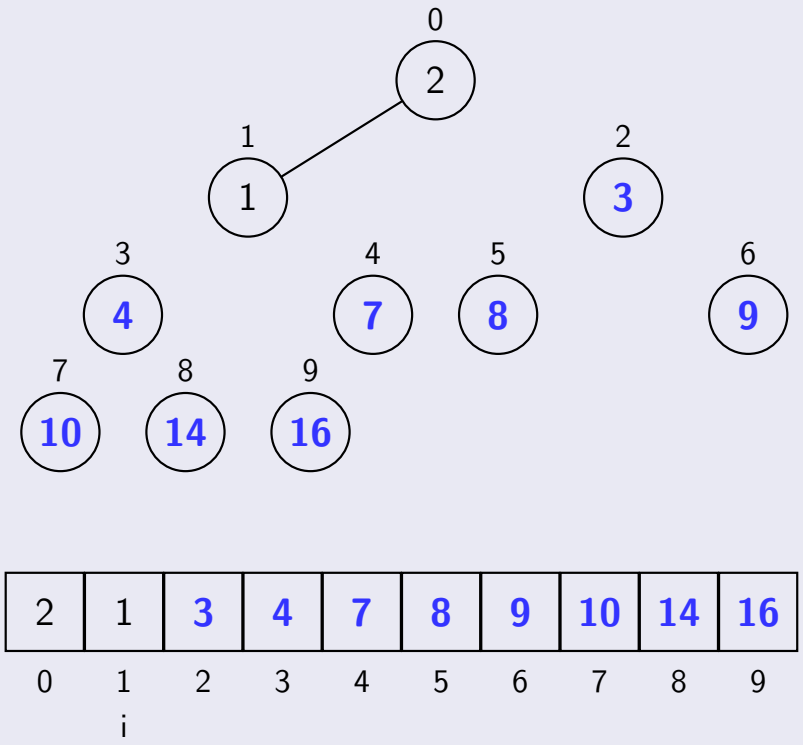
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

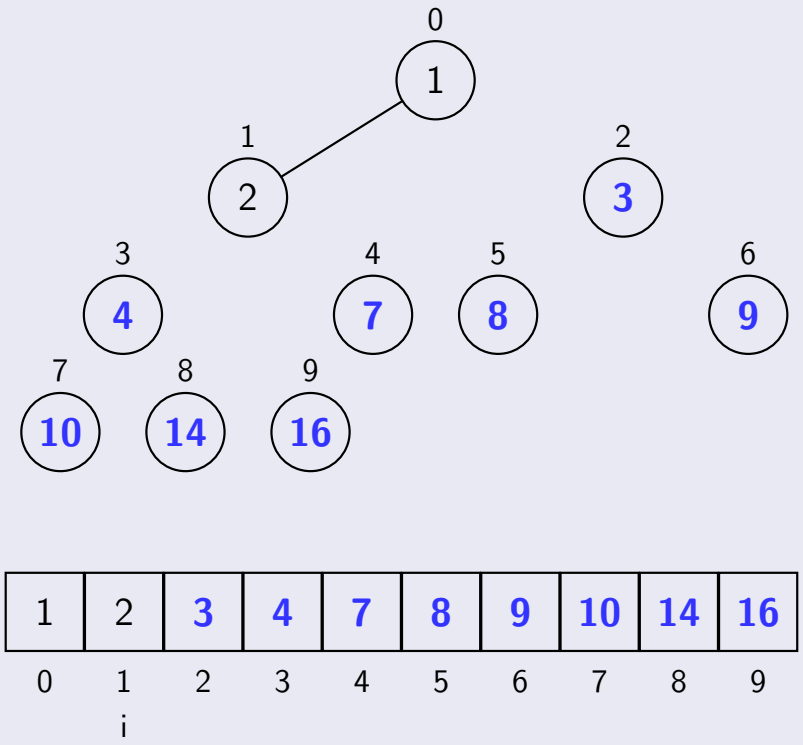
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

