Estruturas de Dados II

Heap e Heap-Sort

Prof[a]. Juliana de Santi
Prof. Rodrigo Minetto
Universidade Tecnológica Federal do Paraná

Sumário

O algoritmo de ordenação **heap-sort** foi inventado por J. Williams em 1964. Este algoritmo é considerado ótimo, com tempo de execução de $\mathcal{O}(n \log n)$ no pior caso. O algoritmo é **in-place** (requer um número constante de elementos armazenados fora do arranjo de entrada em qualquer instante) e **não estável** (não mantém a ordem de elementos iguais).

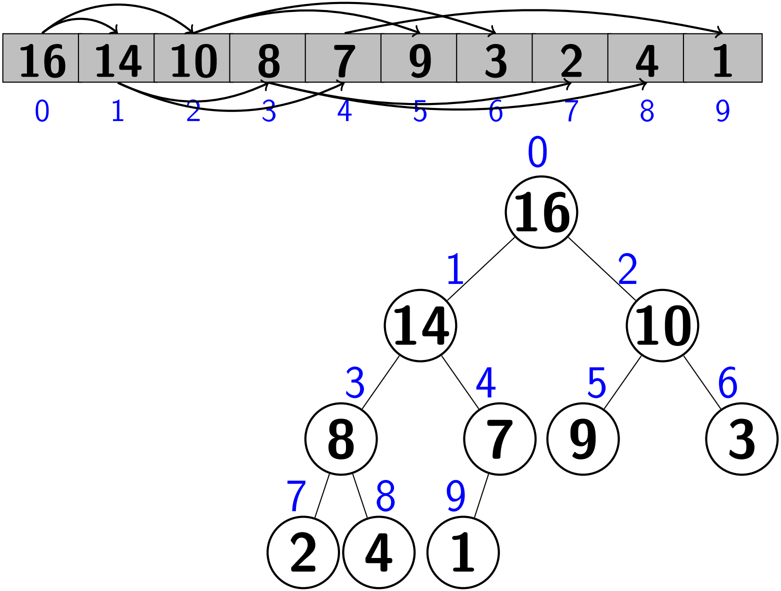
Heap-Sort

O algoritmo **heap-sort** introduziu outra técnica de projetos de algoritmos: o uso de um TAD conhecido como **heap** (ou “monte”) para gerenciar os dados durante a execução do algoritmo. A estrutura do **heap** não é apenas importante para o heap-sort mas ela é essencial para criar uma eficiente **fila de prioridades**.

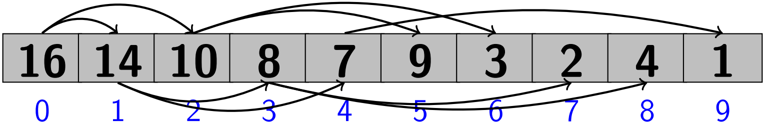
Heap

A estrutura de dados **heap** (**binário**) é um arranjo que pode ser visualizado como uma árvore binária praticamente completa (a exceção ocorre no último nível que é preenchido da esquerda para direita enquanto existirem elementos). Cada nó da árvore corresponde a um elemento do arranjo.

Heap



Heap



Pai (**i**)

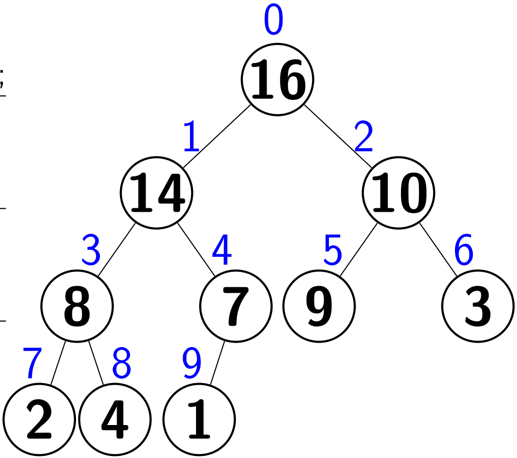
Return $\lfloor (\mathbf{i} - 1)/2 \rfloor$;

Esquerda (**i**)

Return $\mathbf{i} * 2 + 1$;

Direita (**i**)

Return $\mathbf{i} * 2 + 2$;



Heap

A raiz da árvore para um vetor V sempre se encontra no elemento $V[0]$. Existem dois tipos de heaps: **heaps máximos** e **heaps mínimos**. Em um **heap máximo** temos a seguinte propriedade:

$$V[\textsc{Pai}(i)] \geq V[i]$$

isto é, o valor de um nó é no máximo o valor de seu pai. Desse modo, o maior elemento em um heap máximo é armazenado na raiz.

Um **heap mínimo** é organizado de forma oposta, ele possui a seguinte propriedade:

$$V[\textsc{Pai}(i)] \leq V[i]$$

isto é, o menor valor em um heap mínimo está na raiz. O algoritmo *heap-sort* usa um *heap-máximo*. Heaps mínimos são comumente utilizados em filas de prioridades.

Heap

Visualizando um **heap** como uma **árvore**, definimos a **altura** de um nó em um heap como o número de arestas no caminho descendente mais longo da raiz até uma folha. Tendo em vista que um **heap** com n elementos é baseado em uma árvore binária completa, sua altura é $\Theta(\log n)$. As operações básicas sobre um **heap** têm complexidade proporpocional à altura da árvore e assim demoram $\mathcal{O}(\log n)$.

Sumário

Heap máximo

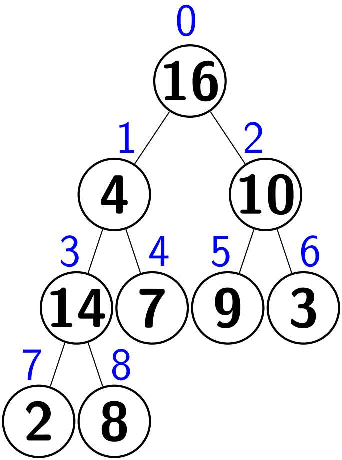
A operação Max-Heapify é útil para manipular **heaps máximos**. Quando Max-Heapify é invocada ela supõe que as árvores binárias com raízes em Esquerda(i) e Direita(i) são heaps máximos, mas que $V[i]$ pode ser menor que seus filhos, violando assim a propriedade de heap máximo. Assim, $V[i]$ deve “descer” no heap máximo, de tal forma que a sub-árvore com raiz em i se torne um heap-máximo.

Heap máximo

MAX-HEAPIFY (V, SIZE, i)

---

e = ESQUERDA(i);
d = DIREITA(i);
If e < SIZE **and** V[e] > V[i]
    maior = e;
Else
    maior = i;
If d < SIZE **and** V[d] > V[maior]
    maior = d;
If maior ≠ i
    V[i] ↔ V[maior];
    MAX-HEAPIFY (V, SIZE, maior);

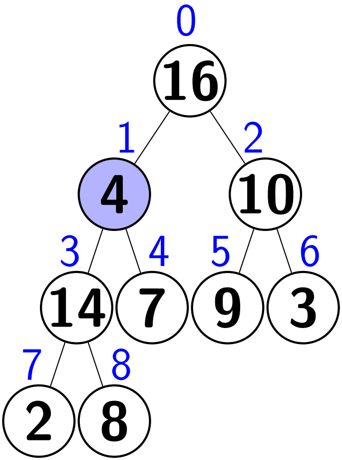


Heap máximo

MAX-HEAPIFY (V, SIZE, 1)

---

e = ESQUERDA(1);
d = DIREITA(1);
If e < SIZE **and** V[e] > V[1]
    maior = e;
Else
    maior = 1;
If d < SIZE **and** V[d] > V[maior]
    maior = d;
If maior ≠ 1
    V[1] ↔ V[maior];
    MAX-HEAPIFY (V, SIZE, maior);

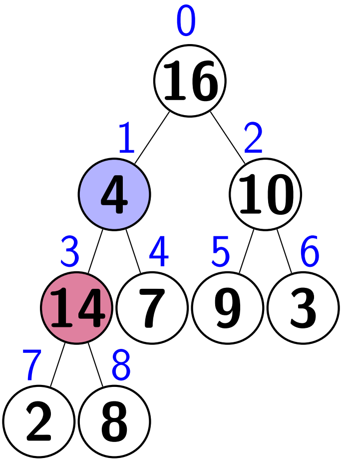


Heap máximo

MAX-HEAPIFY (V, SIZE, **1**)

---

3 $=$ ESQUERDA(**1**);
d $=$ DIREITA(**1**);
If **3** $<$ SIZE **and** V[**3**] $>$ V[**1**]
    maior $=$ **3**;
Else
    maior $=$ **1**;
If **d** $<$ SIZE **and** V[**d**] $>$ V[**maior**]
    maior $=$ **d**;
If **maior** $\neq$ **1**
    V[**1**] $\leftrightarrow$ V[**maior**];
    MAX-HEAPIFY (V, SIZE, **maior**);

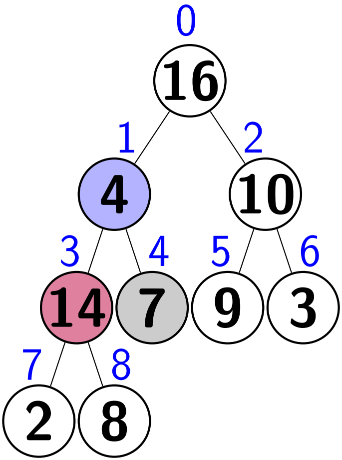


Heap máximo

Max-Heapify (V, Size, 1)

---

3 = Esquerda(1);
4 = Direita(1);
If 3 < Size **and** V[3] > V[1]
    maior = 3;
Else
    maior = 1;
If 4 < Size **and** V[4] > V[maior]
    maior = 4;
If maior ≠ 1
    V[1] ↔ V[maior];
    Max-Heapify (V, Size, maior);

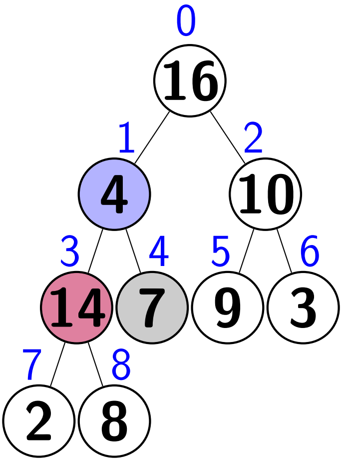


Heap máximo

MAX-HEAPIFY (V, 9, 1)

---

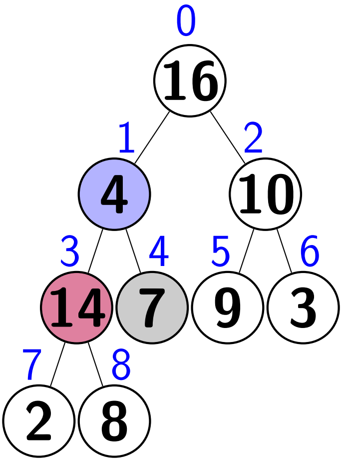
3 = ESQUERDA(1);
4 = DIREITA(1);
If 3 $<$ 9 **and** 14 $>$ 4
  maior = 3;
Else
  maior = 1;
If 4 $<$ SIZE **and** V[4] $>$ V[**maior**]
  maior = 4;
If **maior** $\neq$ 1
  V[1] $\leftrightarrow$ V[**maior**];
  MAX-HEAPIFY (V, SIZE, **maior**);



Heap máximo

$\textsc{Max-Heapify}$ (V, 9, **1**)

$3 = \textsc{Esquerda}(1)$;
$4 = \textsc{Direita}(1)$;
If $3 < 9$ **and** $14 > 4$
  $\triangleright$**maior** $= 3$;
Else
  maior $= 1$;
If $4 < \textsc{Size}$ **and** $V[4] > V[\text{maior}]$
  maior $= 4$;
If **maior** $\neq 1$
  $V[1] \leftrightarrow V[\text{maior}]$;
  $\textsc{Max-Heapify}$ (V, $\textsc{Size}$, **maior**);

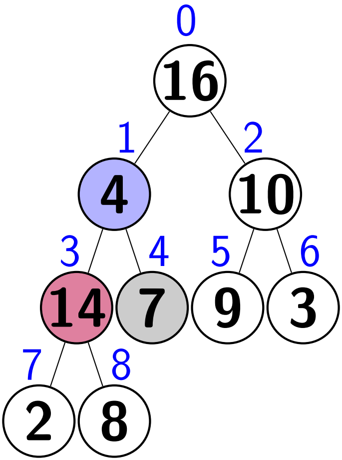


Heap máximo

$\textsc{Max-Heapify}$ (V, 9, 1)

---

3 = $\textsc{Esquerda}$(1);
4 = $\textsc{Direita}$(1);
If 3 < 9 **and** 14 > 4
    ▷**maior** = 3;
Else
    maior = 1;
If 4 < $\textsc{Size}$ **and** V[4] > V[3]
    maior = 4;
If **maior** ≠ 1
    V[1] ↔ V[**maior**];
    $\textsc{Max-Heapify}$ (V, $\textsc{Size}$, **maior**);

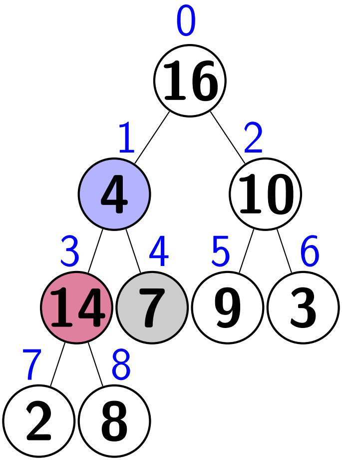


Heap máximo

MAX-HEAPIFY (V, 9, 1)

---

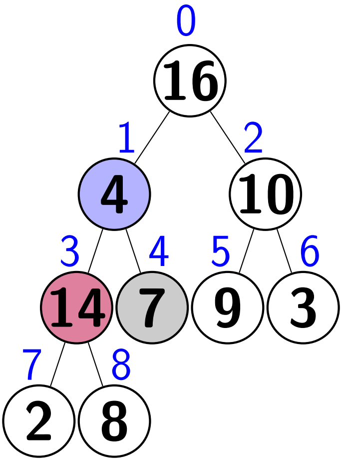
3 = ESQUERDA(1);
4 = DIREITA(1);
If 3 $<$ 9 **and** 14 $>$ 4
    $\triangleright$**maior** = 3;
Else
    maior = 1;
If 4 $<$ 9 **and** 7 $>$ 14
    maior = 4;
If **maior** $\neq$ 1
    V[1] $\leftrightarrow$ V[**maior**];
    MAX-HEAPIFY (V, SIZE, **maior**);



MAX-HEAPIFY (V, 9, **1**)

---

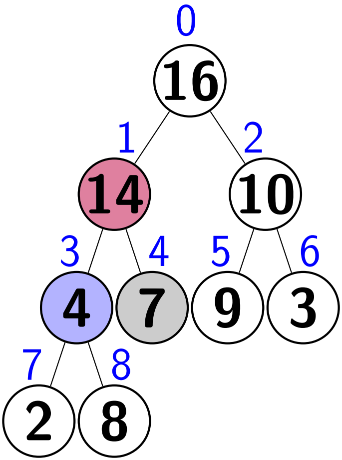
3 = ESQUERDA(**1**);
4 = DIREITA(**1**);
If **3** $<$ 9 **and** **14** $>$ **4**
    $\triangleright$**maior** = **3**;
Else
    maior = **1**;
If 4 $<$ 9 **and** 7 $>$ **14**
    maior = 4;
If **3** $\neq$ **1**
    V[**1**] $\leftrightarrow$ V[**maior**];
    MAX-HEAPIFY (V, SIZE, **maior**);



Heap máximo

$\textsc{Max-Heapify}\ (V,\ 9,\ \mathbf{1})$

$\mathbf{3} = \textsc{Esquerda}(\mathbf{1})$;
$\mathbf{4} = \textsc{Direita}(\mathbf{1})$;
If $\mathbf{3} < 9$ **and** $\mathbf{14} > \mathbf{4}$
  $\triangleright$**maior** $= \mathbf{3}$;
Else
  maior $= \mathbf{1}$;
If $4 < 9$ **and** $7 > \mathbf{14}$
  maior $= 4$;
If $\mathbf{3} \neq \mathbf{1}$
  V[1] $\leftrightarrow$ V[3];
  $\textsc{Max-Heapify}$ (V, $\textsc{Size}$, **maior**);

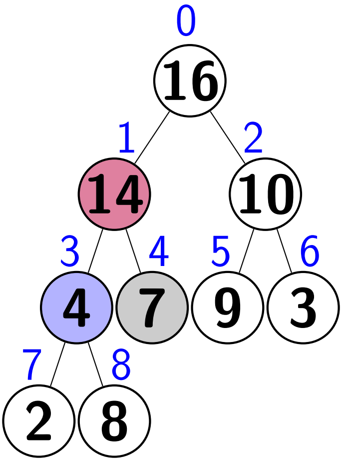


Heap máximo

MAX-HEAPIFY (V, 9, **1**)

---

3 = ESQUERDA(**1**);
4 = DIREITA(**1**);
If **3** $<$ 9 **and** **14** $>$ **4**
  $\triangleright$**maior** = **3**;
Else
  maior = **1**;
If **4** $<$ 9 **and** **7** $>$ **14**
  maior = **4**;
If **3** $\neq$ **1**
  V[**1**] $\leftrightarrow$ V[**3**];
  MAX-HEAPIFY (V, 9, **3**);

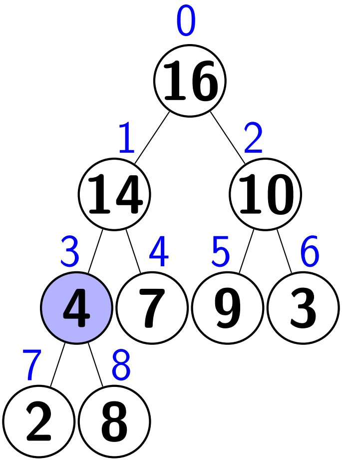


Heap máximo

MAX-HEAPIFY (V, SIZE, **3**)

---

e = ESQUERDA(**3**);
d = DIREITA(**3**);
If **e** < SIZE **and** V[**e**] > V[**3**]
    maior = **e**;
Else
    maior = **3**;
If **d** < SIZE **and** V[**d**] > V[**maior**]
    maior = **d**;
If **maior** ≠ **3**
    V[**3**] ↔ V[**maior**];
    MAX-HEAPIFY (V, SIZE, **maior**);

