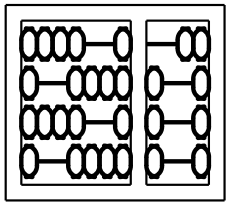


Universidade Estadual de Campinas - UNICAMP
Instituto de Computação - IC

Filas de Prioridade

Tópicos

1. Filas de Prioridade
2. *Heaps*
3. *Heapsort*

Filas de Prioridade

- TAD em que cada elemento do conjunto de dados está associado a uma *prioridade*.

– As operações básicas relativas a uma *fila de prioridade* são:

1. *Construção* de uma fila de prioridade a partir de um conjunto com n itens.
2. *Seleção* do elemento de maior (menor) prioridade.
3. *Inserção* de um novo elemento.
4. *Remoção* do elemento de maior (menor) prioridade.
5. *Alteração* do elemento de maior (menor) prioridade.

Filas de Prioridade

- TAD em que cada elemento do conjunto de dados está associado a uma *prioridade*.

– As operações básicas relativas a uma *fila de prioridade* são:

1. *Construção* de uma fila de prioridade a partir de um conjunto com n itens.
2. *Seleção* do elemento de maior (menor) prioridade.
3. *Inserção* de um novo elemento.
4. *Remoção* do elemento de maior (menor) prioridade.
5. *Alteração* do elemento de maior (menor) prioridade.

Filas de Prioridade

• TAD em que cada elemento do conjunto de dados está associado a uma *prioridade*.

– As operações básicas relativas a uma *fila de prioridade* são:

1. *Construção* de uma fila de prioridade a partir de um conjunto com n itens.
2. *Seleção* do elemento de maior (menor) prioridade.
3. *Inserção* de um novo elemento.
4. *Remoção* do elemento de maior (menor) prioridade.
5. *Alteração* do elemento de maior (menor) prioridade.

Filas de Prioridade

• TAD em que cada elemento do conjunto de dados está associado a uma *prioridade*.

– As operações básicas relativas a uma *fila de prioridade* são:

1. *Construção* de uma fila de prioridade a partir de um conjunto com n itens.
2. *Seleção* do elemento de maior (menor) prioridade.
3. *Inserção* de um novo elemento.
4. *Remoção* do elemento de maior (menor) prioridade.
5. *Alteração* do elemento de maior (menor) prioridade.

Filas de Prioridade

• TAD em que cada elemento do conjunto de dados está associado a uma *prioridade*.

– As operações básicas relativas a uma *fila de prioridade* são:

1. *Construção* de uma fila de prioridade a partir de um conjunto com n itens.
2. *Seleção* do elemento de maior (menor) prioridade.
3. *Inserção* de um novo elemento.
4. *Remoção* do elemento de maior (menor) prioridade.
5. *Alteração* do elemento de maior (menor) prioridade.

Filas de Prioridade

• TAD em que cada elemento do conjunto de dados está associado a uma *prioridade*.

– As operações básicas relativas a uma *fila de prioridade* são:

1. *Construção* de uma fila de prioridade a partir de um conjunto com n itens.
2. *Seleção* do elemento de maior (menor) prioridade.
3. *Inserção* de um novo elemento.
4. *Remoção* do elemento de maior (menor) prioridade.
5. *Alteração* do elemento de maior (menor) prioridade.

Filas de Prioridade

• TAD em que cada elemento do conjunto de dados está associado a uma *prioridade*.

– As operações básicas relativas a uma *fila de prioridade* são:

1. *Construção* de uma fila de prioridade a partir de um conjunto com n itens.
2. *Seleção* do elemento de maior (menor) prioridade.
3. *Inserção* de um novo elemento.
4. *Remoção* do elemento de maior (menor) prioridade.
5. *Alteração* do elemento de maior (menor) prioridade.

Filas de Prioridade

• Estruturalmente, uma *fila de prioridade* pode ser vista como uma árvore binária com as seguintes propriedades:

