MC-202 — Aula 16
Fila de Prioridade e *Heap-Sort*

Lehilton Pedrosa

Instituto de Computação – Unicamp

Segundo Semestre de 2015

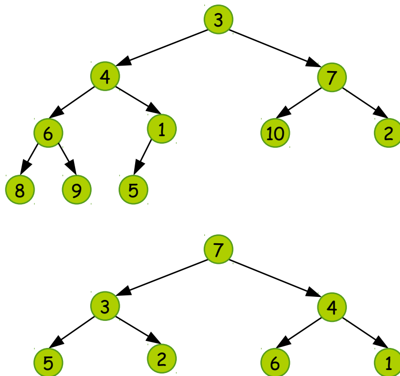
Roteiro

1. Introdução

2. Fila de Prioridade

Introdução

Pergunta: Que propriedade têm essas árvores?



Árvore-B

Árvore quase-completa

Uma árvore quase-completa é uma árvore binária em que:

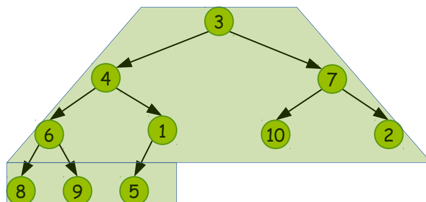
1. com exceção do último, todos níveis têm o máximo de elementos
2. todas as folhas do último nível estão “à esquerda”

Árvore-B

Árvore quase-completa

Uma árvore quase-completa é uma árvore binária em que:

1. com exceção do último, todos níveis têm o máximo de elementos
2. todas as folhas do último nível estão "à esquerda"

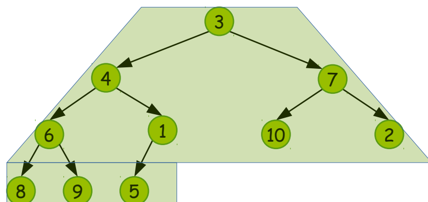


Árvore-B

Árvore quase-completa

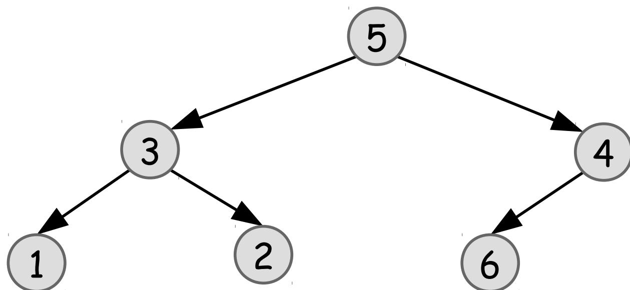
Uma árvore quase-completa é uma árvore binária em que:

1. com exceção do último, todos níveis têm o máximo de elementos
2. todas as folhas do último nível estão "à esquerda"

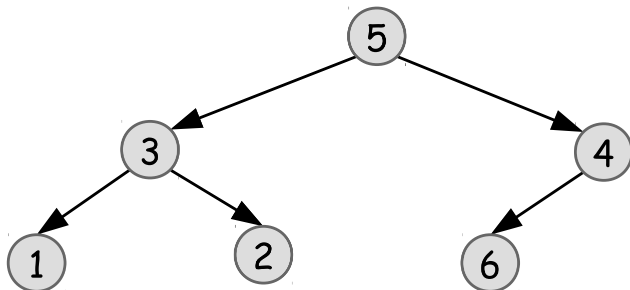


Uma árvore completa é uma árvore quase-completa em que todos níveis têm o máximo de elementos.