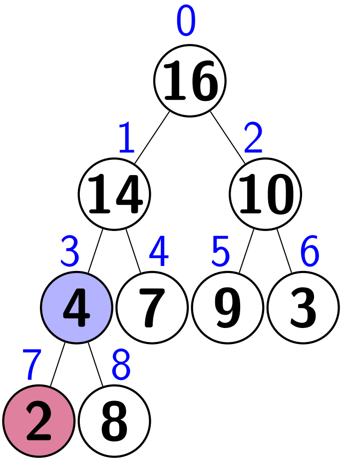


Heap máximo

MAX-HEAPIFY (V, SIZE, **3**)

---

7 = ESQUERDA(**3**);
d = DIREITA(**3**);
If **7** < SIZE **and** V[**7**] > V[**3**]
    maior = **7**;
Else
    maior = **3**;
If **d** < SIZE **and** V[**d**] > V[**maior**]
    maior = **d**;
If **maior** $\neq$ **3**
    V[**3**] $\leftrightarrow$ V[**maior**];
    MAX-HEAPIFY (V, SIZE, **maior**);

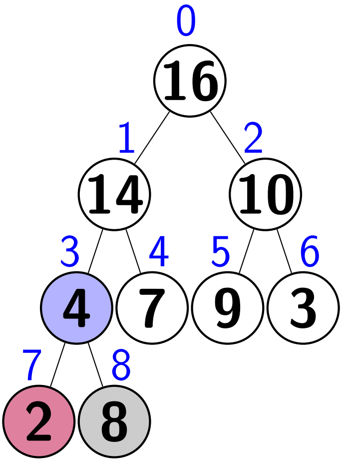


Heap máximo

Max-Heapify (V, Size, **3**)

---

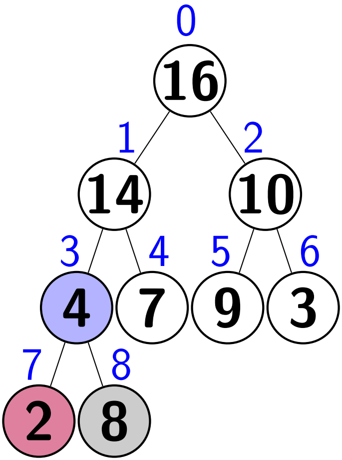
7 = Esquerda(**3**);
8 = Direita(**3**);
If 7 < Size **and** V[7] > V[3]
    maior = **7**;
Else
    maior = **3**;
If 8 < Size **and** V[8] > V[**maior**]
    maior = **8**;
If **maior** $\neq$ **3**
    V[**3**] $\leftrightarrow$ V[**maior**];
    Max-Heapify (V, Size, **maior**);



Heap máximo

MAX-HEAPIFY (V, 9, **3**)

7 = ESQUERDA(**3**);
8 = DIREITA(**3**);
If **7** $<$ 9 **and** **2** $>$ **4**
    maior = **7**;
Else
    maior = **3**;
If **8** $<$ SIZE **and** V[**8**] $>$ V[**maior**]
    maior = **8**;
If **maior** $\neq$ **3**
    V[**3**] $\leftrightarrow$ V[**maior**];
    MAX-HEAPIFY (V, SIZE, **maior**);

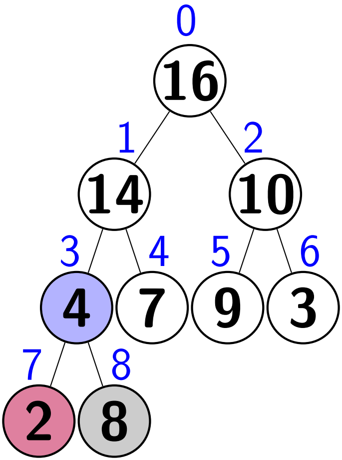


Heap máximo

Max-Heapify (V, 9, **3**)

---

7 = Esquerda(**3**);
8 = Direita(**3**);
If **7** $<$ 9 **and** **2** $>$ **4**
    maior = **7**;
Else
    $\triangleright$**maior** = **3**;
If **8** $<$ Size **and** V[**8**] $>$ V[**maior**]
    maior = **8**;
If **maior** $\neq$ **3**
    V[**3**] $\leftrightarrow$ V[**maior**];
    Max-Heapify (V, Size, **maior**);

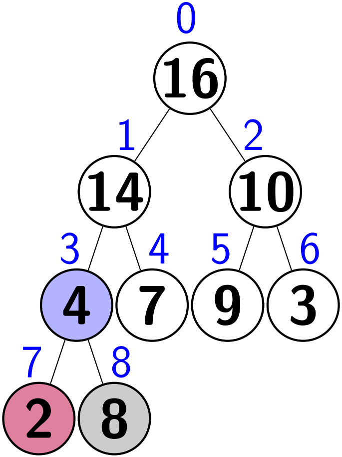


Heap máximo

MAX-HEAPIFY (V, 9, 3)

---

7 = ESQUERDA(3);
8 = DIREITA(3);
If 7 $<$ 9 **and** 2 $>$ 4
    maior = 7;
Else
    ▷**maior** = 3;
If 8 $<$ SIZE **and** V[8] $>$ V[3]
    maior = 8;
If **maior** $\neq$ 3
    V[3] $\leftrightarrow$ V[**maior**];
    MAX-HEAPIFY (V, SIZE, **maior**);

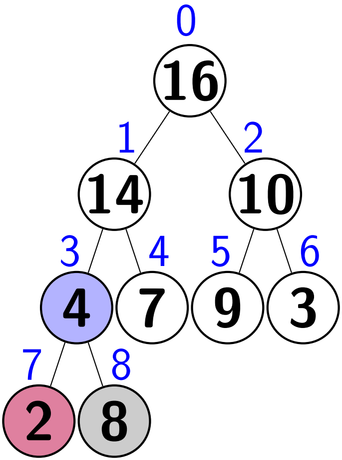


Heap máximo

MAX-HEAPIFY (V, 9, 3)

---

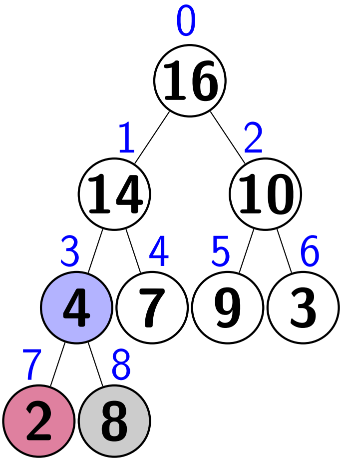
7 = ESQUERDA(3);
8 = DIREITA(3);
If 7 $<$ 9 **and** 2 $>$ 4
    maior = 7;
Else
    ▷**maior** = 3;
If 8 $<$ 9 **and** 8 $>$ 4
    maior = 8;
If maior $\neq$ 3
    V[3] $\leftrightarrow$ V[maior];
    MAX-HEAPIFY (V, SIZE, **maior**);



Heap máximo

$\textsc{Max-Heapify}\ (V,\ 9,\ \mathbf{3})$

$\mathbf{7} = \textsc{Esquerda}(\mathbf{3})$;
$\mathbf{8} = \textsc{Direita}(\mathbf{3})$;
If $\mathbf{7} < 9$ **and** $\mathbf{2} > \mathbf{4}$
  maior $= \mathbf{7}$;
Else
  maior $= \mathbf{3}$;
If $\mathbf{8} < 9$ **and** $\mathbf{8} > \mathbf{4}$
  $\triangleright$ **maior** $= \mathbf{8}$;
If **maior** $\neq \mathbf{3}$
  V[**3**] $\leftrightarrow$ V[**maior**];
  $\textsc{Max-Heapify}$ (V, $\textsc{Size}$, **maior**);



Heap máximo

$\text{MAX-HEAPIFY } (V, 9, \mathbf{3})$

$\mathbf{7} = \text{ESQUERDA}(\mathbf{3});$

$8 = \text{DIREITA}(\mathbf{3});$

If $\mathbf{7} < 9$ **and** $\mathbf{2} > \mathbf{4}$

    maior $= \mathbf{7};$

Else

    maior $= \mathbf{3};$

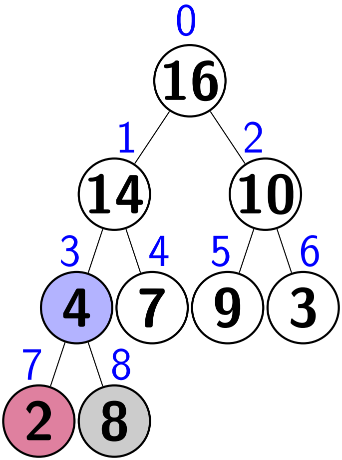
If $8 < 9$ **and** $8 > \mathbf{4}$

    $\triangleright$ **maior** $= 8;$

If $\mathbf{8} \neq \mathbf{3}$

    $V[\mathbf{3}] \leftrightarrow V[\textbf{maior}];$

    $\text{MAX-HEAPIFY } (V, \text{SIZE}, \textbf{maior});$

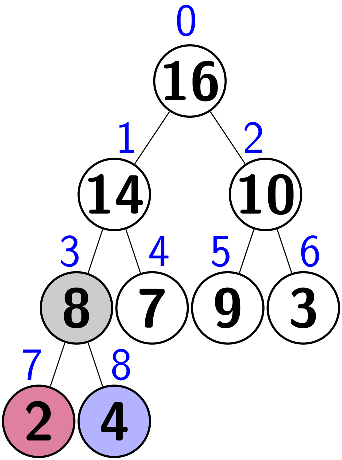


Heap máximo

Max-Heapify (V, 9, **3**)

---

7 = Esquerda(**3**);
8 = Direita(**3**);
If 7 $<$ 9 **and 2** $>$ **4**
  maior = **7**;
Else
  maior = **3**;
If **8** $<$ 9 **and** **8** $>$ **4**
  $\triangleright$ **maior** = **8**;
If 8 $\neq$ **3**
  V[**3**] $\leftrightarrow$ V[**8**];
  Max-Heapify (V, Size, **maior**);

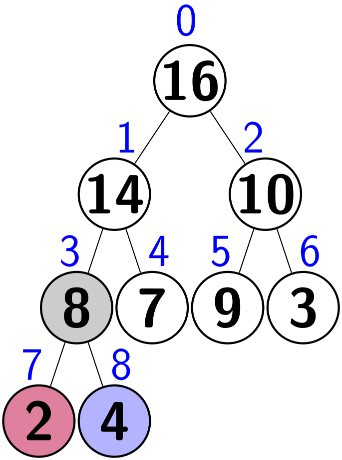


Heap máximo

MAX-HEAPIFY (V, 9, **3**)

---

7 = ESQUERDA(**3**);
8 = DIREITA(**3**);
If **7** < 9 **and** **2** > **4**
    maior = **7**;
Else
    maior = **3**;
If **8** < 9 **and** **8** > **4**
    $\triangleright$ **maior** = **8**;
If **8** $\neq$ **3**
    V[**3**] $\leftrightarrow$ V[**8**];
    MAX-HEAPIFY (V, 9, **8**);

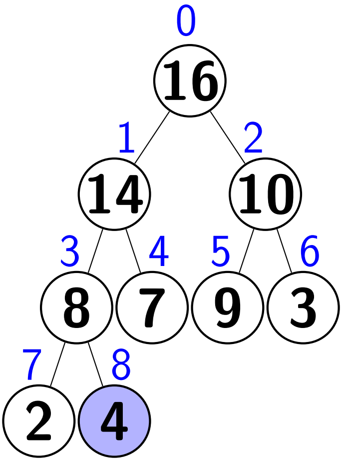


Heap máximo

MAX-HEAPIFY (V, SIZE, **8**)

---

e = ESQUERDA(**8**);
d = DIREITA(**8**);
If **e** < SIZE **and** V[**e**] > V[**8**]
  maior = **e**;
Else
  maior = **8**;
If **d** < SIZE **and** V[**d**] > V[**maior**]
  maior = **d**;
If **maior** ≠ **8**
  V[**8**] ↔ V[**maior**];
  MAX-HEAPIFY (V, SIZE, **maior**);

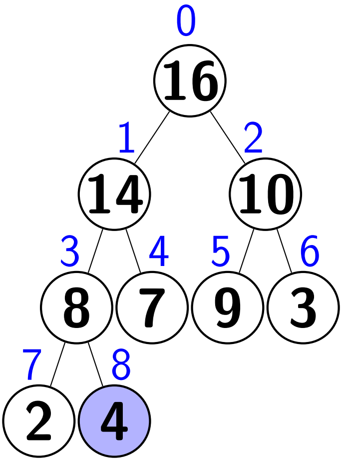


Heap máximo

MAX-HEAPIFY (V, 9, **8**)

---

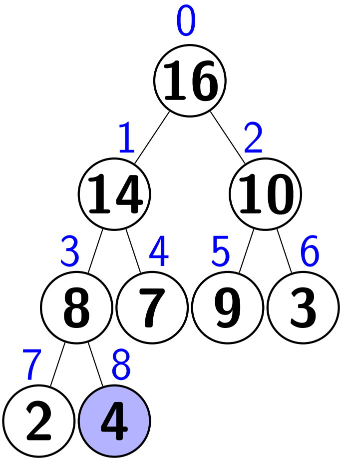
17 = ESQUERDA(**8**);
d = DIREITA(**8**);
If 17 $<$ 9 **and** V[**17**] $>$ V[**8**]
    maior = **17**;
Else
    maior = **8**;
If d $<$ SIZE **and** V[**d**] $>$ V[**maior**]
    maior = **d**;
If maior $\neq$ **8**
    V[**8**] $\leftrightarrow$ V[**maior**];
    MAX-HEAPIFY (V, SIZE, **maior**);



Heap máximo

MAX-HEAPIFY (V, 9, **8**)

17 = ESQUERDA(**8**);
18 = DIREITA(**8**);
If **17** $<$ 9 **and** V[**17**] $>$ V[**8**]
  maior = **17**;
Else
  maior = **8**;
If 18 $<$ 9 **and** V[18] $>$ V[**maior**]
  maior = 18;
If **maior** $\neq$ **8**
  V[**8**] $\leftrightarrow$ V[**maior**];
  MAX-HEAPIFY (V, SIZE, **maior**);

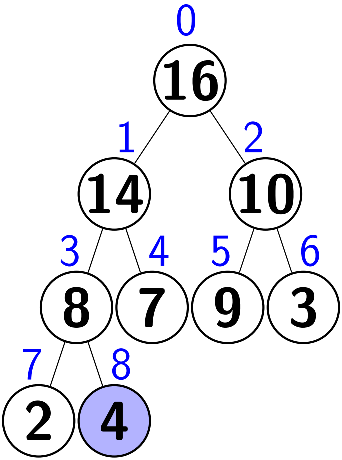


Heap máximo

MAX-HEAPIFY (V, 9, **8**)

---

17 = ESQUERDA(**8**);
18 = DIREITA(**8**);
If **17** < 9
    maior = **17**;
Else
    maior = **8**;
If 18 < 9 **and** V[18] > V[**maior**]
    maior = 18;
If **maior** ≠ **8**
    V[**8**] ↔ V[**maior**];
    MAX-HEAPIFY (V, SIZE, **maior**);

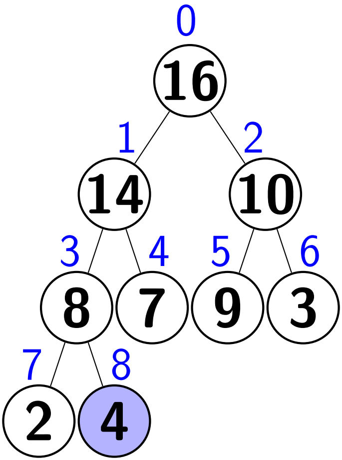


Heap máximo

MAX-HEAPIFY (V, 9, **8**)

---

17 = ESQUERDA(**8**);
18 = DIREITA(**8**);
If **17** < 9
    maior = **17**;
Else
    ▷**maior** = **8**;
If 18 < 9 **and** V[18] > V[**maior**]
    maior = 18;
If **maior** ≠ **8**
    V[**8**] ↔ V[**maior**];
    MAX-HEAPIFY (V, SIZE, **maior**);

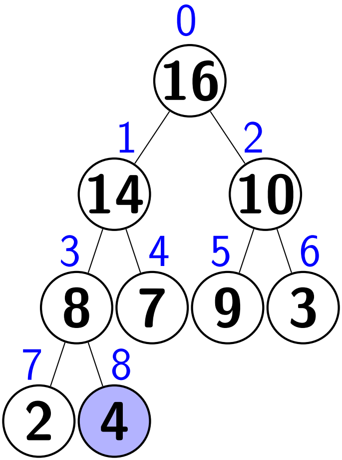


Heap máximo

MAX-HEAPIFY (V, 9, **8**)

---

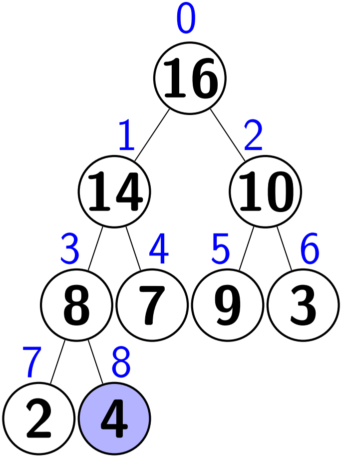
17 = ESQUERDA(**8**);
18 = DIREITA(**8**);
If **17** < 9
    maior = **17**;
Else
    ▷**maior** = **8**;
If 18 < 9 **and** V[18] > V[**8**]
    maior = 18;
If **maior** ≠ **8**
    V[**8**] ↔ V[**maior**];
    MAX-HEAPIFY (V, SIZE, **maior**);



Heap máximo

MAX-HEAPIFY (V, 9, **8**)

17 = ESQUERDA(**8**);
18 = DIREITA(**8**);
If **17** $<$ 9
  maior = **17**;
Else
  ▷**maior** = **8**;
If 18 $<$ 9
  maior = 18;
If **maior** $\neq$ **8**
  V[**8**] $\leftrightarrow$ V[**maior**];
  MAX-HEAPIFY (V, SIZE, **maior**);

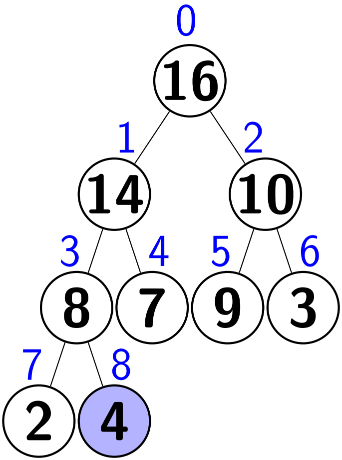


Heap máximo

MAX-HEAPIFY (V, 9, **8**)

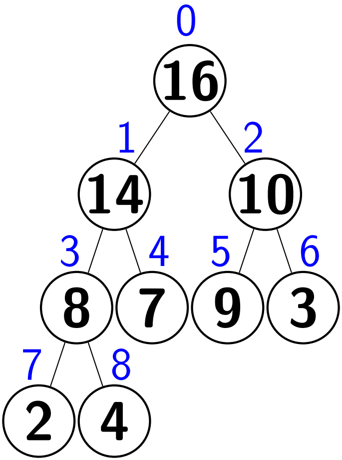
---

17 = ESQUERDA(**8**);
18 = DIREITA(**8**);
If **17** $<$ 9
    maior = **17**;
Else
    ▷**maior** = **8**;
If 18 $<$ 9
    maior = 18;
If **8** $\neq$ **8**
    V[**8**] $\leftrightarrow$ V[**maior**];
    MAX-HEAPIFY (V, SIZE, **maior**);



MAX-HEAPIFY (V, 9, **8**)

```
17 = ESQUERDA(8);
18 = DIREITA(8);
If 17 < 9
    maior = 17;
Else
    ▷maior = 8;
If 18 < 9
    maior = 18;
If 8 ≠ 8
    V[8] ↔ V[maior];
    MAX-HEAPIFY (V, SIZE, maior);
```

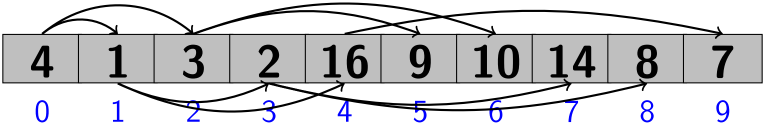


Sumário

Heap máximo

Podemos usar o procedimento $\textsc{Max-Heapify}$ de baixo para cima, com o objetivo de converter um arranjo $V[1 \ldots n]$ em um **heap máximo**. Note que em um **heap** os elementos $V[(\lfloor n/2 \rfloor + 1) \ldots n]$ são todos folhas da árvore, e assim não precisam ser organizados (são heaps máximos de 1 elemento).