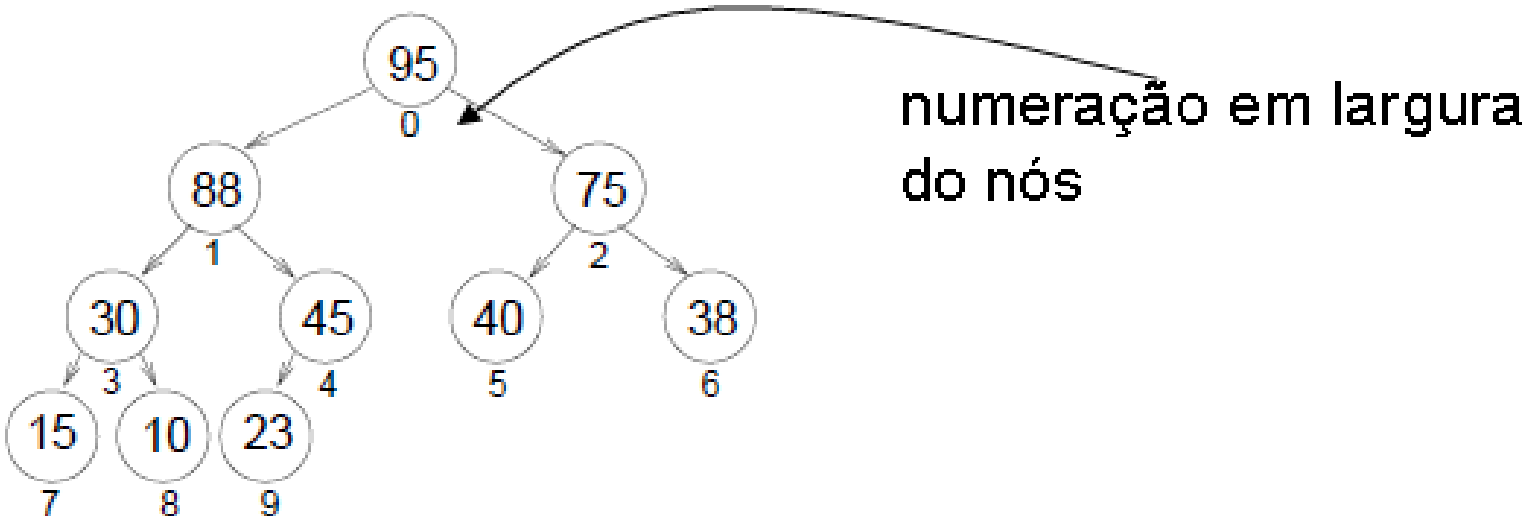
1. É uma árvore binária *completa* ou *quase completa*.
2. Em cada nó da árvore, o valor da chave é maior (menor) do que os valores das chaves dos filhos.

Filas de Prioridade

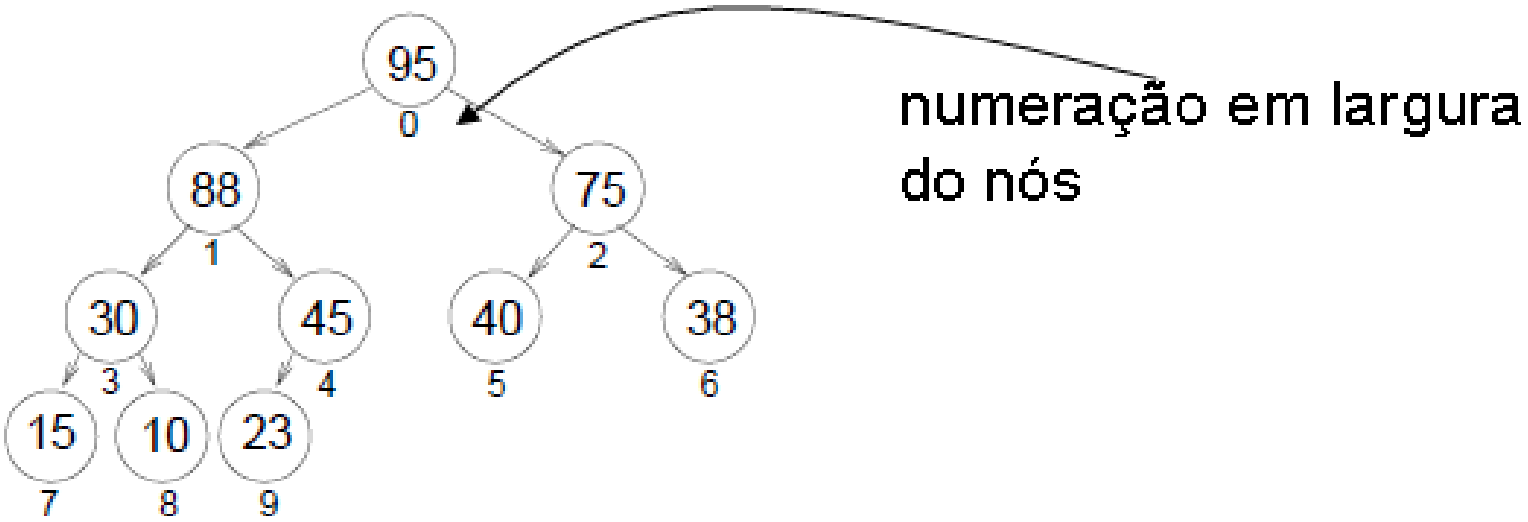
• Estruturalmente, uma *fila de prioridade* pode ser vista como uma árvore binária com as seguintes propriedades:

1. É uma árvore binária *completa* ou *quase completa*.
2. Em cada nó da árvore, o valor da chave é maior (menor) do que os valores das chaves dos filhos.