Percorrendo um árvore quase-completa

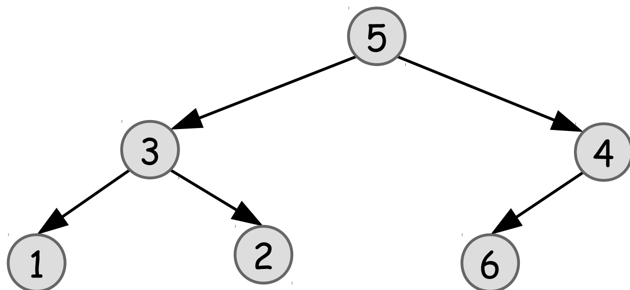


Percorrendo um árvore quase-completa

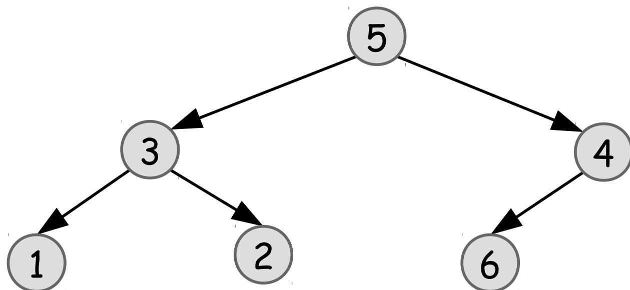


5

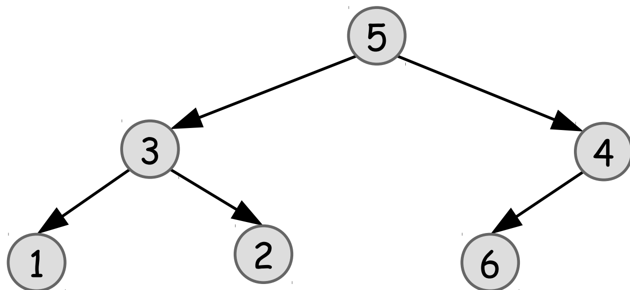
Percorrendo um árvore quase-completa



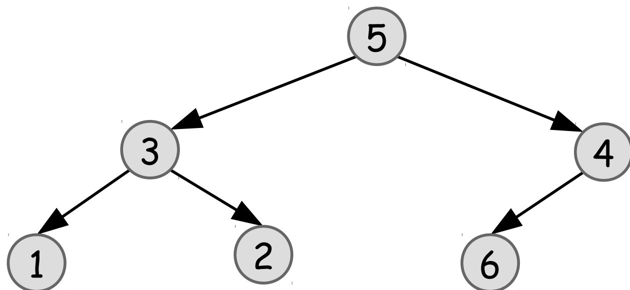
Percorrendo um árvore quase-completa



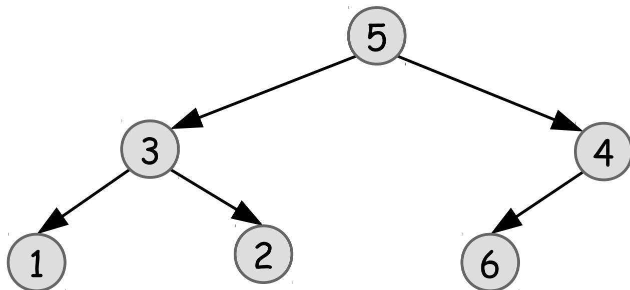
Percorrendo um árvore quase-completa



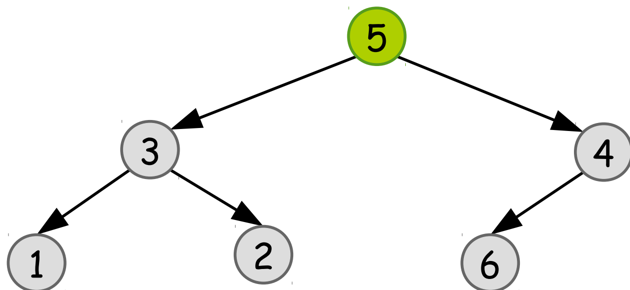
Percorrendo um árvore quase-completa



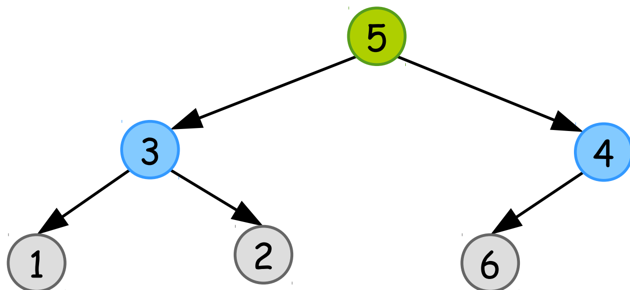
Percorrendo um árvore quase-completa



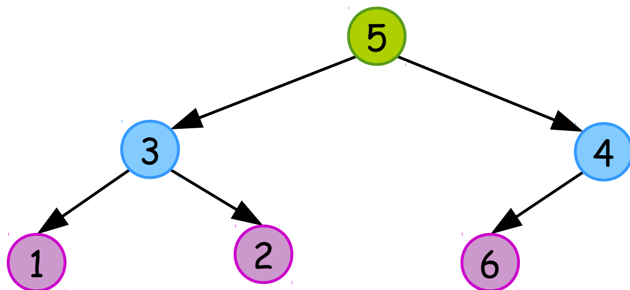
Percorrendo um árvore quase-completa



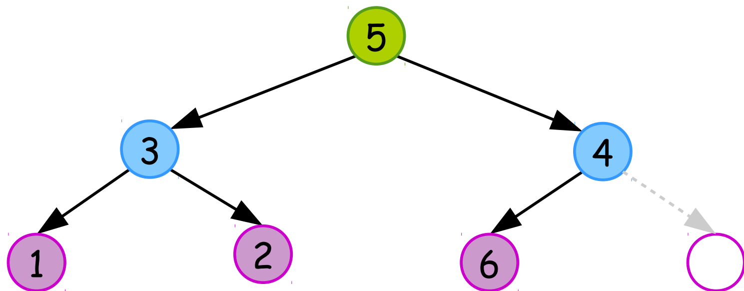
Percorrendo um árvore quase-completa

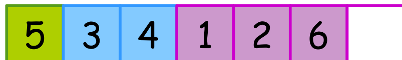


Percorrendo um árvore quase-completa

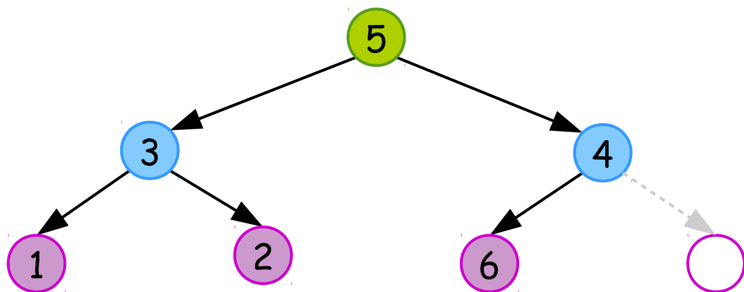


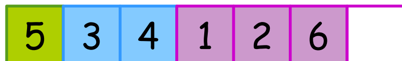
Percorrendo um árvore quase-completa





Percorrendo um árvore quase-completa





Podemos utilizar um vetor para armazenar!

Árvore quase-completa em vetor

```
typdef struct {
    int *v;