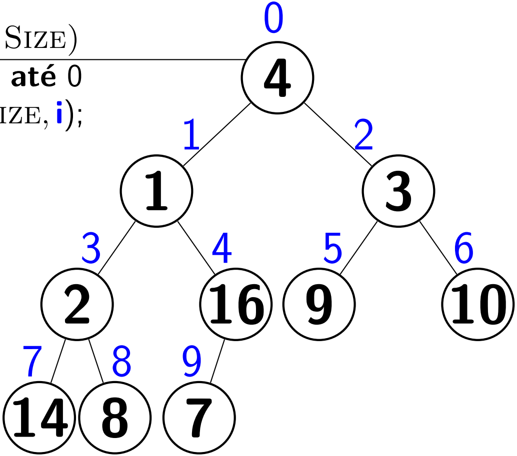
Construindo um Heap Máximo



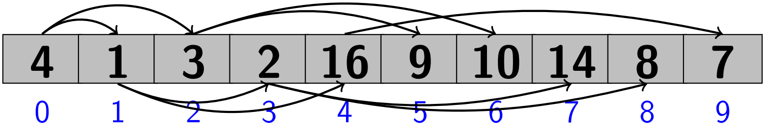
Build-Max-Heap (V, Size)

Para **i** de $\lfloor(\text{Size}/2)\rfloor - 1$ **até** 0

    Max-Heapify (V, Size, **i**);

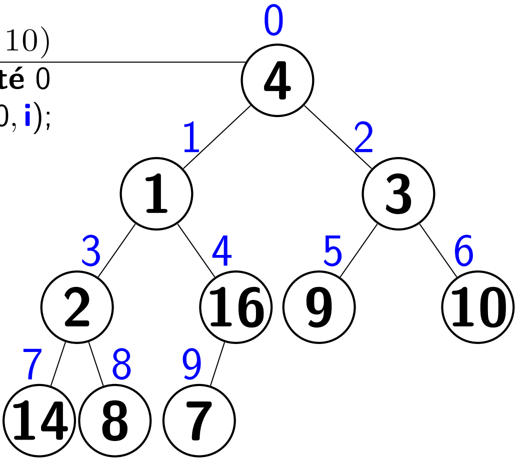


Construindo um Heap Máximo

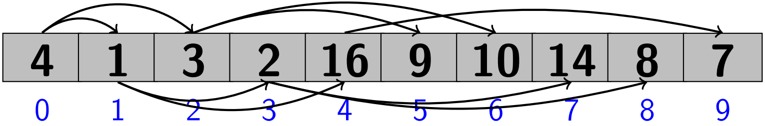


BUILD-MAX-HEAP $(V,\ 10)$

Para **i** de $\lfloor (10/2) \rfloor - 1$ **até** 0
    MAX-HEAPIFY $(V, 10, \mathbf{i})$;



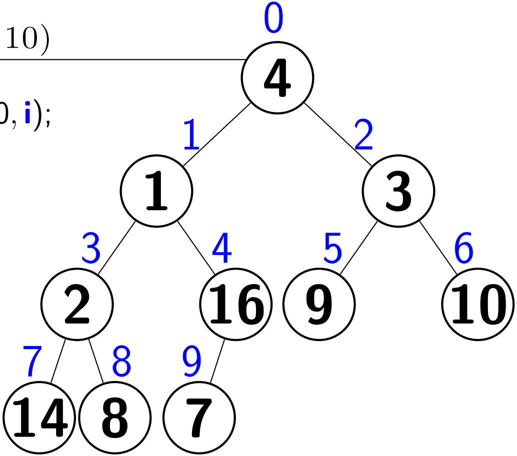
Construindo um Heap Máximo



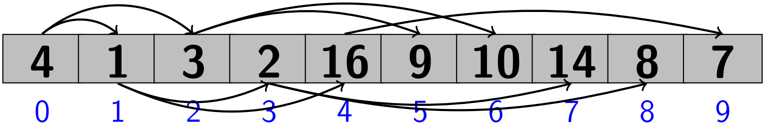
Build-Max-Heap $(V,\ 10)$

Para i de $\lfloor 5 \rfloor$-1 **até** 0

    Max-Heapify $(V, 10, \mathbf{i})$;



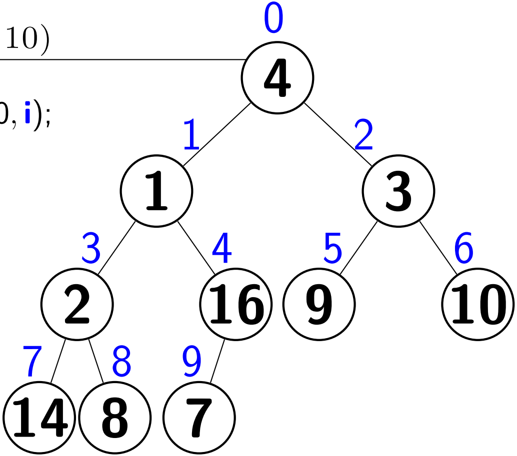
Construindo um Heap Máximo



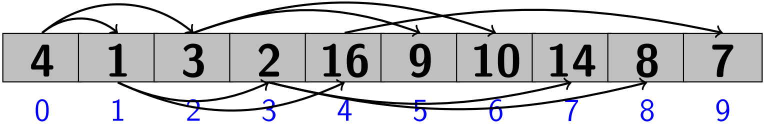
Build-Max-Heap $(V,\ 10)$

Para i de 4 **até** 0

Max-Heapify $(V, 10, \mathbf{i})$;



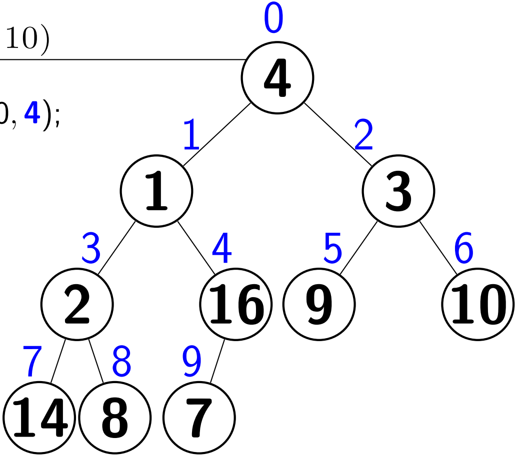
Construindo um Heap Máximo



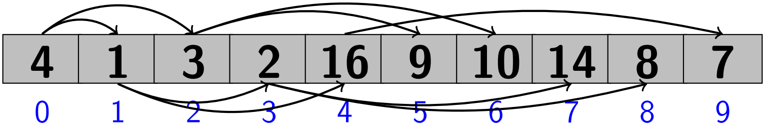
BUILD-MAX-HEAP (V, 10)

Para **i** de 4 **até** 0

MAX-HEAPIFY (V, 10, **4**);

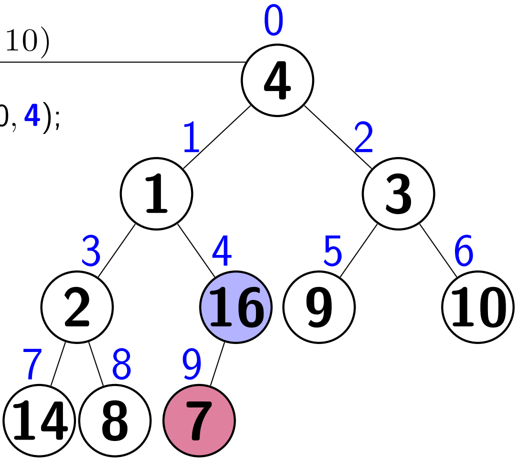


Construindo um Heap Máximo

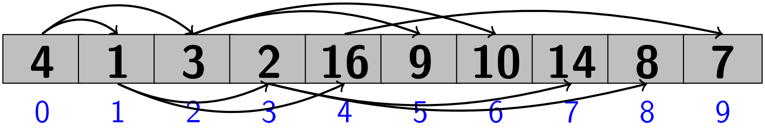


Build-Max-Heap (V, 10)

Para i de 4 **até** 0
    Max-Heapify (V, 10, **4**);

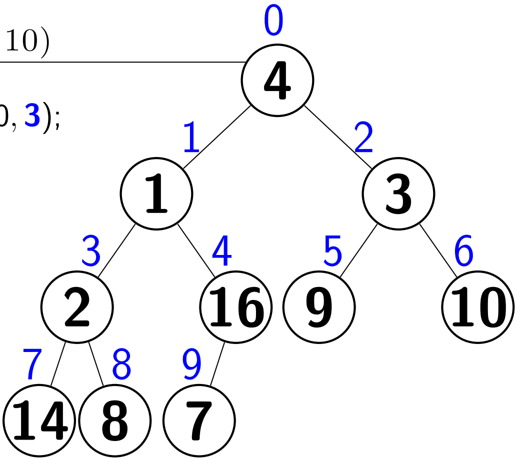


Construindo um Heap Máximo

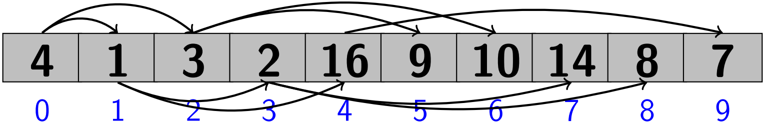


Build-Max-Heap $(V,\ 10)$

Para **i** de 4 **até** 0
    Max-Heapify $(V, 10, \mathbf{3})$;



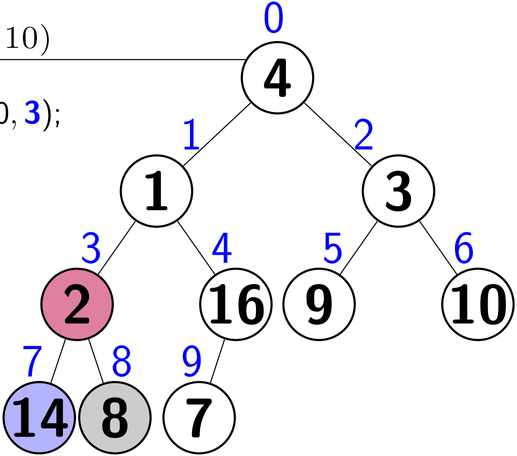
Construindo um Heap Máximo



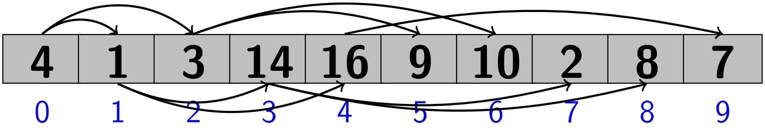
BUILD-MAX-HEAP $(V,\ 10)$

Para **i** de 4 **até** 0

MAX-HEAPIFY $(V, 10, \mathbf{3})$;



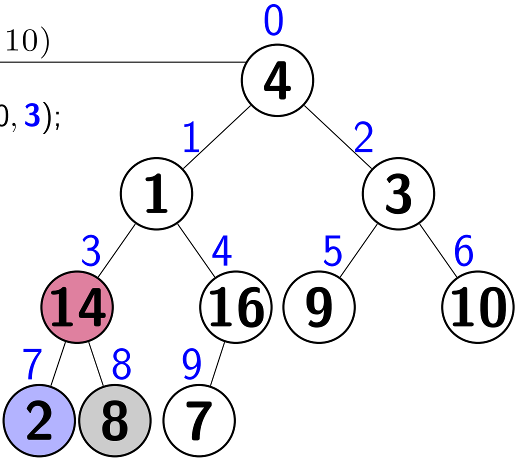
Construindo um Heap Máximo



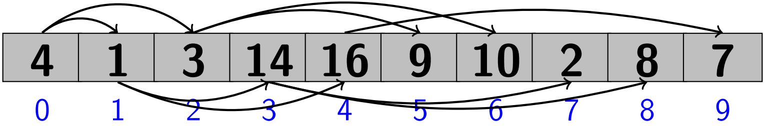
BUILD-MAX-HEAP $(V,\ 10)$

Para **i** de 4 **até** 0

MAX-HEAPIFY $(V, 10, \mathbf{3})$;



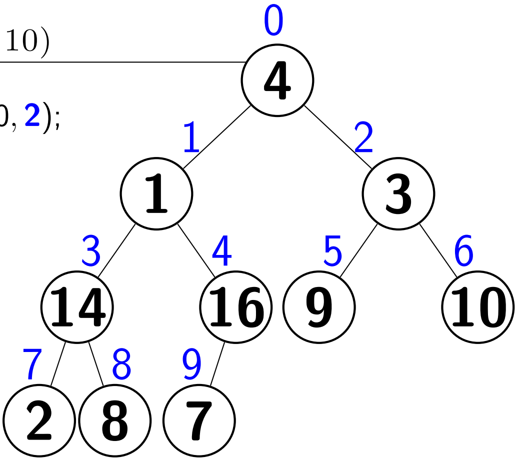
Construindo um Heap Máximo



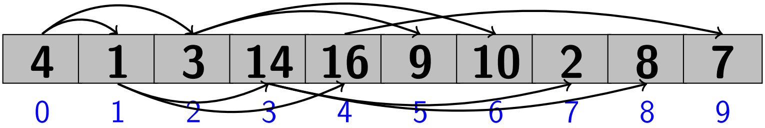
BUILD-MAX-HEAP $(V,\ 10)$

Para **i** de 4 **até** 0

MAX-HEAPIFY $(V, 10, \mathbf{2})$;



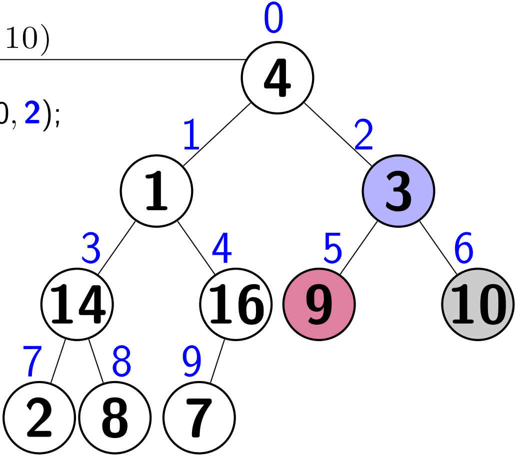
Construindo um Heap Máximo



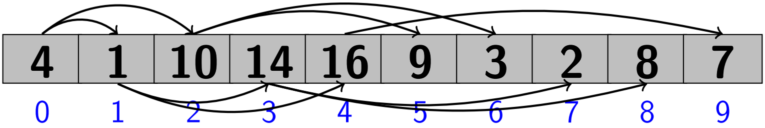
Build-Max-Heap $(V,\ 10)$

Para i de 4 **até** 0

  Max-Heapify $(V, 10, \mathbf{2})$;

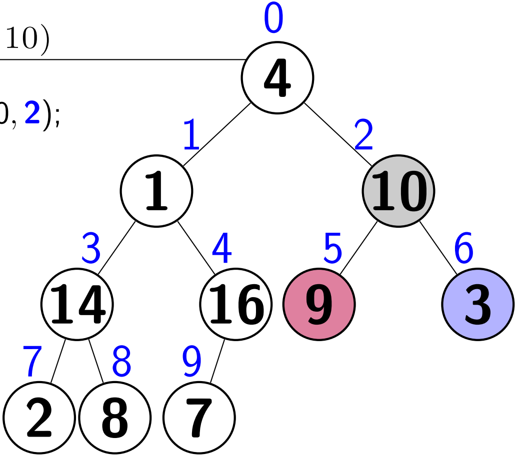


Construindo um Heap Máximo

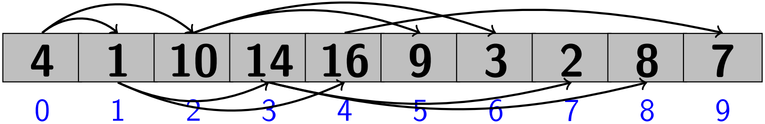


Build-Max-Heap $(V,\ 10)$

Para **i** de 4 **até** 0
    Max-Heapify $(V, 10, \mathbf{2})$;