- Exemplo:


95
0
88
1
75
2
30
3
45
4
40
5
38
6
15
7
10
8
23
9
numeração em largura
do nós

- Exemplo:



- Heap: Implementação em vetor

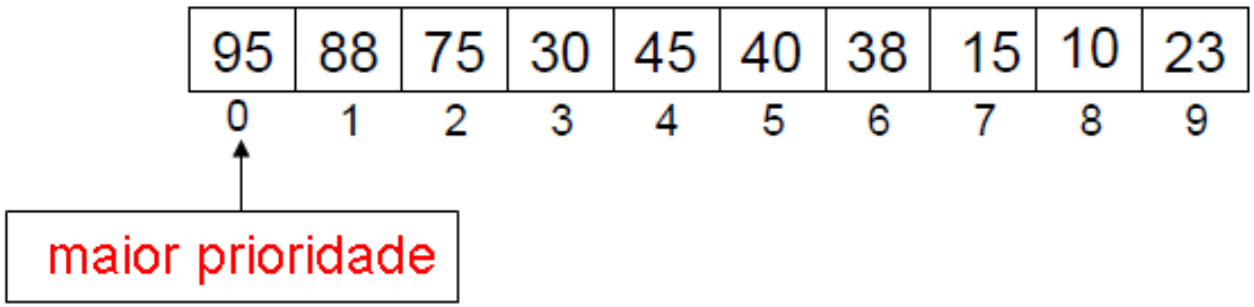
95	88	75	30	45	40	38	15	10	23
0	1	2	3	4	5	6	7	8	9

• Definição: Um *heap* é uma lista linear composta de elementos com chaves $s_0, ..., s_{n-1}$, satisfazendo a seguinte propriedade:

$$s_{\lfloor \frac{i-1}{2} \rfloor} \geq s_i, \quad \forall\ 0 < i \leq n-1 \quad \text{(max-heap)}$$

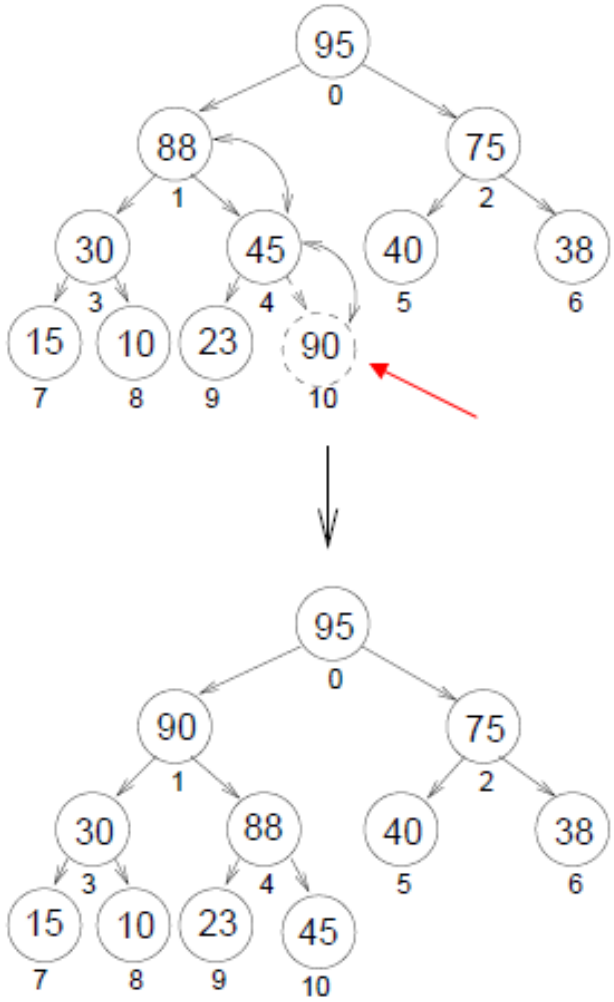
ou

$$s_{\lfloor \frac{i-1}{2} \rfloor} \leq s_i, \quad \forall\ 0 < i \leq n-1 \quad \text{(min-heap)}$$



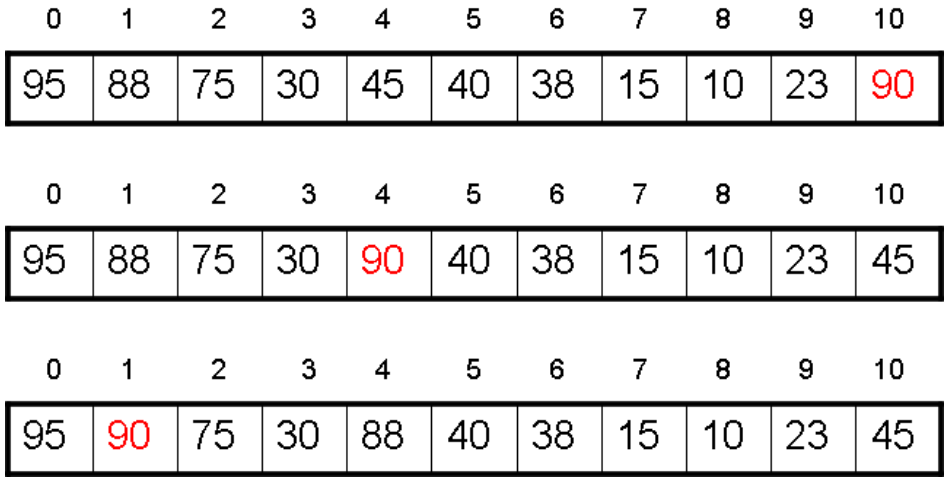
- Operações básicas sobre um heap:

– Subida ($O(logn)$):



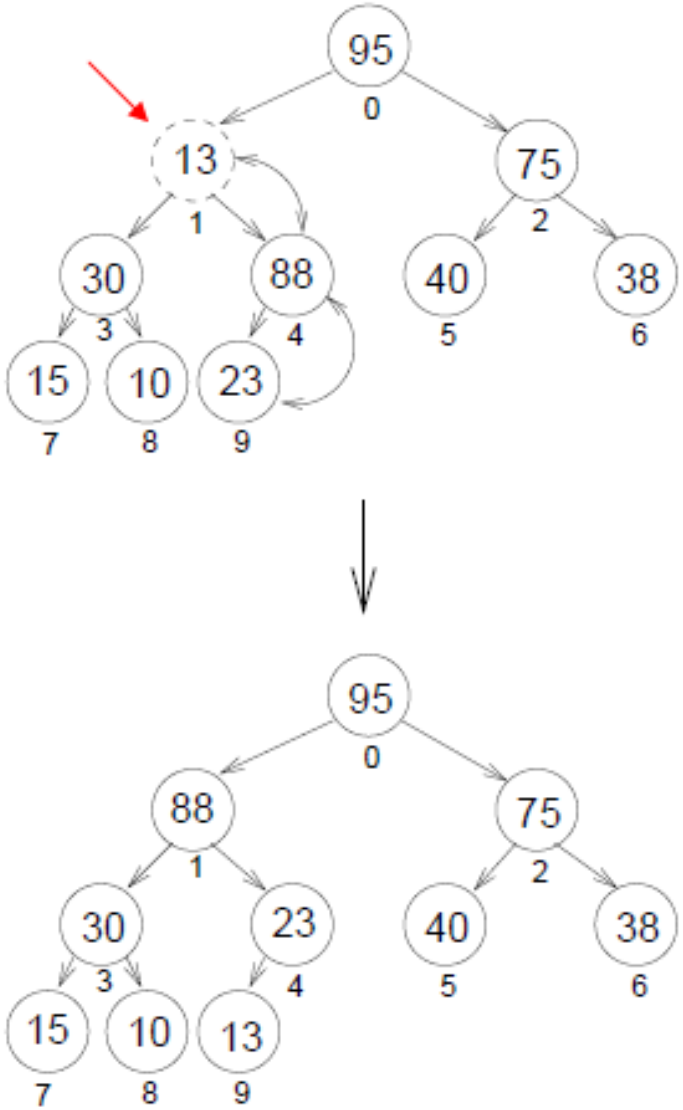
- Operações básicas sobre um heap:

– Subida em vetor ($O(logn)$):



- Operações básicas sobre um heap:

– Descida ($O(logn)$):



- Operações básicas sobre um heap:

– Descida em vetor ($O(logn)$):

0	1	2	3	4	5	6	7	8	9
95	13	75	30	88	40	38	15	10	23

0	1	2	3	4	5	6	7	8	9
95	88	75	30	13	40	38	15	10	23

0	1	2	3	4	5	6	7	8	9
95	88	75	30	23	40	38	15	10	13

- Operação *Sobe* em C:

```c
#define TAM_MAX 50

typedef struct {
  T vetor[TAM_MAX];
  int tam;
} Heap;
```

```c
void Sobe(Heap *h, int m) {
  int j = (m-1)/2;
  T x = (*h).vetor[m];

  while ((m>0) && ((*h).vetor[j]<x)) {
    (*h).vetor[m] = (*h).vetor[j];
    m = j;
    j = (j-1)/2;
  }
  (*h).vetor[m] = x;
} /* Sobe */
```

- Operação *Desce* em C:

```
void Desce(Heap *h, int m) {
  int k = 2*m+1;
  T x = (*h).vetor[m];
  while (k<(*h).tam) {
    if ((k<((*h).tam)-1) && ((*h).vetor[k]<(*h).vetor[k+1]))
      k++;
    if (x<(*h).vetor[k]) {
      (*h).vetor[m] = (*h).vetor[k];
      m = k;
      k = 2*k+1;
    } else
      break;
  }
  (*h).vetor[m] = x;
} /* Desce */
```