    int tam;
} Arv;
```

Árvore quase-completa em vetor

```
typdef struct {
    int *v;
    int tam;
} Arv;

// retorna o índice do filho esquerdo
int esq(int i) {
    return 2*i + 1;
}
```

Árvore quase-completa em vetor

```
typdef struct {
    int *v;
    int tam;
} Arv;

// retorna o índice do filho esquerdo
int esq(int i) {
    return 2*i + 1;
}
// retorna o índice do filho direito
int dir(int i) {
    return 2*i + 2;
}
```

Árvore quase-completa em vetor

```
typdef struct {
    int *v;
    int tam;
} Arv;

// retorna o índice do filho esquerdo
int esq(int i) {
    return 2*i + 1;
}
// retorna o índice do filho direito
int dir(int i) {
    return 2*i + 2;
}
// retorna o índice do pai
int pai(int i) {
    return (i - 1)/2;
}
```

Árvore quase-completa em vetor

```
typdef struct {
    int *v;
    int tam;
} Arv;

// retorna o índice do filho esquerdo
int esq(int i) {
    return 2*i + 1;
}
// retorna o índice do filho direito
int dir(int i) {
    return 2*i + 2;
}
// retorna o índice do pai
int pai(int i) {
    return (i - 1)/2;
}
```

Como verifico se há filho?