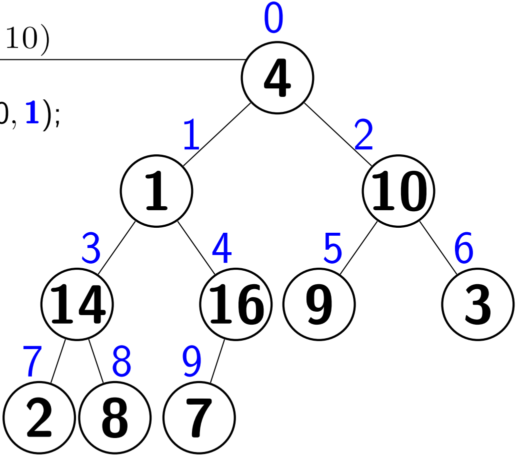
Construindo um Heap Máximo



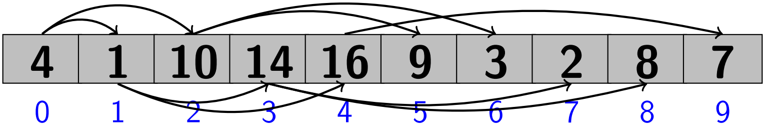
BUILD-MAX-HEAP (V, 10)

Para **i** de 4 **até** 0

MAX-HEAPIFY (V, 10, **1**);



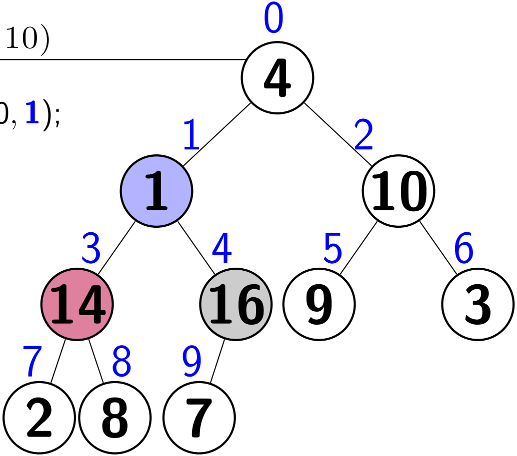
Construindo um Heap Máximo



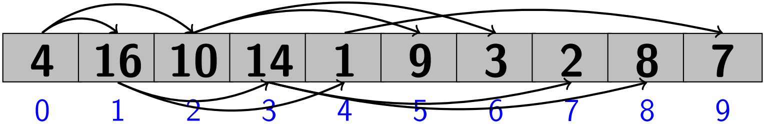
BUILD-MAX-HEAP (V, 10)

Para **i** de 4 **até** 0

    MAX-HEAPIFY (V, 10, **1**);



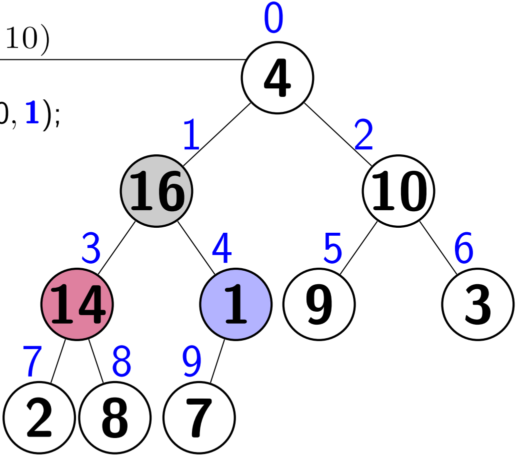
Construindo um Heap Máximo



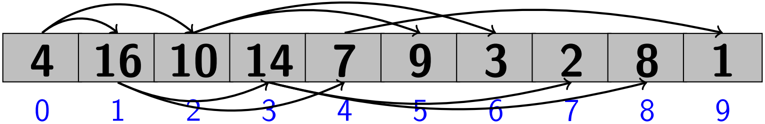
Build-Max-Heap (V, 10)

Para i de 4 **até** 0

  Max-Heapify (V, 10, **1**);



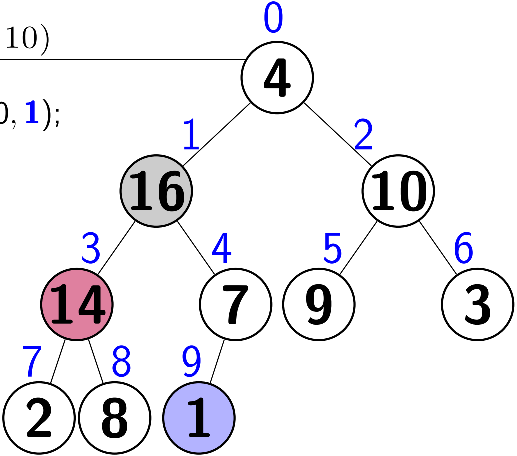
Construindo um Heap Máximo



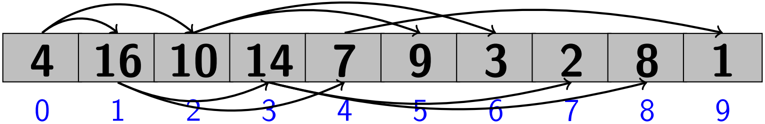
BUILD-MAX-HEAP (V, 10)

Para i de 4 **até** 0

MAX-HEAPIFY (V, 10, **1**);



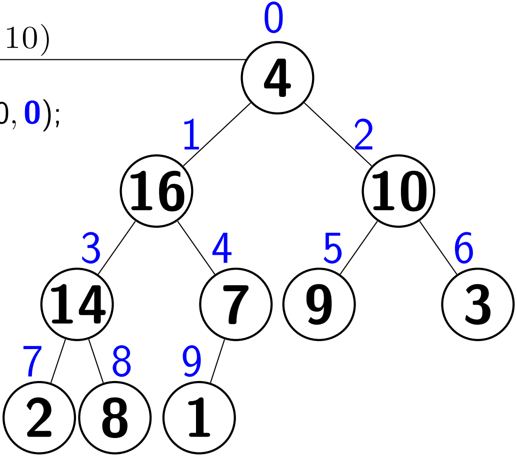
Construindo um Heap Máximo



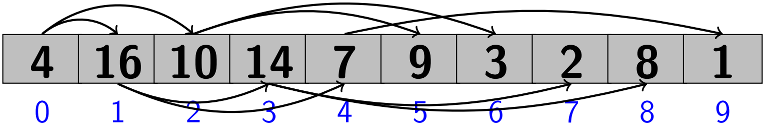
Build-Max-Heap (V, 10)

Para i de 4 **até** 0

    Max-Heapify (V, 10, **0**);



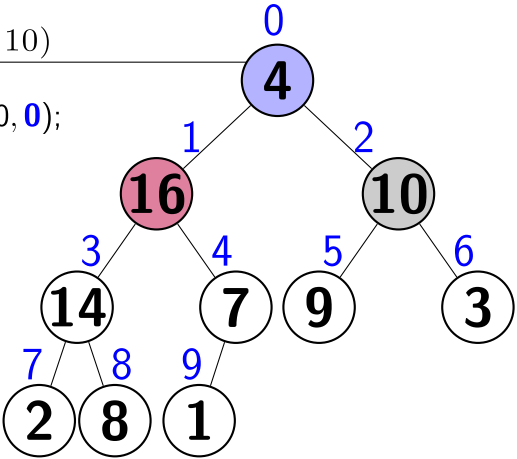
Construindo um Heap Máximo



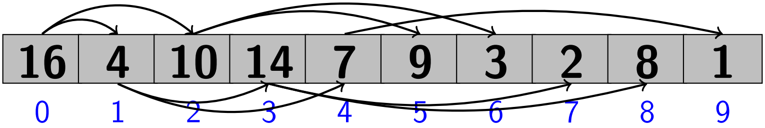
BUILD-MAX-HEAP (V, 10)

Para i de 4 **até** 0

MAX-HEAPIFY ($V, 10, \mathbf{0}$);



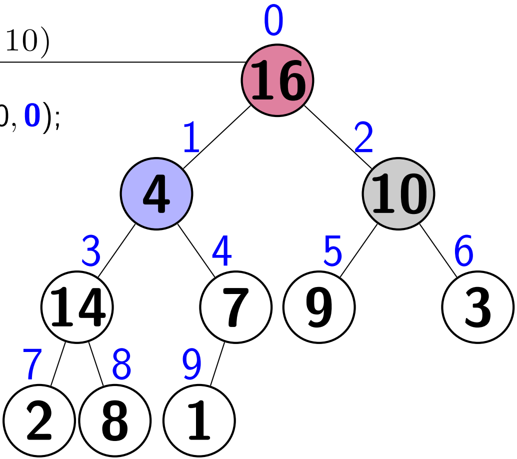
Construindo um Heap Máximo



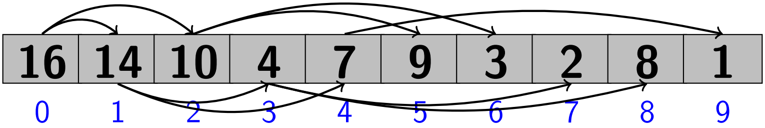
BUILD-MAX-HEAP (V, 10)

Para **i** de 4 **até** 0

MAX-HEAPIFY (V, 10, **0**);

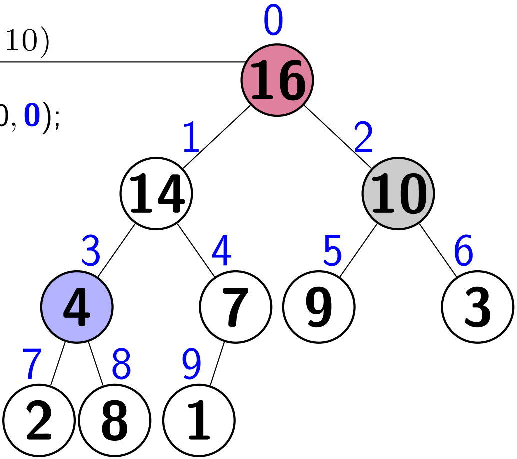


Construindo um Heap Máximo

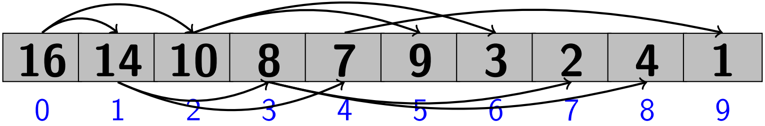


BUILD-MAX-HEAP (V, 10)

Para **i** de 4 **até** 0
    MAX-HEAPIFY (V, 10, **0**);



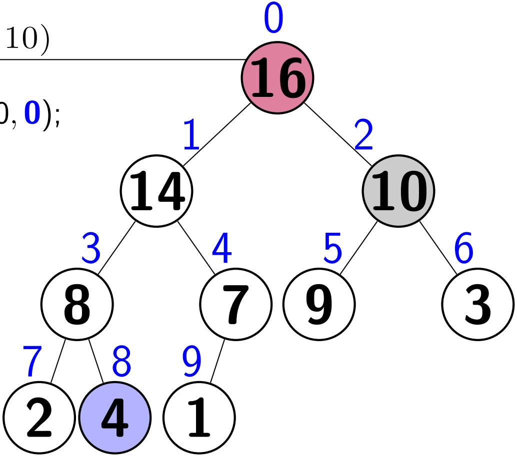
Construindo um Heap Máximo



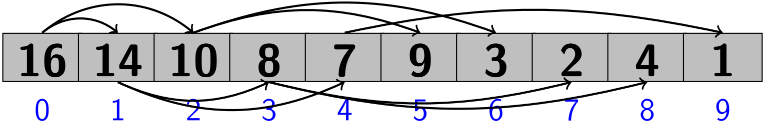
Build-Max-Heap (V, 10)

Para i de 4 **até** 0

Max-Heapify (V, 10, **0**);



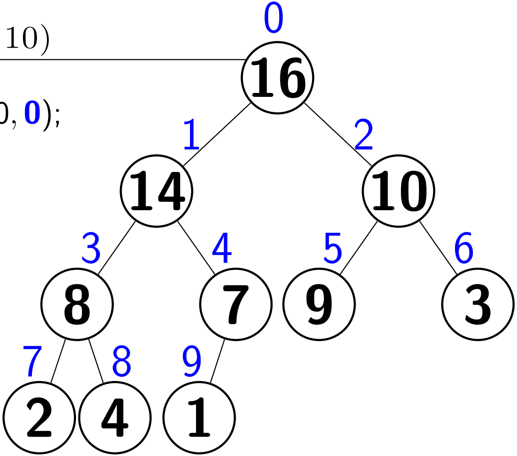
Construindo um Heap Máximo



Build-Max-Heap $(V,\ 10)$

Para **i** de 4 **até** 0

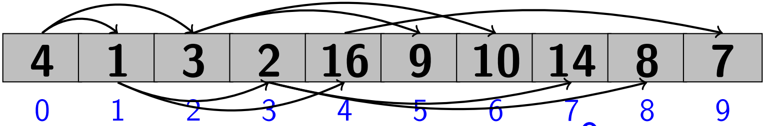
Max-Heapify $(V, 10, \mathbf{0})$;



Complexidade: um limite superior simples para o procedimento BUILD-MAX-HEAP pode ser obtido através do número de $\mathcal{O}(n)$ chamadas do procedimento MAX-HEAPIFY que por sua vez tem um custo de $\mathcal{O}(\log n)$. Deste modo, o tempo de execução do BUILD-MAX-HEAP é $\mathcal{O}(n \log n)$. Embora esse limite seja correto, ele não é restrito, veja em Cormen um limite mais exato.

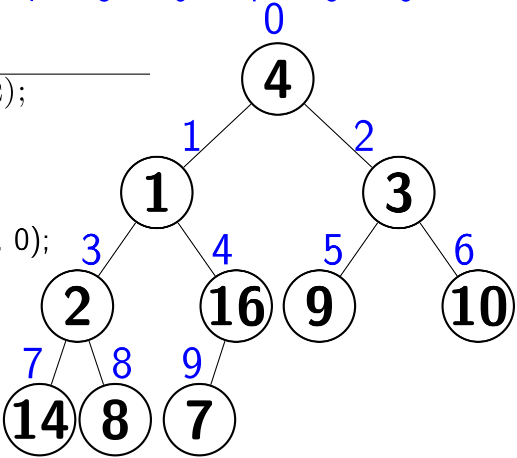
Sumário

Heap-Sort

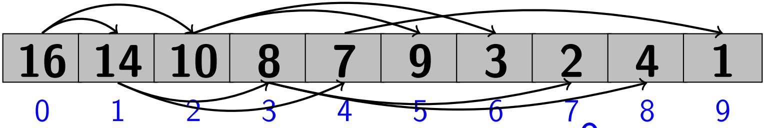


HEAP-SORT (V, SIZE)

BUILD-MAX-HEAP (V,SIZE);
Para i de SIZE-1 **até** 1 **faça**
    $V[0] \leftrightarrow V[\mathbf{i}]$;
    SIZE = i;
    MAX-HEAPIFY (V, SIZE, 0);

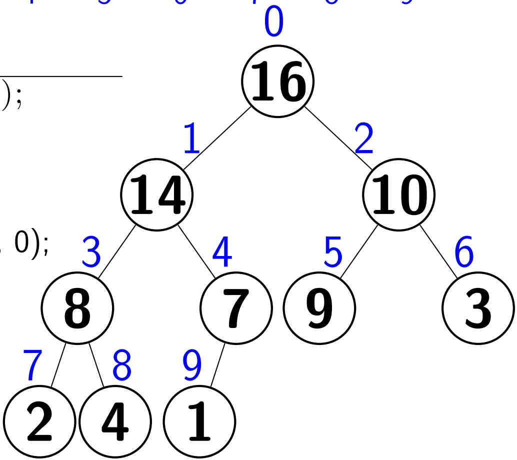


Heap-Sort

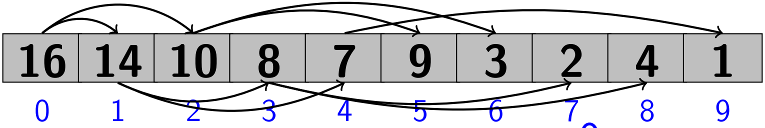


HEAP-SORT (V, 10)

▷BUILD-MAX-HEAP (V,10);
Para **i** de SIZE-1 **até** 1 **faça**
    $V[0] \leftrightarrow V[\mathbf{i}]$;
    SIZE = i;
    MAX-HEAPIFY (V, SIZE, 0);



Heap-Sort



HEAP-SORT (V, 10)

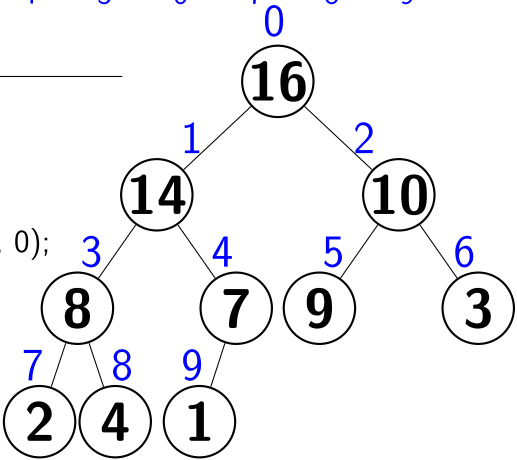
BUILD-MAX-HEAP (V,10);

▷**Para i** de 9 **até** 1 **faça**

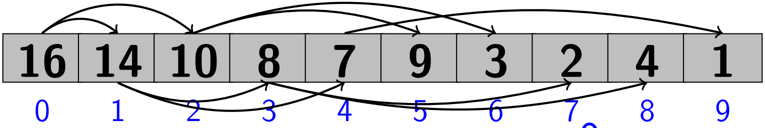
$V[0] \leftrightarrow V[\mathbf{i}]$;