- Construção de *heaps* (recebe vetor *h* e constrói heap): $O(nlogn)$

```
void ConstroiHeap1(Heap *h) {
  int i;
  for (i=1; i<(*h).tam; i++)
    Sobe(h,i);
} /* ConstroiHeap1 */
```

```
void ConstroiHeap2(Heap *h) {
  int i;
  for (i=((*h).tam-2)/2; i>=0; i--)
    Desce(h,i);
} /* ConstroiHeap2 */
```

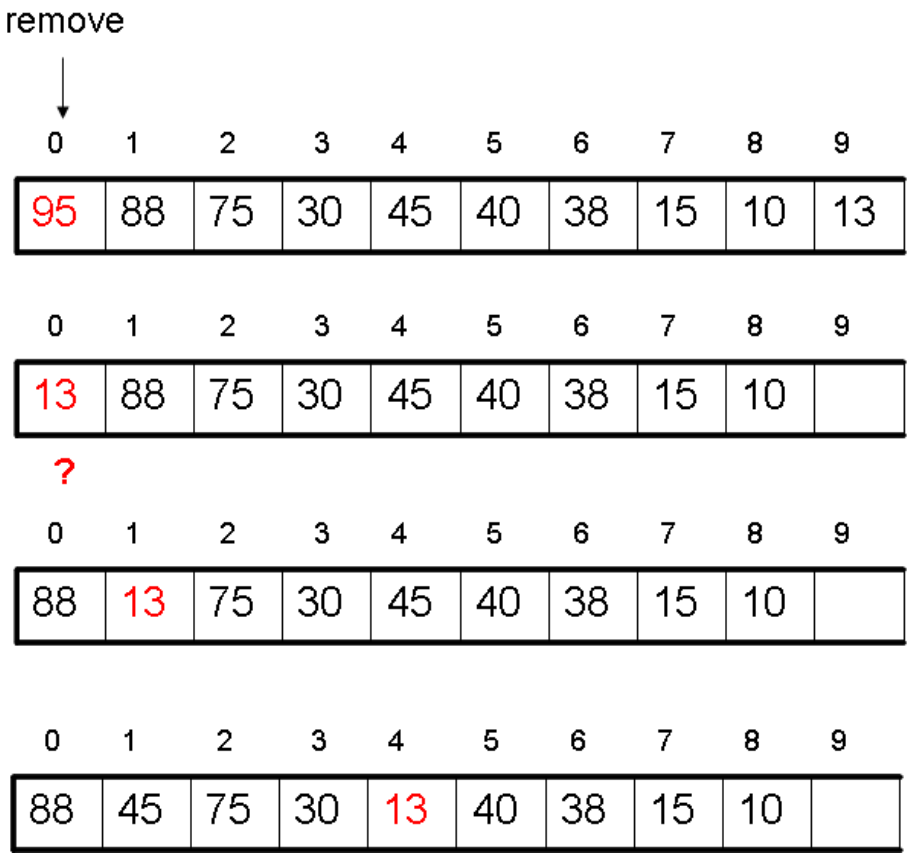
- Inserção em *heaps*: $O(logn)$

```
void InsereHeap(Heap *h, T  x) {
  vetor[(*h).tam] = x ;   /* insere no final do vetor */
  ((*h).tam++ ;
  Sobe(h, ((*h).tam-1) ; /* coloca x na posicao correta */
} /* InsereHeap */
```

- Remoção em *heaps*: $O(logn)$
  - Sempre feita na raiz, retirando-se o elemento de maior prioridade.
  - A lista passa a ter $n-1$ elementos.
  - O último elemento do conjunto é colocado no início da lista para forçar a aparição do novo elemento de maior prioridade.

- Remoção em *heaps*: $O(logn)$

  - Sempre feita na raiz, retirando-se o elemento de maior prioridade.
  - A lista passa a ter $n - 1$ elementos.
  - O último elemento do conjunto é colocado no início da lista para forçar a aparição do novo elemento de maior prioridade.



- Remoção em *heaps* em C: $O(logn)$

```
void RemoveHeap (Heap *h, T  *x) {
   *x = (*h).vetor[0] ;       /* remove a raiz */
   ((*h).tam)-- ;
   (*h).vetor[0] = (*h).vetor[(*h).tam] ;
   Desce(h,0) ;          /* desce chave na posicao 0 */
}  /* RemoveHeap */
```

Filas de Prioridade

- Implementação por *heap*:
  - Seleção: $O(1)$
  - Inserção: $O(logn)$
  - Remoção: $O(logn)$
  - Alteração: $O(logn)$
  - Construção: $O(nlogn)$

Filas de Prioridade

- Implementação por *heap*:
  - Seleção: $O(1)$
  - Inserção: $O(logn)$
  - Remoção: $O(logn)$
  - Alteração: $O(logn)$
  - Construção: $O(nlogn)$

Filas de Prioridade

- Implementação por *heap*:
  - Seleção: $O(1)$
  - Inserção: $O(logn)$
  - Remoção: $O(logn)$
  - Alteração: $O(logn)$
  - Construção: $O(nlogn)$