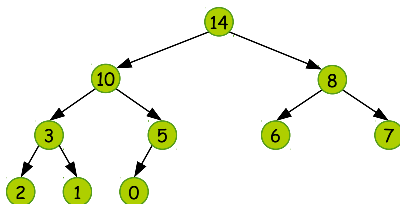
Árvore quase-completa em vetor

```
typdef struct {
    int *v;
    int tam;
} Arv;

// retorna o índice do filho esquerdo
int esq(int i) {
    return 2*i + 1;
}
// retorna o índice do filho direito
int dir(int i) {
    return 2*i + 2;
}
// retorna o índice do pai
int pai(int i) {
    return (i - 1)/2;
}
```

Como verifico se há filho? `esq(i) < arvore->tam`

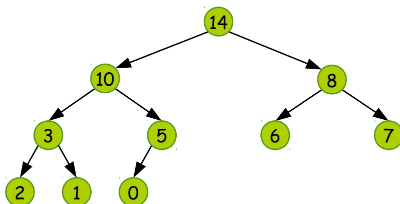
Fila de prioridade

Pergunta: Que propriedade tem essa árvore?



Fila de prioridade

Pergunta: Que propriedade tem essa árvore?



Fila de prioridade (*Max-Heap*)

Uma árvore binária é uma fila de prioridade se

1. ela for quase-completa
2. a chave de todo nó não é menor que a de seus descendentes

Fila de Prioridade como Conjunto Dinâmico

Tipo abstrato de dados

```
typedef struct {
    int *v;
    int tam;
} Heap;
```

Fila de Prioridade como Conjunto Dinâmico

Tipo abstrato de dados

```
typedef struct {
    int *v;
    int tam;
} Heap;

void criar_fila(Heap *fila, int *dados, int tam);
```

Fila de Prioridade como Conjunto Dinâmico

Tipo abstrato de dados

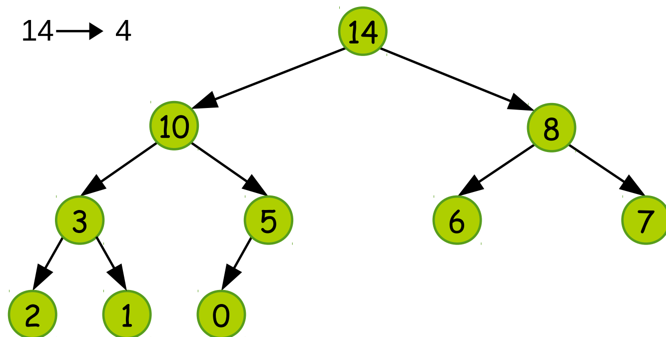
```
typedef struct {
    int *v;
    int tam;
} Heap;

void criar_fila(Heap *fila, int *dados, int tam);

void inserir(Heap *fila, int x);
int remover_max(Heap *fila);
```

Fila de Prioridade como Conjunto Dinâmico

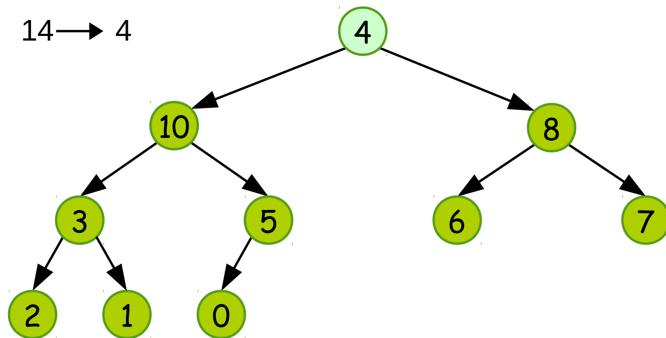
Tipo abstrato de dados

```
typedef struct {
    int *v;
    int tam;
} Heap;

void criar_fila(Heap *fila, int *dados, int tam);

void inserir(Heap *fila, int x);
int remover_max(Heap *fila);

// x é a nova prioridade do elemento na posição i
void aumenta_prioridade(Heap *fila, int i, int x);
void diminui_prioridade(Heap *fila, int i, int x);
```