SIZE = i;

MAX-HEAPIFY (V, SIZE, 0);

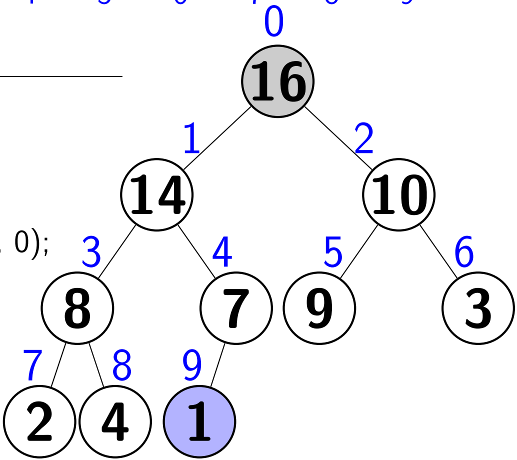


Heap-Sort

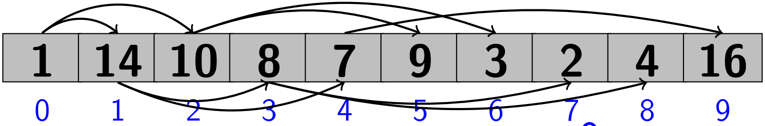


HEAP-SORT (V, 10)

BUILD-MAX-HEAP (V,10);
Para **i** de 9 **até** 1 **faça**
  $\triangleright V[0] \leftrightarrow V[\mathbf{9}]$;
  SIZE = i;
  MAX-HEAPIFY (V, SIZE, 0);

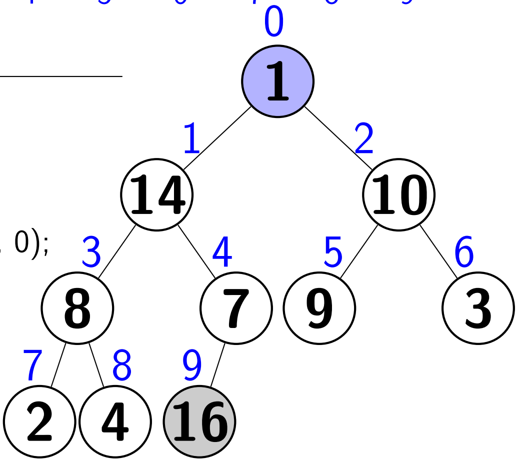


Heap-Sort

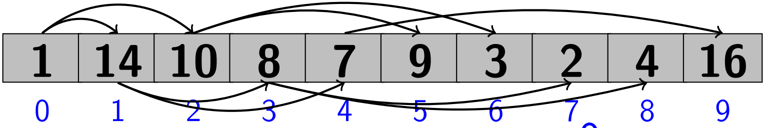


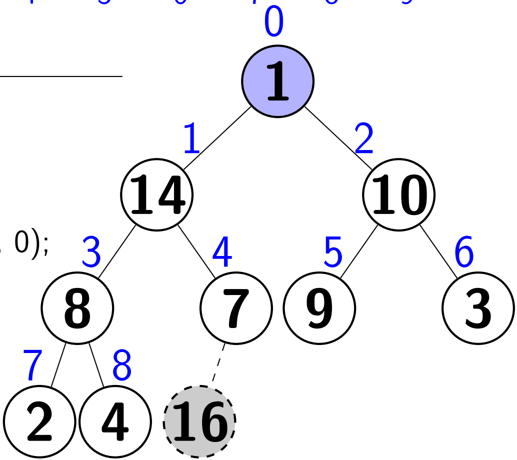
HEAP-SORT (V, 10)

BUILD-MAX-HEAP (V,10);
Para **i** de 9 **até** 1 **faça**
- $\triangleright V[0] \leftrightarrow V[\mathbf{9}]$;
- SIZE = i;
- MAX-HEAPIFY (V, SIZE, 0);



Heap-Sort





HEAP-SORT (V, 10)

BUILD-MAX-HEAP (V,10);

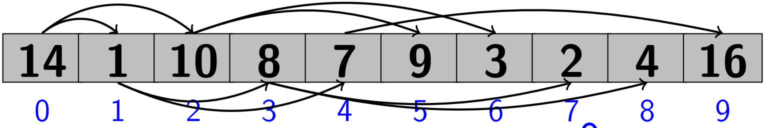
Para i de 9 **até** 1 **faça**

  $V[0] \leftrightarrow V[\mathbf{9}]$;

  ▷SIZE = 9;

  MAX-HEAPIFY (V, SIZE, 0);

Heap-Sort



HEAP-SORT (V, 10)

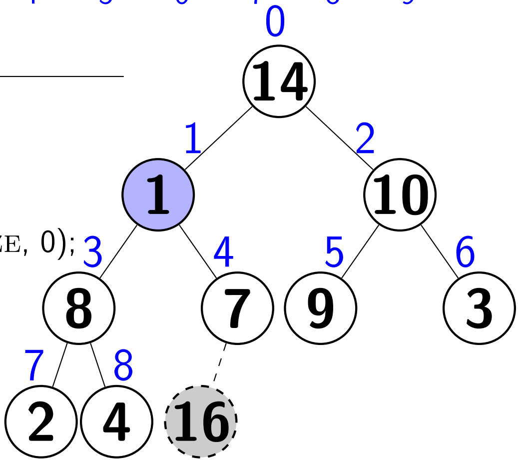
BUILD-MAX-HEAP (V,10);

Para **i** de 9 **até** 1 **faça**

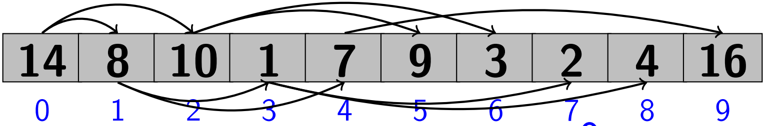
$V[0] \leftrightarrow V[\mathbf{9}]$;

SIZE = 9;

$\triangleright$MAX-HEAPIFY (V, SIZE, 0);



Heap-Sort



Heap-Sort (V, 10)

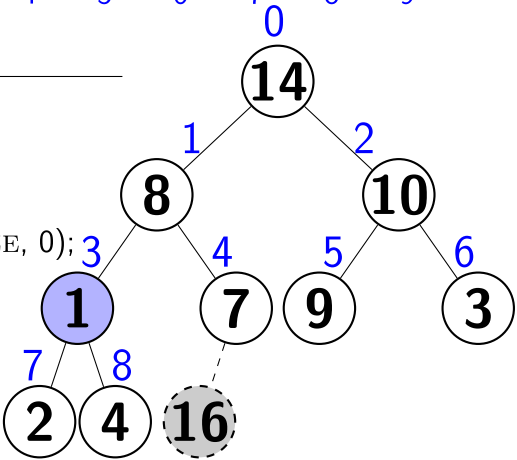
Build-Max-Heap (V,10);

Para **i** de 9 **até** 1 **faça**

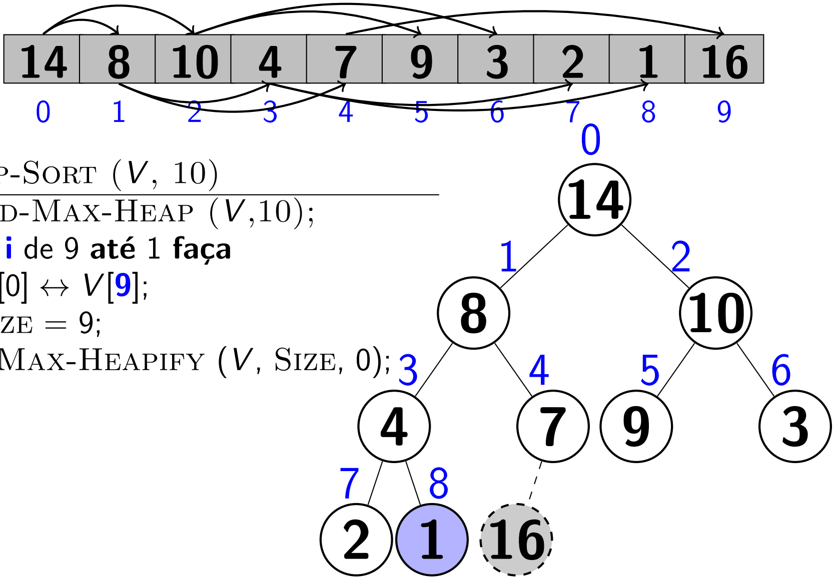
    $V[0] \leftrightarrow V[\mathbf{9}]$;

    Size = 9;

    $\triangleright$Max-Heapify (V, Size, 0);



Heap-Sort



Heap-Sort (V, 10)

Build-Max-Heap (V,10);

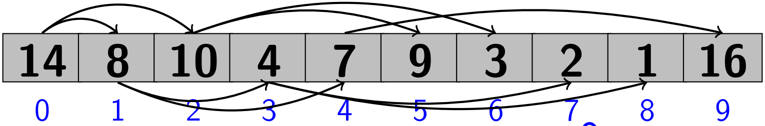
Para **i** de 9 **até** 1 **faça**

$V[0] \leftrightarrow V[\mathbf{9}]$;

Size = 9;

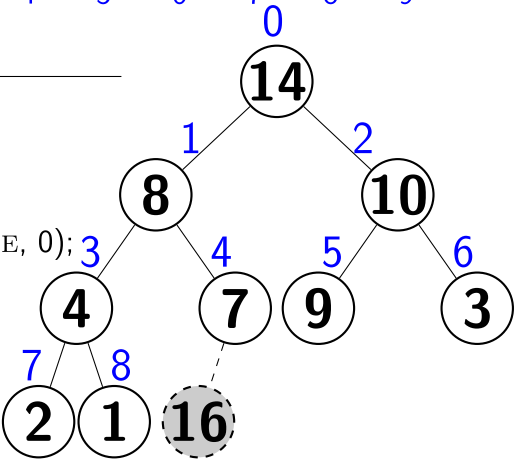
$\triangleright$Max-Heapify (V, Size, 0);

Heap-Sort

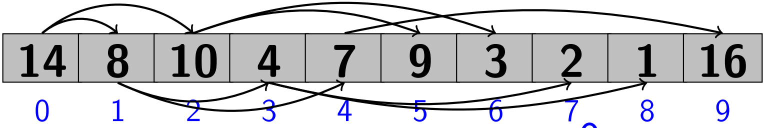


HEAP-SORT (V, 10)

BUILD-MAX-HEAP (V,10);
Para i de 9 **até** 1 **faça**
  $V[0] \leftrightarrow V[\mathbf{9}]$;
  SIZE = 9;
  ▷MAX-HEAPIFY (V, SIZE, 0);



Heap-Sort



HEAP-SORT (V, 10)

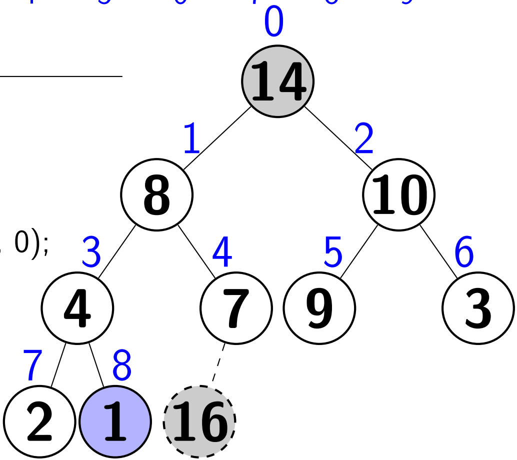
BUILD-MAX-HEAP (V,10);

Para i de 9 **até** 1 **faça**

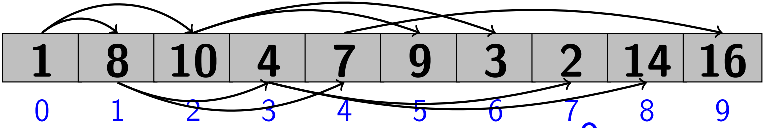
$\triangleright V[0] \leftrightarrow V[\mathbf{8}]$;

SIZE = 9;

MAX-HEAPIFY (V, SIZE, 0);



Heap-Sort



HEAP-SORT (V, 10)

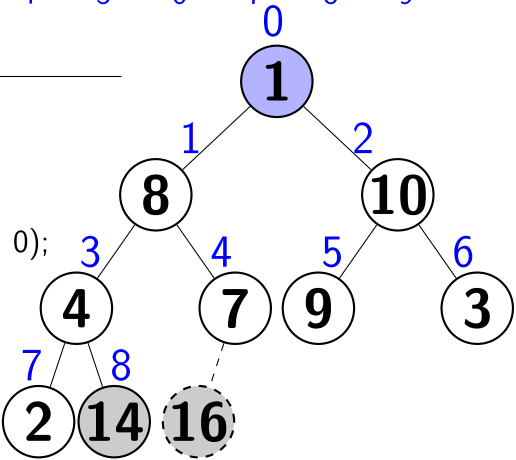
BUILD-MAX-HEAP (V,10);

Para **i** de 9 **até** 1 **faça**

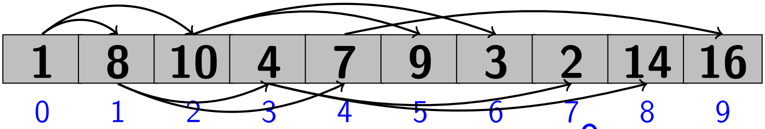
  $\triangleright V[0] \leftrightarrow V[\mathbf{8}]$;

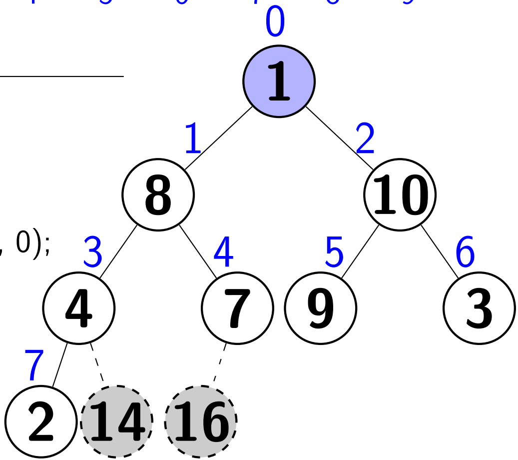
  SIZE = 9;

  MAX-HEAPIFY (V, SIZE, 0);



Heap-Sort

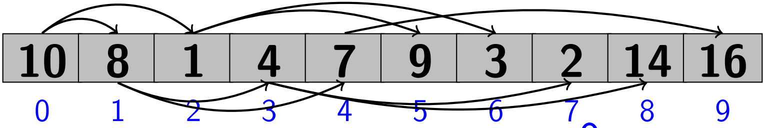




HEAP-SORT (V, 10)

BUILD-MAX-HEAP (V,10);
Para **i** de 9 **até** 1 **faça**
  $V[0] \leftrightarrow V[\mathbf{8}]$;
  ▷SIZE = 8;
  MAX-HEAPIFY (V, SIZE, 0);

Heap-Sort



HEAP-SORT (V, 10)

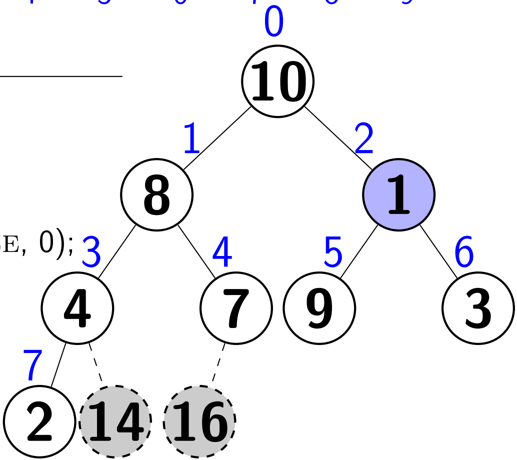
BUILD-MAX-HEAP (V,10);

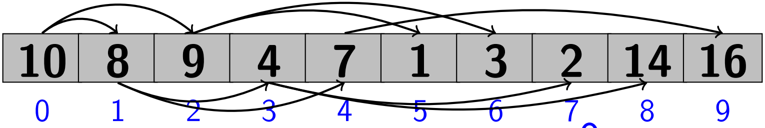
Para **i** de 9 **até** 1 **faça**

$V[0] \leftrightarrow V[\mathbf{8}]$;

SIZE = 8;

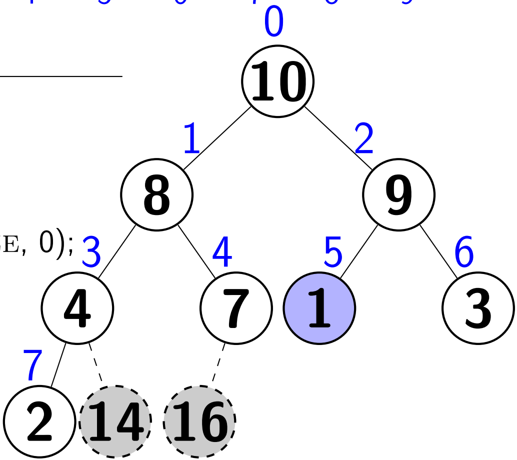
$\triangleright$MAX-HEAPIFY (V, SIZE, 0);



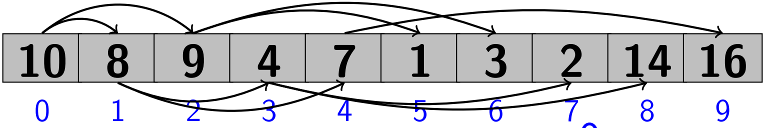


HEAP-SORT (V, 10)

BUILD-MAX-HEAP (V,10);
Para i de 9 **até** 1 **faça**
  $V[0] \leftrightarrow V[8]$;
  SIZE = 8;
  ▷MAX-HEAPIFY (V, SIZE, 0);