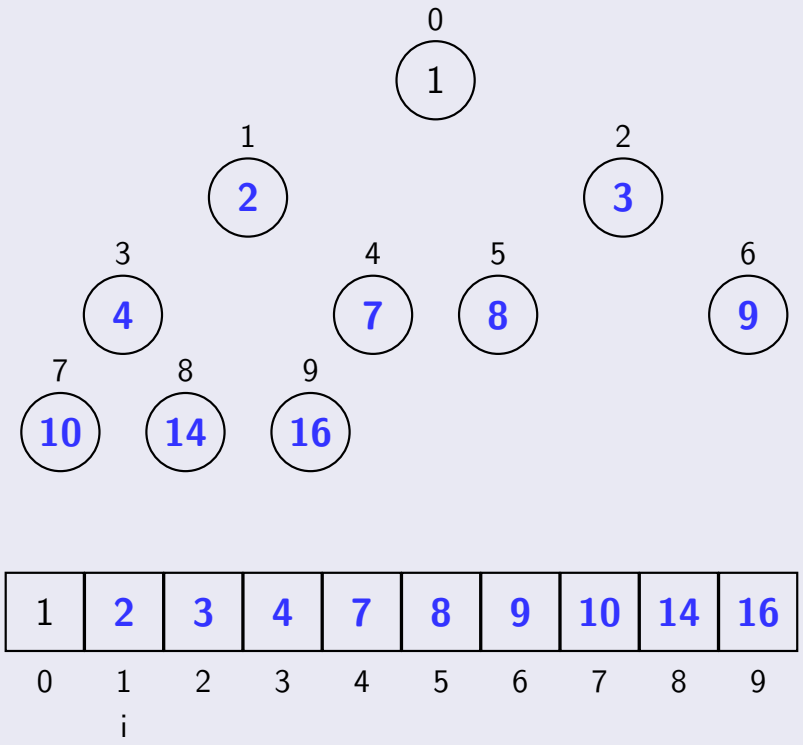
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

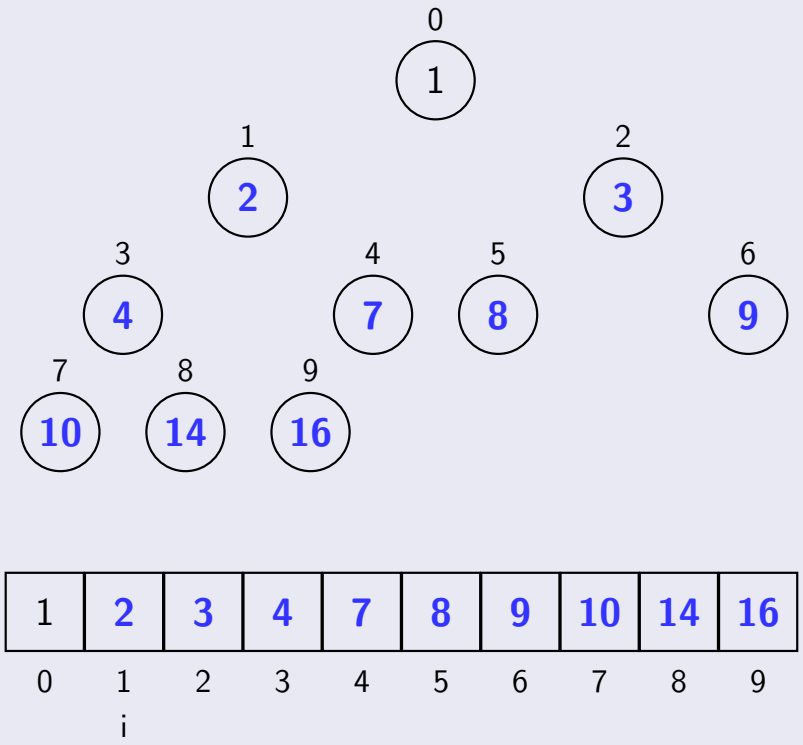
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