Filas de Prioridade

- Implementação por *heap*:
  - Seleção: $O(1)$
  - Inserção: $O(logn)$
  - Remoção: $O(logn)$
  - Alteração: $O(logn)$
  - Construção: $O(nlogn)$

Filas de Prioridade

- Implementação por *heap*:
  - Seleção: $O(1)$
  - Inserção: $O(logn)$
  - Remoção: $O(logn)$
  - Alteração: $O(logn)$
  - Construção: $O(nlogn)$

Filas de Prioridade

- Implementação por lista não ordenada:
  - Seleção: $O(n)$
  - Inserção: $O(1)$
  - Remoção: $O(n)$
  - Alteração: $O(n)$
  - Construção: $O(n)$

Filas de Prioridade

- Implementação por lista não ordenada:
  - Seleção: $O(n)$
  - Inserção: $O(1)$
  - Remoção: $O(n)$
  - Alteração: $O(n)$
  - Construção: $O(n)$

Filas de Prioridade

- Implementação por lista não ordenada:
  - Seleção: $O(n)$
  - Inserção: $O(1)$
  - Remoção: $O(n)$
  - Alteração: $O(n)$
  - Construção: $O(n)$

Filas de Prioridade

- Implementação por lista não ordenada:
  - Seleção: $O(n)$
  - Inserção: $O(1)$
  - Remoção: $O(n)$
  - Alteração: $O(n)$
  - Construção: $O(n)$

Filas de Prioridade

- Implementação por lista não ordenada:
  - Seleção: $O(n)$
  - Inserção: $O(1)$
  - Remoção: $O(n)$
  - Alteração: $O(n)$
  - Construção: $O(n)$

Filas de Prioridade

- Implementação por lista não ordenada:
  - Seleção: $O(n)$
  - Inserção: $O(1)$
  - Remoção: $O(n)$
  - Alteração: $O(n)$
  - Construção: $O(n)$

Filas de Prioridade

- Implementação por lista ordenada:
  - Seleção: $O(1)$
  - Inserção: $O(n)$
  - Remoção: $O(1)$
  - Alteração: $O(n)$
  - Construção: $O(nlogn)$

$\Rightarrow$ Estas implementações apresentam operações importantes de complexidade $O(n)$.

Filas de Prioridade

- Implementação por lista ordenada:
  - Seleção: $O(1)$
  - Inserção: $O(n)$
  - Remoção: $O(1)$
  - Alteração: $O(n)$
  - Construção: $O(nlogn)$

$\Rightarrow$ Estas implementações apresentam operações importantes de complexidade $O(n)$.

Filas de Prioridade

- Implementação por lista ordenada:
  - Seleção: $O(1)$
  - Inserção: $O(n)$
  - Remoção: $O(1)$
  - Alteração: $O(n)$
  - Construção: $O(nlogn)$

$\Rightarrow$ Estas implementações apresentam operações importantes de complexidade $O(n)$.

Filas de Prioridade

- Implementação por lista ordenada:
  - Seleção: $O(1)$
  - Inserção: $O(n)$
  - Remoção: $O(1)$
  - Alteração: $O(n)$
  - Construção: $O(nlogn)$

$\Rightarrow$ Estas implementações apresentam operações importantes de complexidade $O(n)$.

Filas de Prioridade

- Implementação por lista ordenada:
  - Seleção: $O(1)$
  - Inserção: $O(n)$
  - Remoção: $O(1)$
  - Alteração: $O(n)$
  - Construção: $O(nlogn)$

$\Rightarrow$ Estas implementações apresentam operações importantes de complexidade $O(n)$.

Filas de Prioridade

- Implementação por lista ordenada:
  - Seleção: $O(1)$
  - Inserção: $O(n)$
  - Remoção: $O(1)$
  - Alteração: $O(n)$
  - Construção: $O(nlogn)$

$\Rightarrow$ Estas implementações apresentam operações importantes de complexidade $O(n)$.

Filas de Prioridade

- Implementação por lista ordenada:
  - Seleção: $O(1)$
  - Inserção: $O(n)$
  - Remoção: $O(1)$
  - Alteração: $O(n)$
  - Construção: $O(nlogn)$

$\Rightarrow$ Estas implementações apresentam operações importantes de complexidade $O(n)$.

Outra aplicação do *heap*

- O *Heapsort*

– Algoritmo:

1. Construir uma fila de prioridade com os elementos a serem ordenados,

2. Aplicar remoções sucessivas dispondo o primeiro elemento (maximal ou minimal) na sua posição correta (a última casa do vetor contendo o heap).

- ```
  /* O Heapsort */
  CrontroiHeap(h)
  m = tam_heap-1
  enquanto m > 0 faça:
    Troque h(0) com h(m)
    m = m - 1
    Desce(0, m)
  ```
  ```

Outra aplicação do *heap*

- O *Heapsort*

– Algoritmo:

1. Construir uma fila de prioridade com os elementos a serem ordenados,

2. Aplicar remoções sucessivas dispondo o primeiro elemento (maximal ou minimal) na sua posição correta (a última casa do vetor contendo o heap).