Diminuindo a prioridade

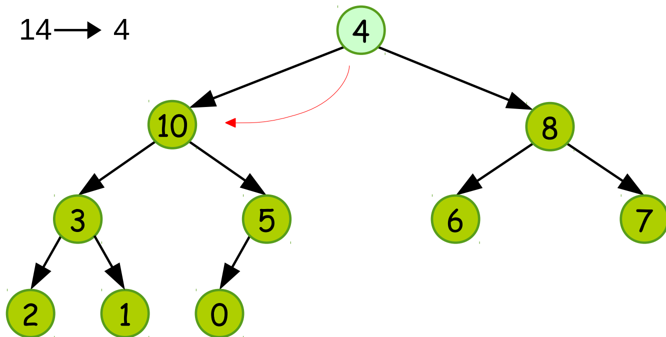


Diminuindo a prioridade



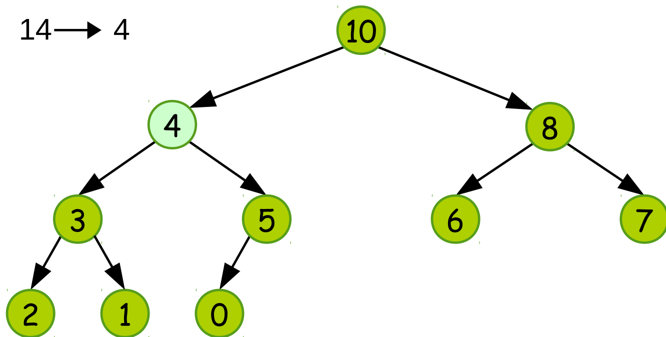
1. mudamos a prioridade

Diminuindo a prioridade



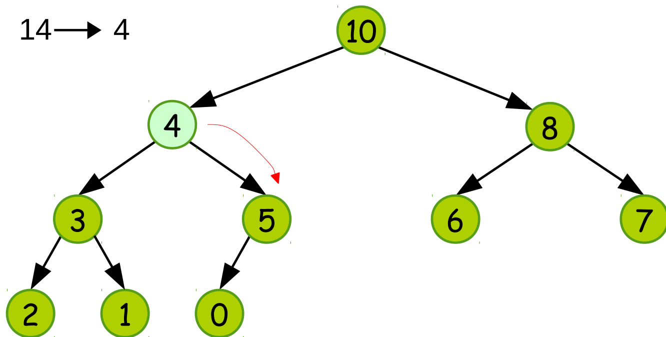
1. mudamos a prioridade
2. descemos na árvore

Diminuindo a prioridade



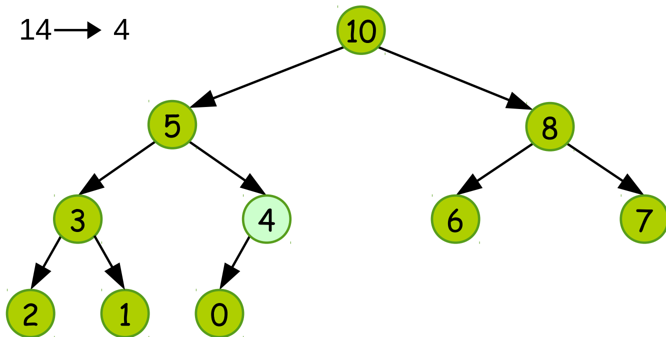
1. mudamos a prioridade
2. descemos na árvore

Diminuindo a prioridade



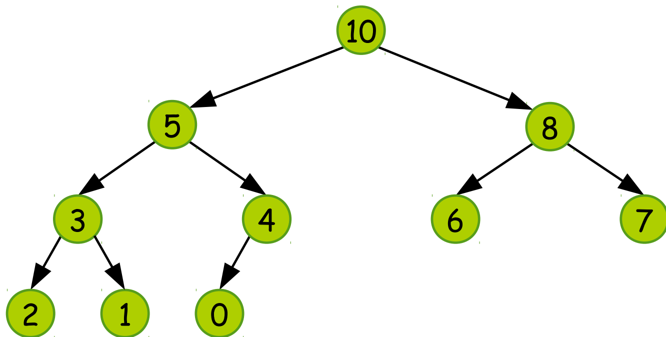
1. mudamos a prioridade
2. descemos na árvore