Heap-Sort



<u>HEAP-SORT (V, 10)</u>

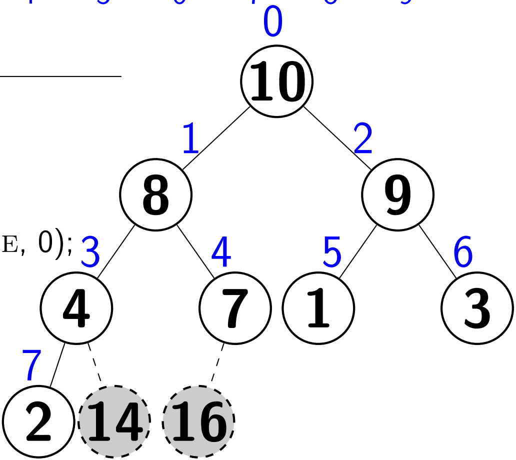
BUILD-MAX-HEAP (V,10);

Para i de 9 **até** 1 **faça**

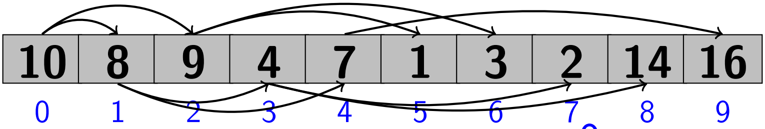
  $V[0] \leftrightarrow V[\mathbf{8}]$;

  SIZE $=$ 8;

  $\triangleright$MAX-HEAPIFY (V, SIZE, 0);

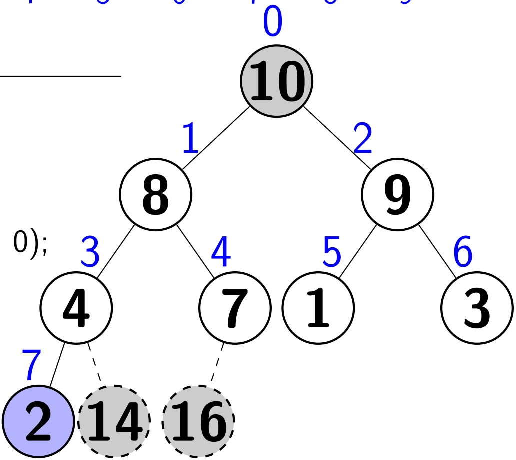


Heap-Sort

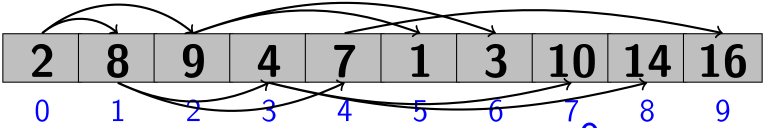


$\textsc{Heap-Sort}\ (V,\ 10)$

$\textsc{Build-Max-Heap}\ (V{,}10);$
Para **i** de 9 **até** 1 **faça**
 $\triangleright V[0] \leftrightarrow V[\mathbf{7}];$
 $\textsc{Size} = 8;$
 $\textsc{Max-Heapify}\ (V,\ \textsc{Size},\ 0);$

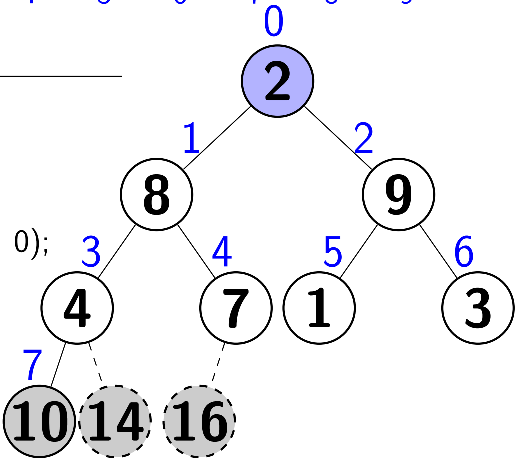


Heap-Sort

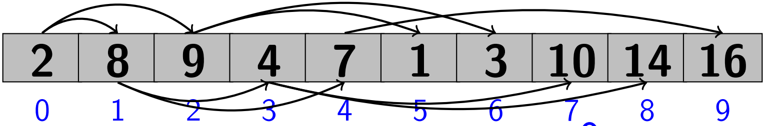


HEAP-SORT (V, 10)

BUILD-MAX-HEAP (V,10);
Para **i** de 9 **até** 1 **faça**
  ▷ $V[0] \leftrightarrow V[\mathbf{7}]$;
  SIZE = 8;
  MAX-HEAPIFY (V, SIZE, 0);



Heap-Sort



HEAP-SORT (V, 10)

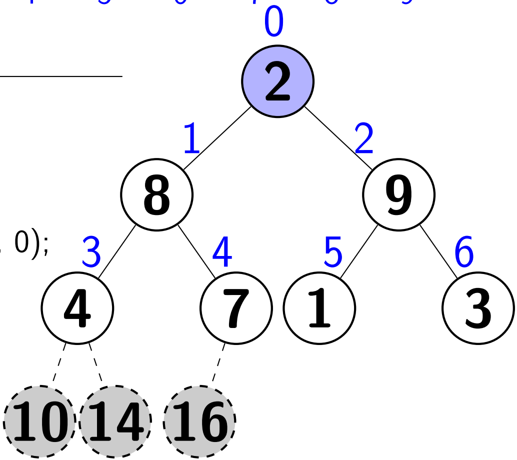
BUILD-MAX-HEAP (V,10);

Para i de 9 **até** 1 **faça**

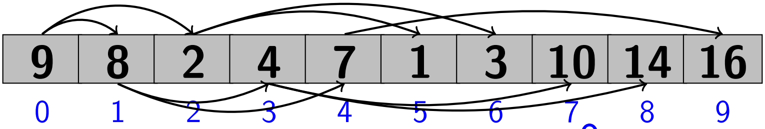
$V[0] \leftrightarrow V[\mathbf{7}]$;

$\triangleright$SIZE = 7;

MAX-HEAPIFY (V, SIZE, 0);



Heap-Sort



HEAP-SORT (V, 10)

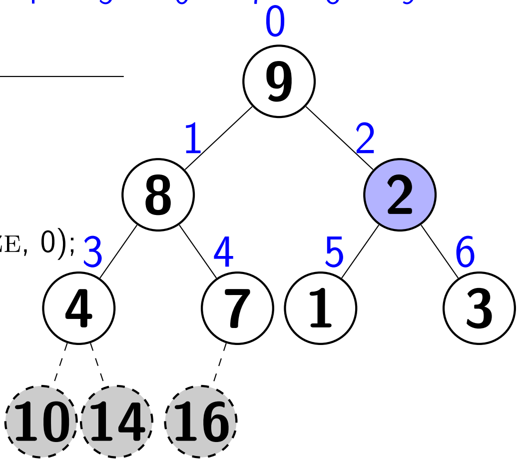
BUILD-MAX-HEAP (V,10);

Para **i** de 9 **até** 1 **faça**

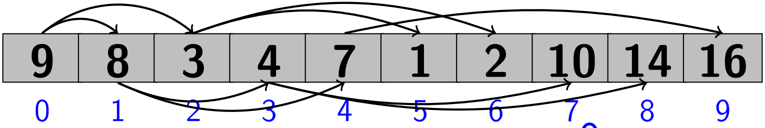
$V[0] \leftrightarrow V[7]$;

SIZE = 7;

$\triangleright$MAX-HEAPIFY (V, SIZE, 0);



Heap-Sort



Heap-Sort (V, 10)

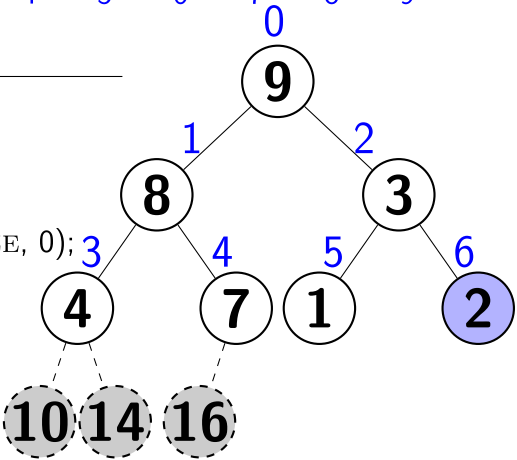
Build-Max-Heap (V,10);

Para **i** de 9 **até** 1 **faça**

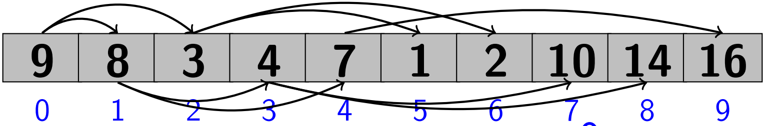
$V[0] \leftrightarrow V[\mathbf{7}]$;

Size = 7;

$\triangleright$Max-Heapify (V, Size, 0);



Heap-Sort



Heap-Sort $(V, 10)$

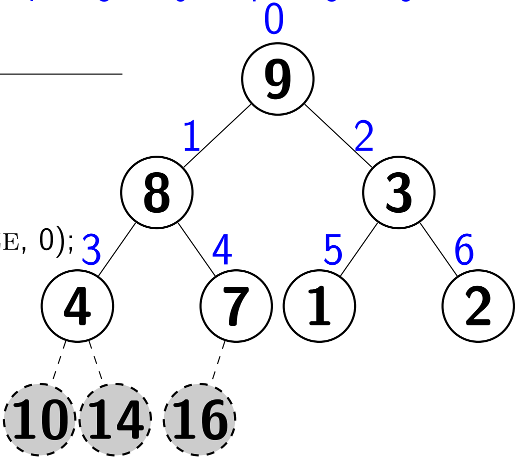
Build-Max-Heap $(V, 10)$;

Para **i** de 9 **até** 1 **faça**

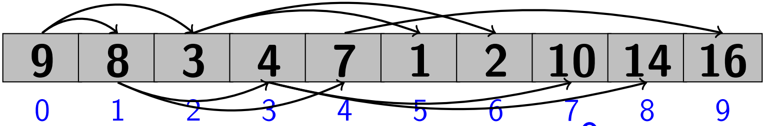
$V[0] \leftrightarrow V[\mathbf{7}]$;

Size = 7;

$\triangleright$Max-Heapify $(V$, Size, $0)$;

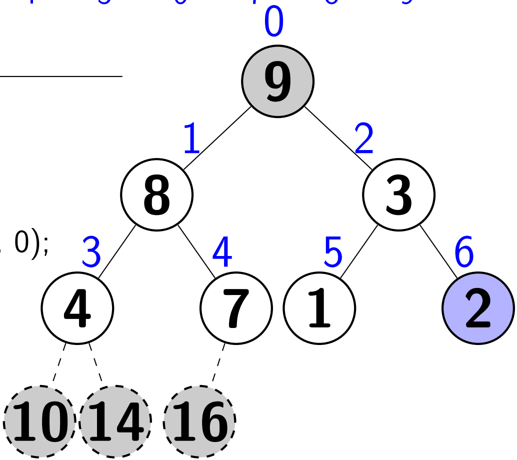


Heap-Sort

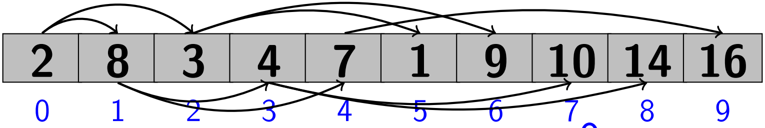


HEAP-SORT (V, 10)

BUILD-MAX-HEAP (V,10);
Para i de 9 **até** 1 **faça**
$\triangleright V[0] \leftrightarrow V[\mathbf{6}]$;
SIZE = 7;
MAX-HEAPIFY (V, SIZE, 0);



Heap-Sort



HEAP-SORT (V, 10)

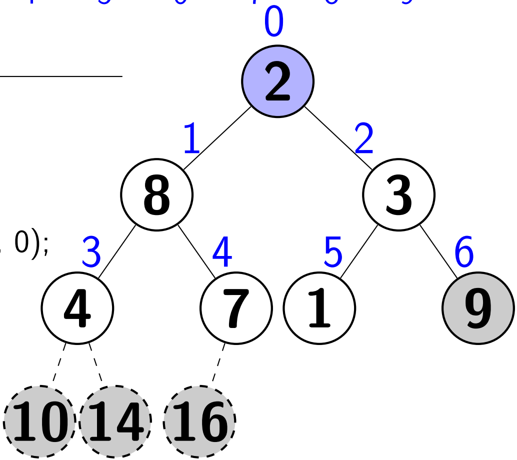
BUILD-MAX-HEAP (V,10);

Para i de 9 **até** 1 **faça**

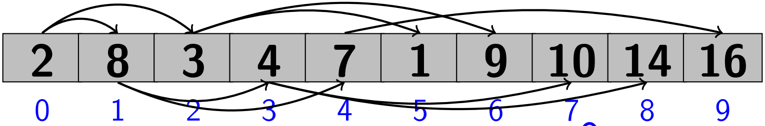
$\triangleright V[0] \leftrightarrow V[\mathbf{6}]$;

SIZE = 7;

MAX-HEAPIFY (V, SIZE, 0);

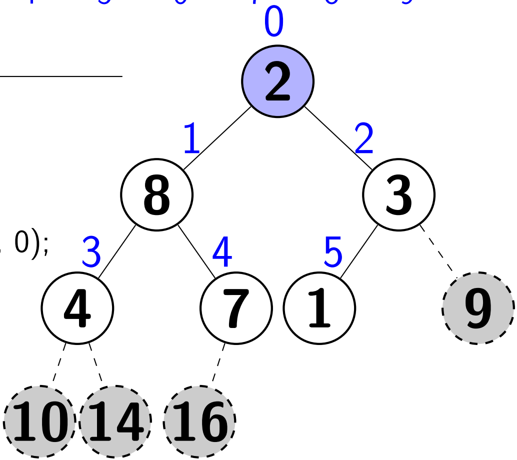


Heap-Sort

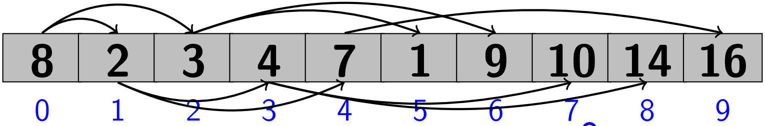


HEAP-SORT (V, 10)

BUILD-MAX-HEAP (V,10);
Para **i** de 9 **até** 1 **faça**
    $V[0] \leftrightarrow V[\mathbf{6}]$;
    ▷SIZE = 6;
    MAX-HEAPIFY (V, SIZE, 0);

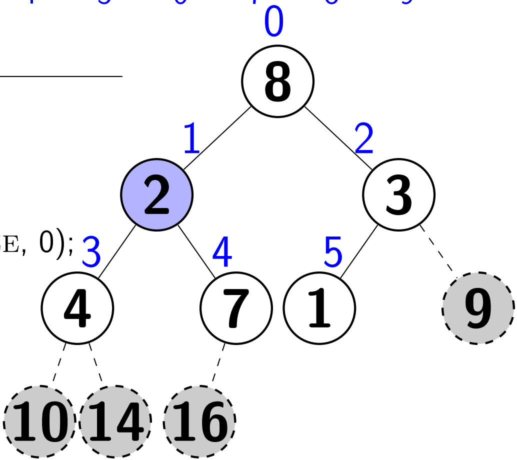


Heap-Sort

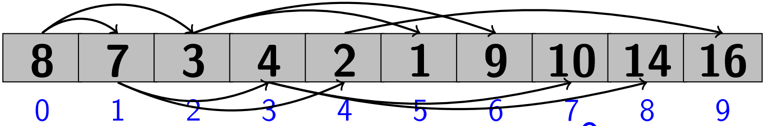


<u>HEAP-SORT (V, 10)</u>

BUILD-MAX-HEAP (V,10);
Para **i** de 9 **até** 1 **faça**
  $V[0] \leftrightarrow V[\mathbf{6}]$;
  SIZE $=$ 6;
  $\triangleright$MAX-HEAPIFY (V, SIZE, 0);



Heap-Sort



HEAP-SORT (V, 10)

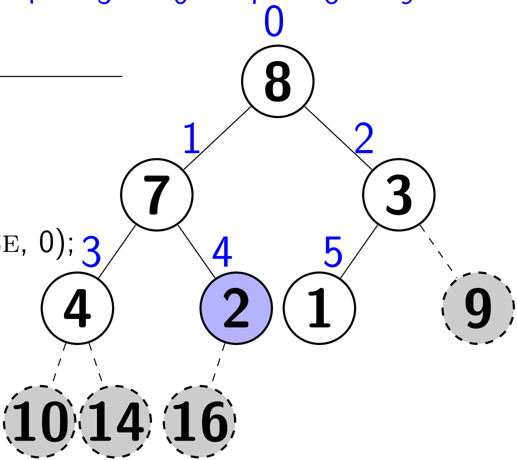
BUILD-MAX-HEAP (V,10);

Para i de 9 **até** 1 **faça**

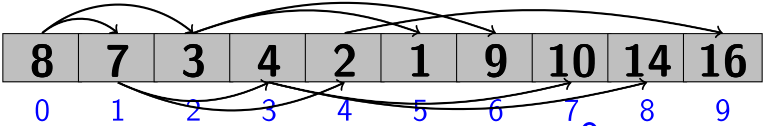
$V[0] \leftrightarrow V[\mathbf{6}]$;

SIZE = 6;

▷MAX-HEAPIFY (V, SIZE, 0);



Heap-Sort



HEAP-SORT (V, 10)

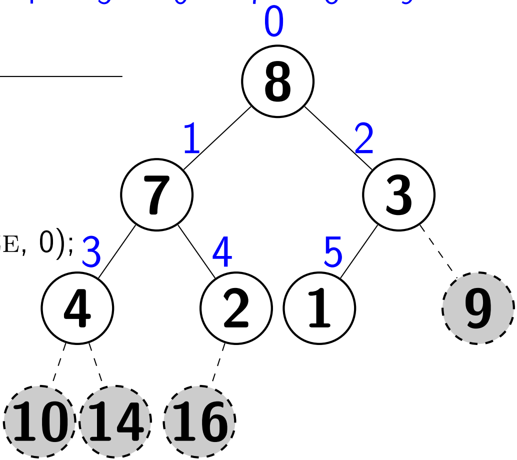
BUILD-MAX-HEAP (V,10);