```
/* O Heapsort */
CrontroiHeap(h)
m = tam_heap-1
enquanto m > 0 faça:
  Troque h(0) com h(m)
  m = m - 1
  Desce(0, m)
```

Outra aplicação do *heap*

- O *Heapsort*

– Algoritmo:

1. Construir uma fila de prioridade com os elementos a serem ordenados,

2. Aplicar remoções sucessivas dispondo o primeiro elemento (maximal ou minimal) na sua posição correta (a última casa do vetor contendo o heap).

```
/* O Heapsort */
CrontroiHeap(h)
m = tam_heap-1
enquanto m > 0 faça:
  Troque h(0) com h(m)
  m = m - 1
  Desce(0, m)
```

Outra aplicação do *heap*

- O *Heapsort*

– Algoritmo:

1. Construir uma fila de prioridade com os elementos a serem ordenados,

2. Aplicar remoções sucessivas dispondo o primeiro elemento (maximal ou minimal) na sua posição correta (a última casa do vetor contendo o heap).

- ```
  /* O Heapsort */
  CrontroiHeap(h)
  m = tam_heap-1
  enquanto m > 0 faça:
    Troque h(0) com h(m)
    m = m - 1
    Desce(0, m)
  ```
  ```

- O *Heapsort* em C: $O(nlogn)$

```
void HeapSort(Heap *h) {
  int i, n = (*h).tam;
  for (i=(n-2)/2; i>=0; i--) /* constrói heap */
    Desce(h,i);
  for (i=n-1; i>0; i--) {    /* ordena */
    T t = (*h).vetor[0];
    (*h).vetor[0] = (*h).vetor[i];
    (*h).vetor[i] = t;
    (*h).tam--;
    Desce(h,0);
  }
  (*h).tam = n;
} /* HeapSort */
```

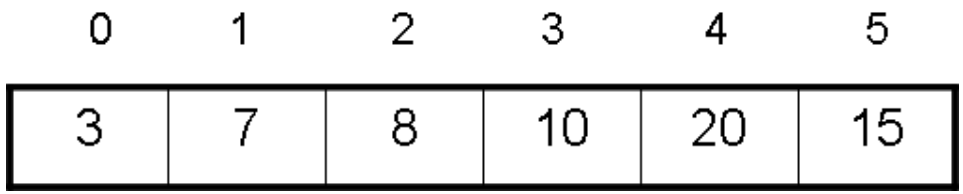
- Exemplo:

Dados a serem ordenados:

0	1	2	3	4	5
10	8	7	3	20	15

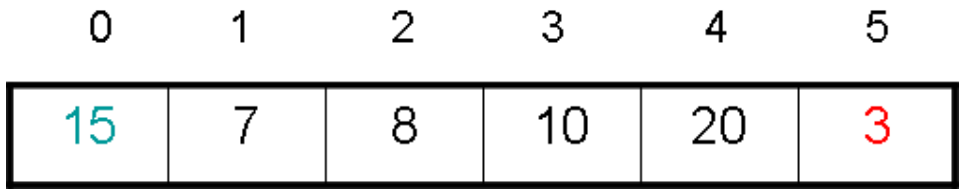
- Exemplo:

Heap:

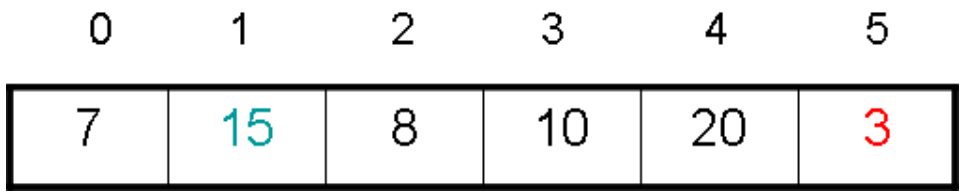


- Exemplo:

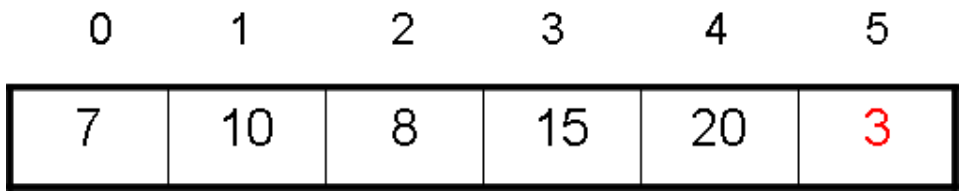
Troca 3 com 15 e Desce até 4:



- Exemplo:

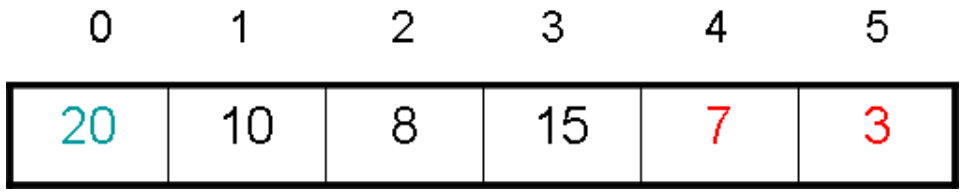


- Exemplo:

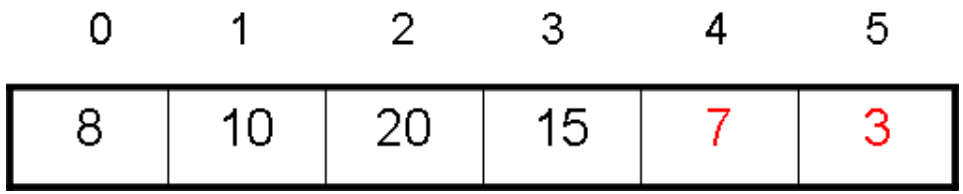


- Exemplo:

Troca 7 com 20 e Desce até 3:

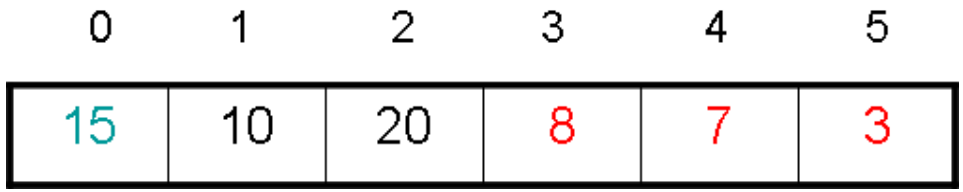


- Exemplo:

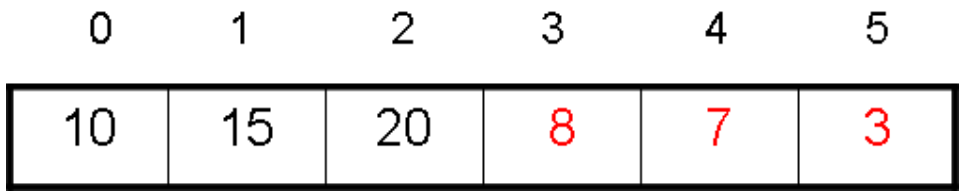


- Exemplo:

Troca 8 com 15 e Desce até 2:

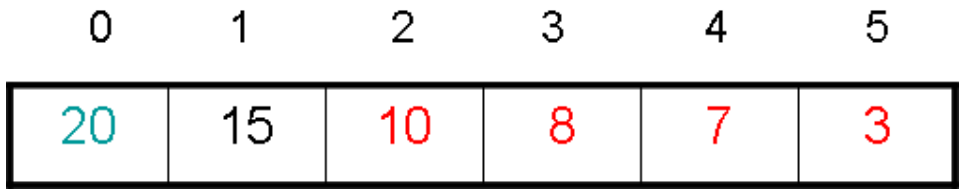


- Exemplo:

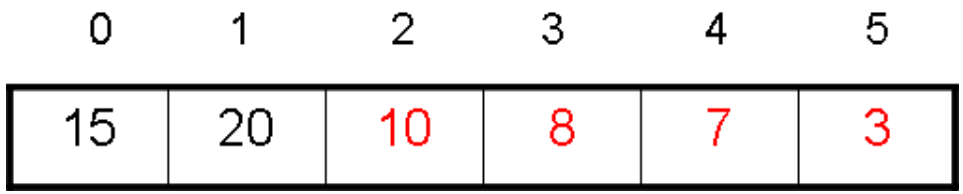


- Exemplo:

Troca 10 com 20 e Desce até 1:

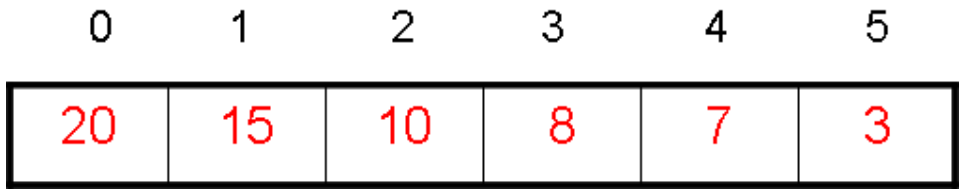


- Exemplo:



- Exemplo:

Troca 15 com 20 e desce até 0 (FIM!!!):



Referências:

– Jayme L. Szwarcfiter e Lilian Markenzon. Estruturas de Dados e seus Algoritmos.Editora LTC, 1994.

– Nívio Ziviani. Projeto de Algoritmos com Implementações em Pascal e C, Thomson Learning, 2004.

– Apostila dos professores Thomasz e Lucchesi.