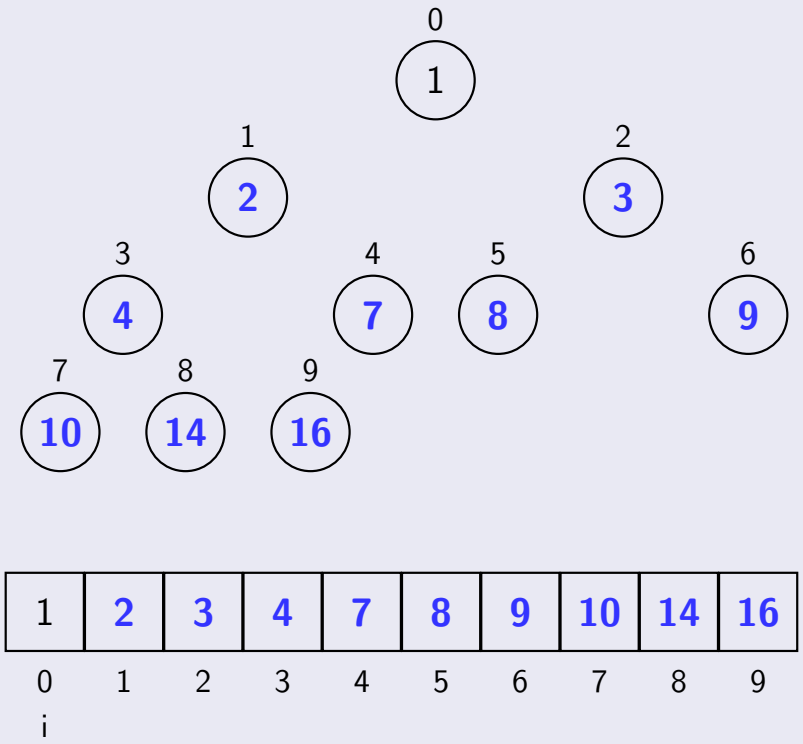
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

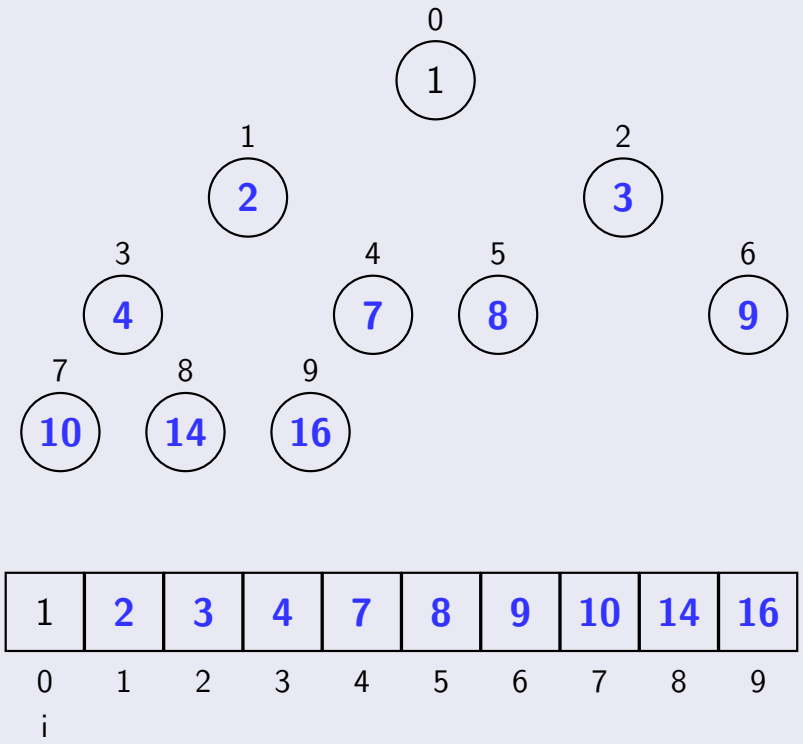
```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```



HeapSort

Código

- Qual a complexidade desse código?

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

HeapSort

Código

- Qual a complexidade desse código?
  - $O(n)$

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

HeapSort

Código

- Qual a complexidade desse código?
  - $O(n)$

    $+(n-1)$

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

HeapSort

Código

- Qual a complexidade desse código?
  - $O(n)$

    $+(n-1) \times O(log\ n)$

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

HeapSort

Código

- Qual a complexidade desse código?
  - $O(n)$

    $+(n-1) \times O(log\ n)$
- $O(n\ log\ n)$ portanto

```
void heapSort(int A[], int n) {
  int i, compHeap, temp;

  compHeap = n;
  constroiHeapMax(A, n);
  for(i = n-1; i > 0; i--) {
    temp = A[0];
    A[0] = A[i];
    A[i] = temp;
    compHeap--;
    refazHeapMax(A, 0, compHeap);
  }
}
```

Referências

- Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford. Introduction to Algorithms. 2a ed. MIT Press, 2001.
- Ziviani, Nivio. Projeto de Algoritmos: com implementações em Java e C++. Cengage. 2007.
- Slides dos professores Delano Beder e Marcos Chain

Aula 20 – Heapsort

Norton T. Roman & Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

2023