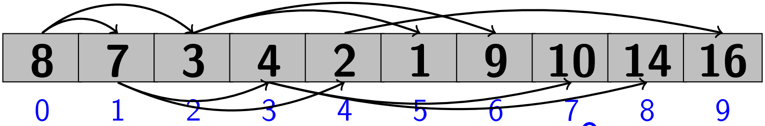
Para i de 9 **até** 1 **faça**

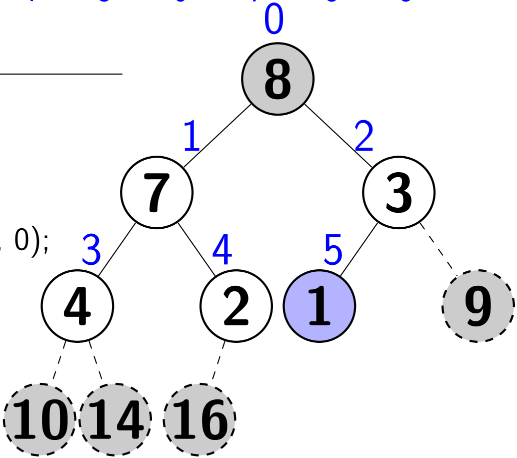
 $V[0] \leftrightarrow V[\mathbf{6}]$;

 SIZE = 6;

 ▷MAX-HEAPIFY (V, SIZE, 0);







Heap-Sort (V, 10)

Build-Max-Heap (V,10);

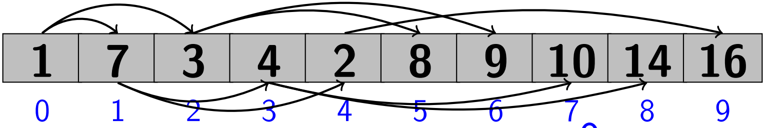
Para **i** de 9 **até** 1 **faça**

$\triangleright V[0] \leftrightarrow V[\mathbf{5}]$;

Size = 6;

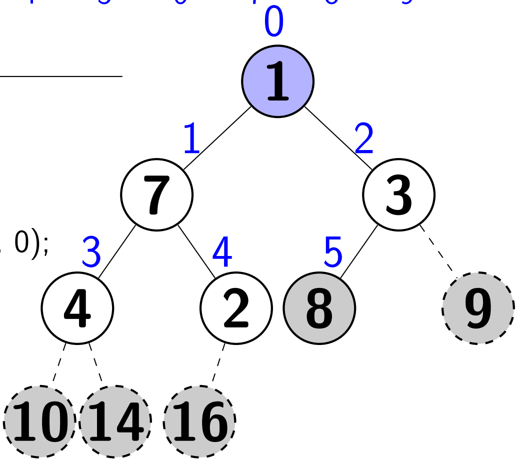
Max-Heapify (V, Size, 0);

Heap-Sort

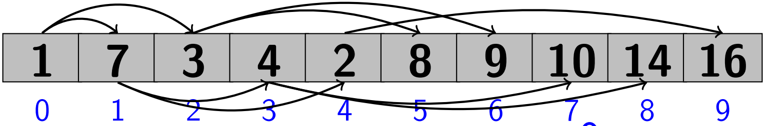


HEAP-SORT (V, 10)

BUILD-MAX-HEAP (V,10);
Para i de 9 **até** 1 **faça**
  $\triangleright V[0] \leftrightarrow V[\mathbf{5}]$;
  SIZE = 6;
  MAX-HEAPIFY (V, SIZE, 0);

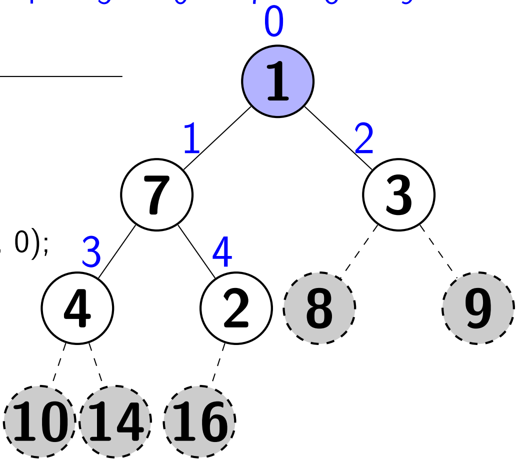


Heap-Sort

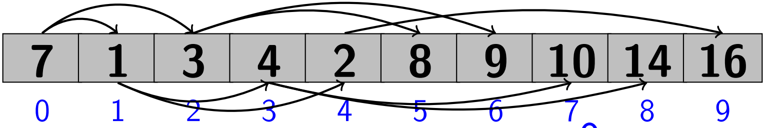


HEAP-SORT (V, 10)

BUILD-MAX-HEAP (V,10);
Para i de 9 **até** 1 **faça**
  $V[0] \leftrightarrow V[\mathbf{5}]$;
  ▷SIZE = 5;
  MAX-HEAPIFY (V, SIZE, 0);

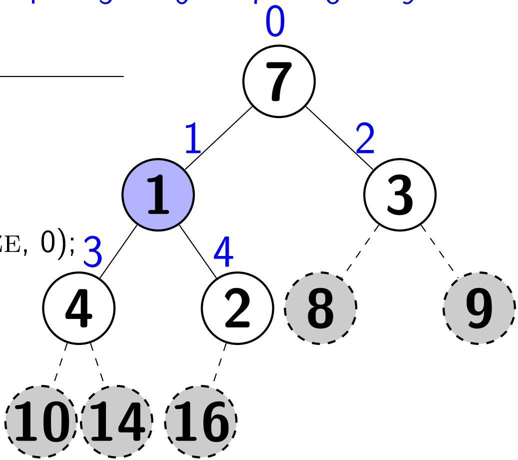


Heap-Sort

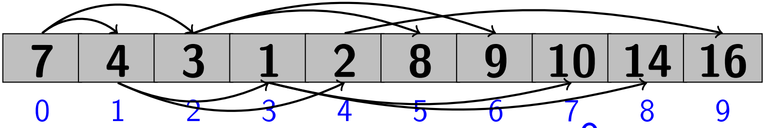


HEAP-SORT (V, 10)

BUILD-MAX-HEAP (V,10);
Para i de 9 **até** 1 **faça**
  $V[0] \leftrightarrow V[\mathbf{5}]$;
  SIZE $=$ 5;
  $\triangleright$MAX-HEAPIFY (V, SIZE, 0);



Heap-Sort



HEAP-SORT (V, 10)

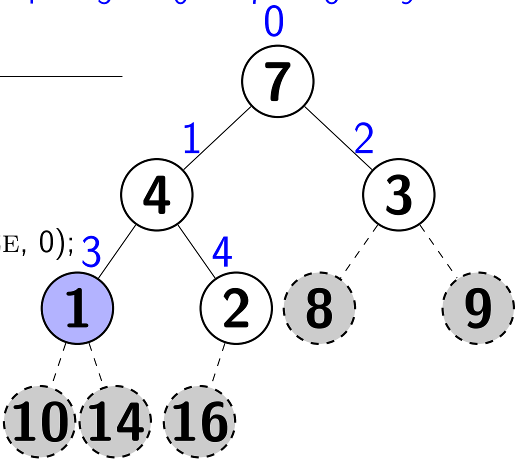
BUILD-MAX-HEAP (V,10);

Para i de 9 **até** 1 **faça**

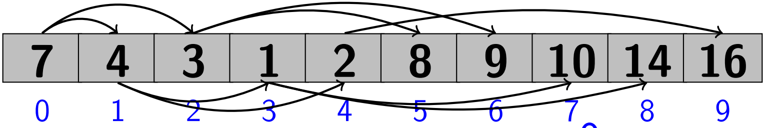
$V[0] \leftrightarrow V[\mathbf{5}]$;

SIZE = 5;

$\triangleright$MAX-HEAPIFY (V, SIZE, 0);



Heap-Sort



HEAP-SORT (V, 10)

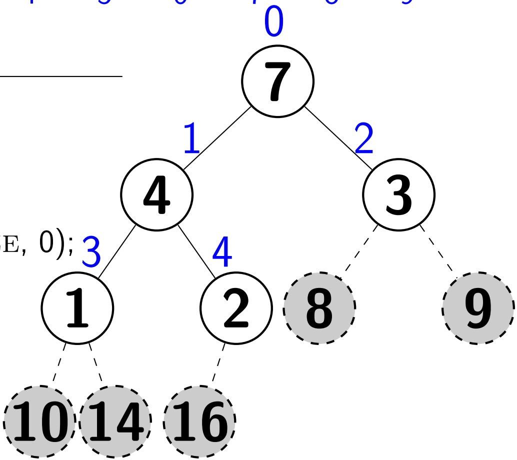
BUILD-MAX-HEAP (V,10);

Para **i** de 9 **até** 1 **faça**

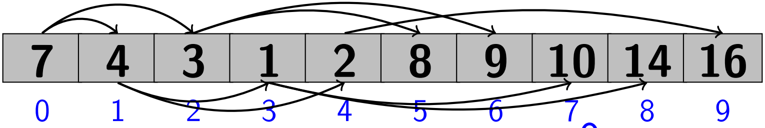
$V[0] \leftrightarrow V[\mathbf{5}]$;

SIZE = 5;

▷MAX-HEAPIFY (V, SIZE, 0);

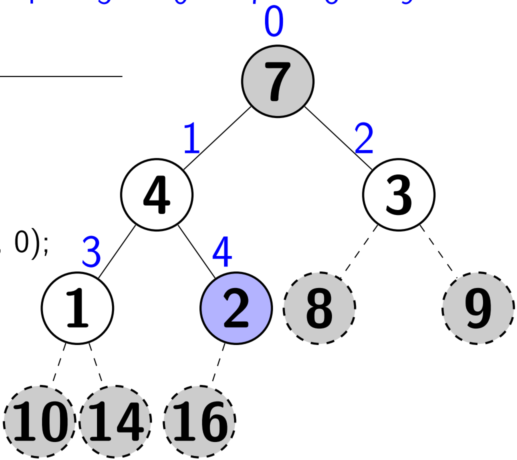


Heap-Sort

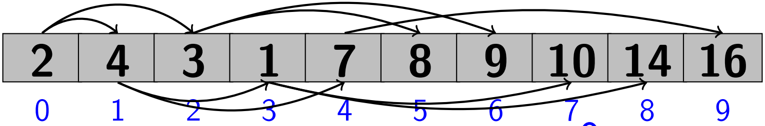


HEAP-SORT $(V,\ 10)$

BUILD-MAX-HEAP $(V, 10)$;
Para **i** de 9 **até** 1 **faça**
  $\triangleright\, V[0] \leftrightarrow V[\mathbf{4}]$;
  SIZE = 5;
  MAX-HEAPIFY $(V,$ SIZE, $0)$;



Heap-Sort



HEAP-SORT (V, 10)

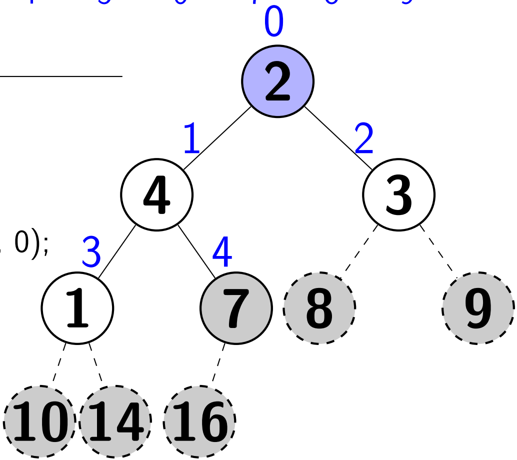
BUILD-MAX-HEAP (V,10);

Para **i** de 9 **até** 1 **faça**

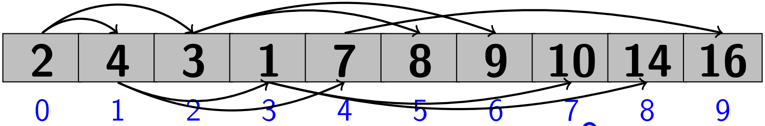
  ▷ $V[0] \leftrightarrow V[\mathbf{4}]$;

  SIZE = 5;

  MAX-HEAPIFY (V, SIZE, 0);

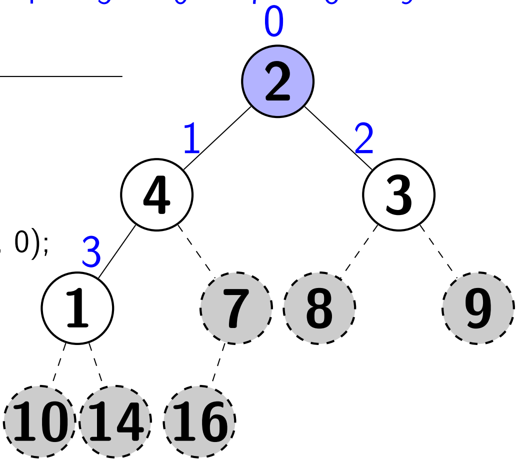


Heap-Sort

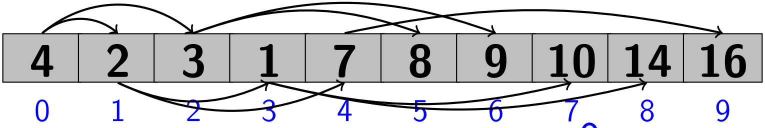


HEAP-SORT (V, 10)

BUILD-MAX-HEAP (V,10);
Para **i** de 9 **até** 1 **faça**
    $V[0] \leftrightarrow V[\mathbf{4}]$;
    ▷SIZE = 4;
    MAX-HEAPIFY (V, SIZE, 0);

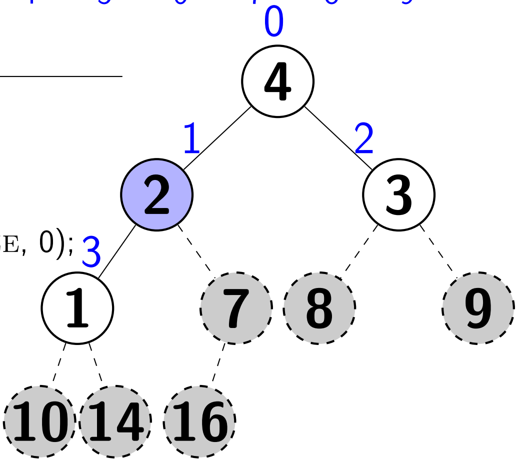


Heap-Sort

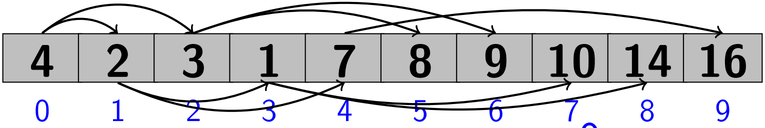


HEAP-SORT (V, 10)

BUILD-MAX-HEAP (V,10);
Para **i** de 9 **até** 1 **faça**
    $V[0] \leftrightarrow V[\mathbf{4}]$;
    SIZE = 4;
    ▷MAX-HEAPIFY (V, SIZE, 0);

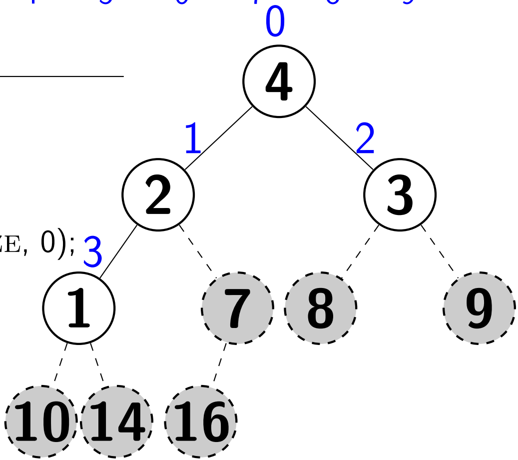


Heap-Sort

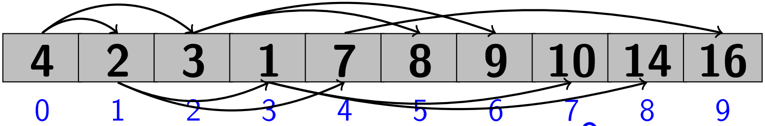


<u>HEAP-SORT (*V*, 10)</u>

BUILD-MAX-HEAP (*V*,10);
Para i de 9 **até** 1 **faça**
  $V[0] \leftrightarrow V[\mathbf{4}]$;
  SIZE = 4;
  ▷MAX-HEAPIFY (*V*, SIZE, 0);

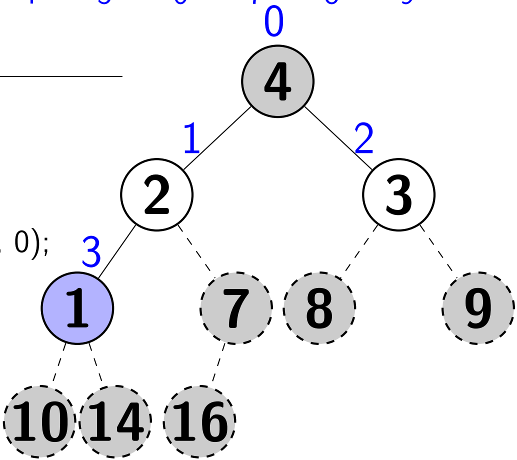


Heap-Sort

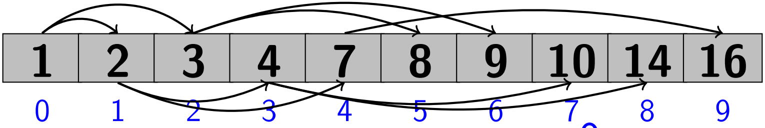


HEAP-SORT (V, 10)

BUILD-MAX-HEAP (V,10);
Para **i** de 9 **até** 1 **faça**
  $\triangleright V[0] \leftrightarrow V[\mathbf{3}]$;
  SIZE = 4;
  MAX-HEAPIFY (V, SIZE, 0);

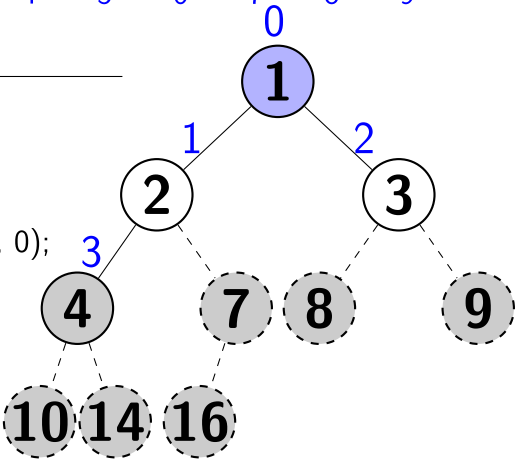


Heap-Sort

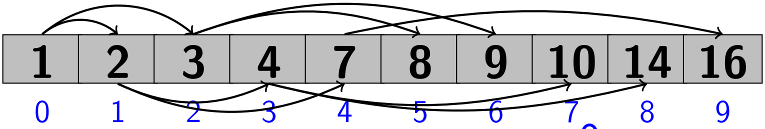


HEAP-SORT (V, 10)

BUILD-MAX-HEAP (V,10);
Para **i** de 9 **até** 1 **faça**
$\triangleright V[0] \leftrightarrow V[\mathbf{3}]$;
SIZE = 4;
MAX-HEAPIFY (V, SIZE, 0);

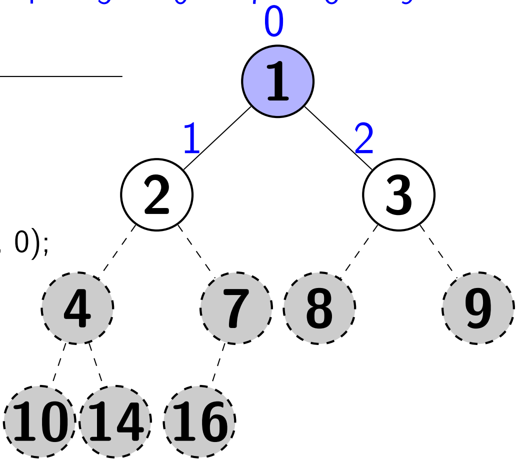


Heap-Sort

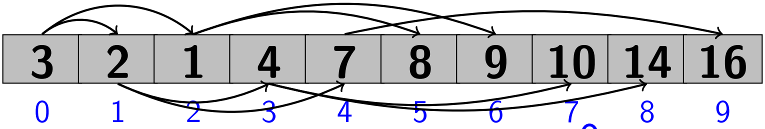


Heap-Sort (V, 10)

Build-Max-Heap (V,10);
Para i de 9 **até** 1 **faça**
    $V[0] \leftrightarrow V[\mathbf{3}]$;
    ▷Size = 3;
    Max-Heapify (V, Size, 0);

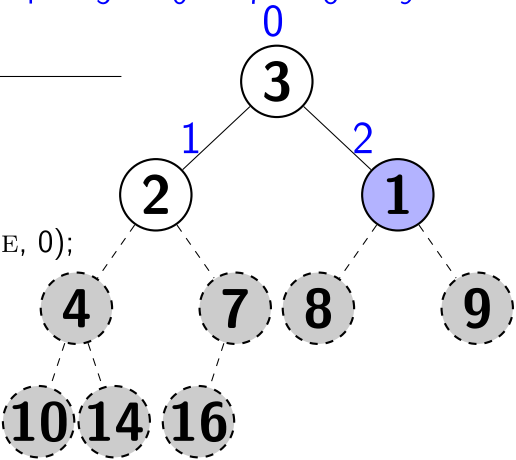


Heap-Sort

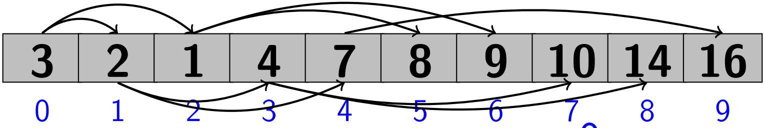


HEAP-SORT (V, 10)

BUILD-MAX-HEAP (V,10);
Para **i** de 9 **até** 1 **faça**
    $V[0] \leftrightarrow V[\mathbf{3}]$;
    SIZE = 3;
    $\triangleright$MAX-HEAPIFY (V, SIZE, 0);



Heap-Sort



HEAP-SORT (V, 10)

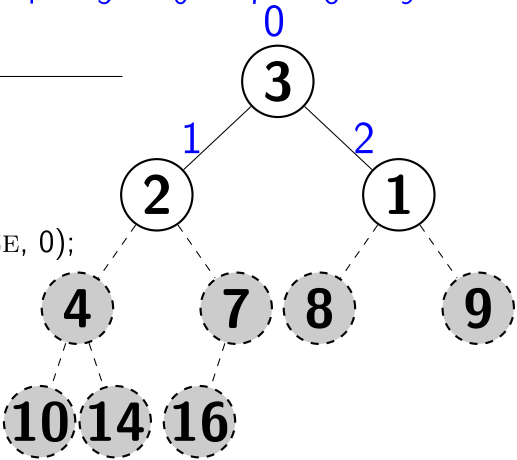
BUILD-MAX-HEAP (V,10);

Para i de 9 **até** 1 **faça**

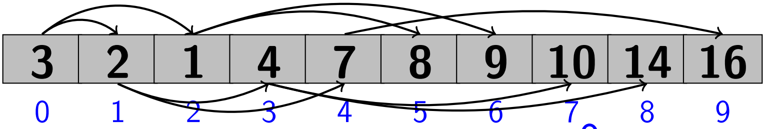
$V[0] \leftrightarrow V[\mathbf{3}]$;

SIZE = 3;

$\triangleright$MAX-HEAPIFY (V, SIZE, 0);



Heap-Sort



Heap-Sort (V, 10)

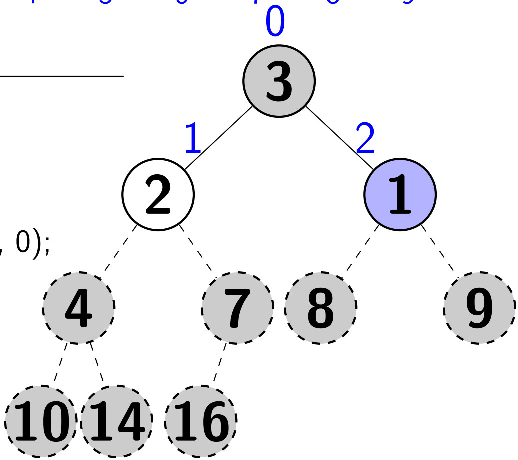
Build-Max-Heap (V,10);

Para **i** de 9 **até** 1 **faça**

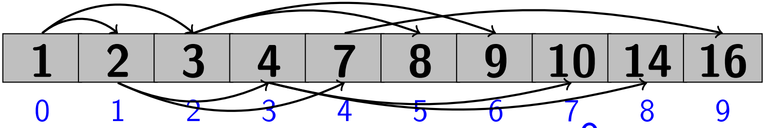
$\triangleright V[0] \leftrightarrow V[\mathbf{2}]$;

Size = 3;

Max-Heapify (V, Size, 0);

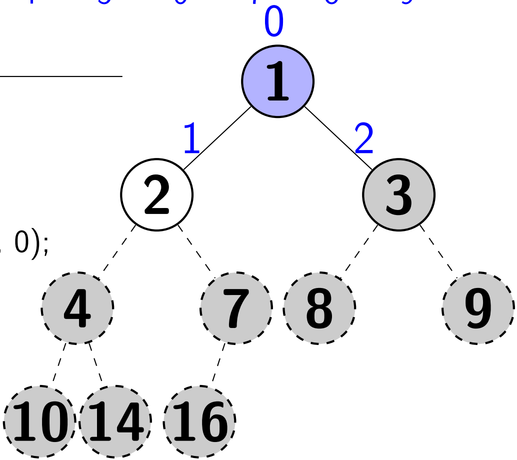


Heap-Sort

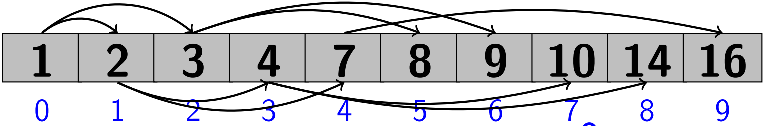


HEAP-SORT (V, 10)

BUILD-MAX-HEAP (V,10);
Para i de 9 **até** 1 **faça**
$\triangleright V[0] \leftrightarrow V[\mathbf{2}]$;
SIZE = 3;
MAX-HEAPIFY (V, SIZE, 0);

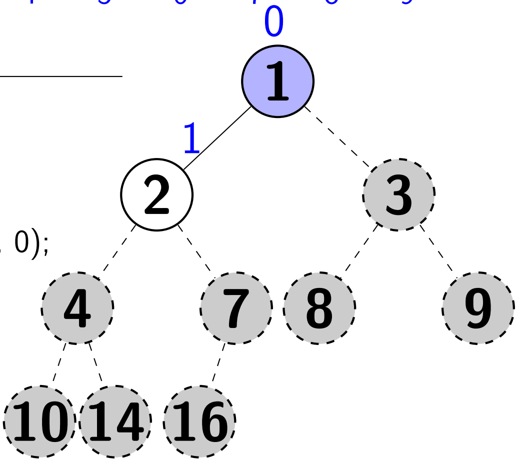


Heap-Sort

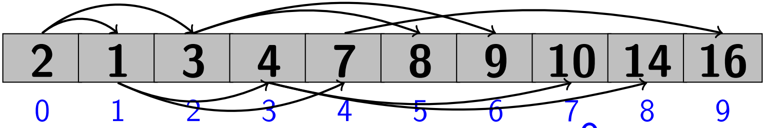


HEAP-SORT (V, 10)

BUILD-MAX-HEAP (V,10);
Para i de 9 **até** 1 **faça**
  $V[0] \leftrightarrow V[\mathbf{2}]$;
  $\triangleright$SIZE = 2;
  MAX-HEAPIFY (V, SIZE, 0);



Heap-Sort



HEAP-SORT (V, 10)

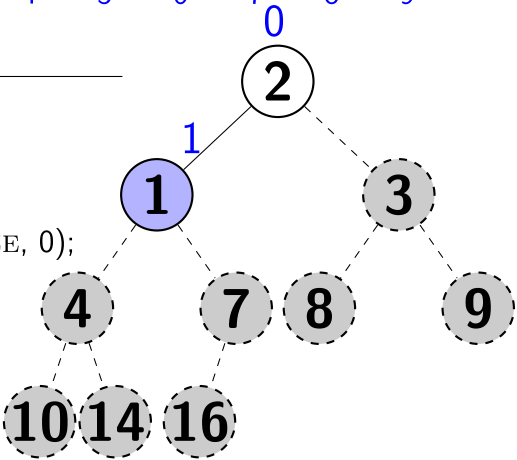
BUILD-MAX-HEAP (V,10);

Para **i** de 9 **até** 1 **faça**

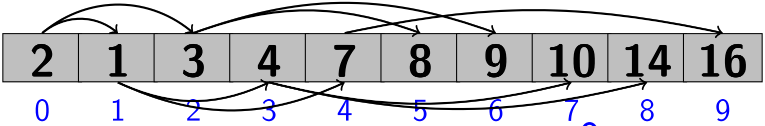
$V[0] \leftrightarrow V[\mathbf{2}]$;

SIZE $= 2$;

$\triangleright$MAX-HEAPIFY (V, SIZE, 0);

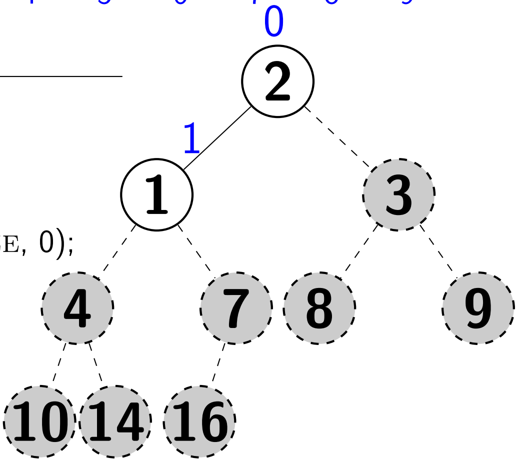


Heap-Sort

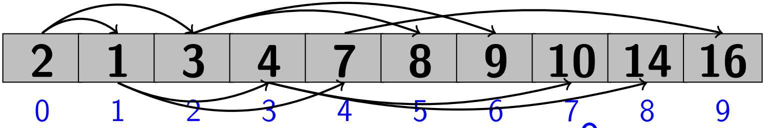


HEAP-SORT (V, 10)

BUILD-MAX-HEAP (V,10);
Para i de 9 **até** 1 **faça**
    $V[0] \leftrightarrow V[\mathbf{2}]$;
    SIZE = 2;
    ▷MAX-HEAPIFY (V, SIZE, 0);

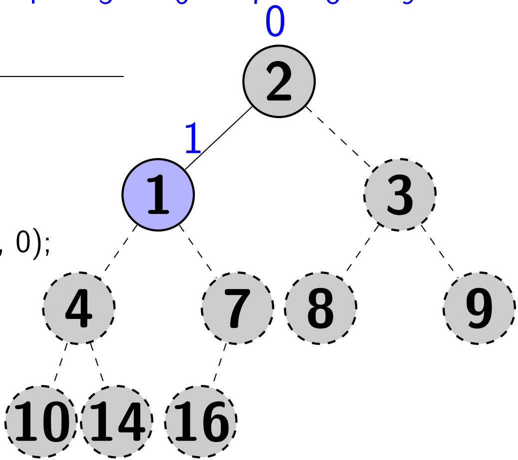


Heap-Sort

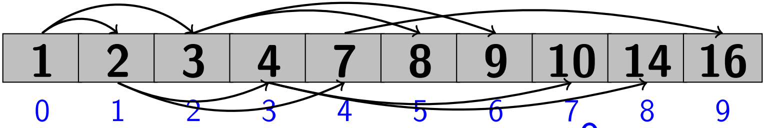


Heap-Sort (V, 10)

Build-Max-Heap (V,10);
Para **i** de 9 **até** 1 **faça**
  ▷ $V[0] \leftrightarrow V[1]$;
  Size = 2;
  Max-Heapify (V, Size, 0);

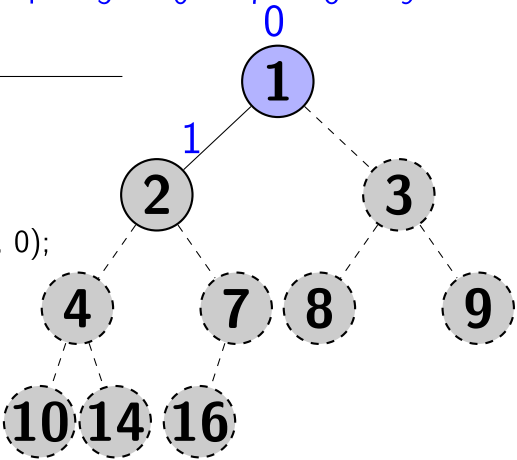


Heap-Sort

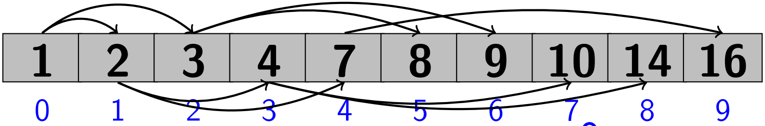


Heap-Sort (V, 10)

Build-Max-Heap (V,10);
Para i de 9 **até** 1 **faça**
  $\triangleright V[0] \leftrightarrow V[1]$;
  Size = 2;
  Max-Heapify (V, Size, 0);

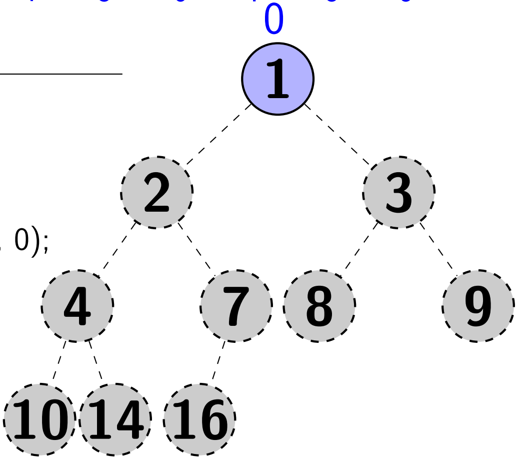


Heap-Sort

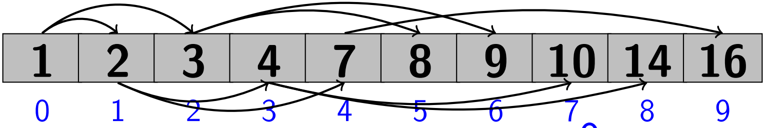


HEAP-SORT (V, 10)

BUILD-MAX-HEAP (V,10);
Para **i** de 9 **até** 1 **faça**
  $V[0] \leftrightarrow V[\mathbf{1}]$;
  $\triangleright$SIZE = 1;
  MAX-HEAPIFY (V, SIZE, 0);

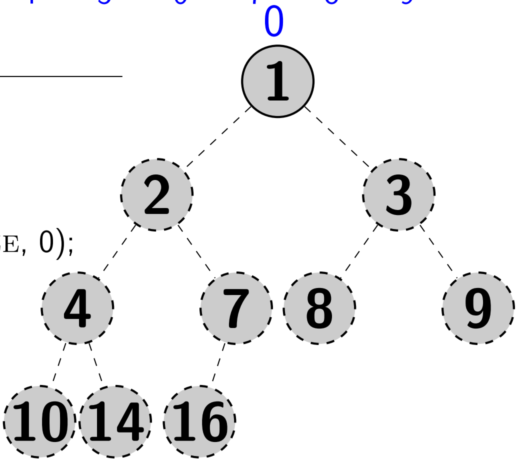


Heap-Sort



Heap-Sort (V, 10)

Build-Max-Heap (V,10);
Para i de 9 **até** 1 **faça**
  $V[0] \leftrightarrow V[\mathbf{1}]$;
  Size = 1;
  ▷Max-Heapify (V, Size, 0);



Complexidade: o algoritmo **Heap-Sort** possui complexidade $\mathcal{O}(n \log n)$, pois a chamada a $\textsc{Build-Max-Heap}$ demora tempo $\mathcal{O}(n \log n)$, e cada uma das $n-1$ chamadas a $\textsc{Max-Heapify}$ demora tempo $\mathcal{O}(\log n)$.

Referências

`Algoritmos: Teoria e prática. Cormen et al.`