Diminuindo a prioridade



1. mudamos a prioridade
2. descemos na árvore

Diminuindo a prioridade



1. mudamos a prioridade
2. descemos na árvore

Diminuir a prioridade em C

```
void diminui_prioridade(Heap *fila, int i, int x) {
    fila->v[i] = x;
    desce(fila,i);
}
```

Diminuir a prioridade em C

```
void diminui_prioridade(Heap *fila, int i, int x) {
    fila->v[i] = x;
    desce(fila,i);
}
void desce(Heap *fila, int i) {
    // se não for folha
    int maior = i;
    if (esq(i) < fila->tam && fila->v[esq(i)] > fila->v[maior])
        maior = esq(i);
    if (dir(i) < fila->tam && fila->v[dir(i)] > fila->v[maior])
        maior = dir(i);
```

Diminuir a prioridade em C

```
void diminui_prioridade(Heap *fila, int i, int x) {
    fila->v[i] = x;
    desce(fila,i);
}
void desce(Heap *fila, int i) {
    // se não for folha
    int maior = i;
    if (esq(i) < fila->tam && fila->v[esq(i)] > fila->v[maior])
        maior = esq(i);
    if (dir(i) < fila->tam && fila->v[dir(i)] > fila->v[maior])
        maior = dir(i);
    if (i != maior) {
        troca(&fila->v[i], &fila->v[maior]);
        desce(fila, maior);
    }
}
```

Diminuir a prioridade em C

```
void diminui_prioridade(Heap *fila, int i, int x) {
    fila->v[i] = x;
    desce(fila,i);
}
void desce(Heap *fila, int i) {
    // se não for folha
    int maior = i;
    if (esq(i) < fila->tam && fila->v[esq(i)] > fila->v[maior])
        maior = esq(i);
    if (dir(i) < fila->tam && fila->v[dir(i)] > fila->v[maior])
        maior = dir(i);
    if (i != maior) {
        troca(&fila->v[i], &fila->v[maior]);
        desce(fila, maior);
    }
}
```

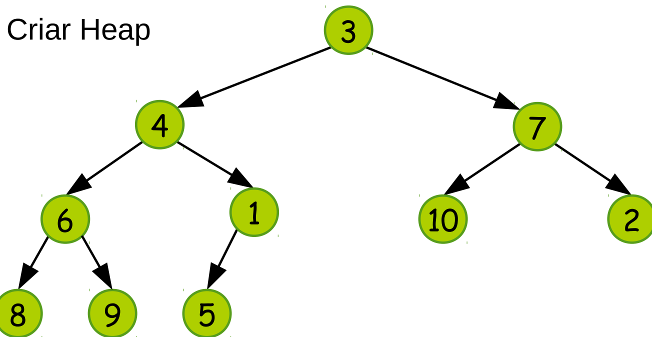
Qual complexidade?

Diminuir a prioridade em C

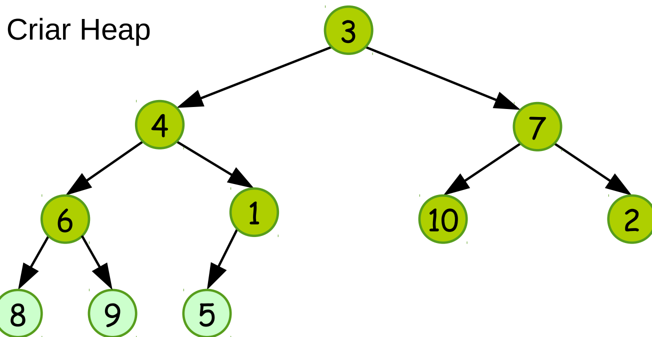
```
void diminui_prioridade(Heap *fila, int i, int x) {
    fila->v[i] = x;
    desce(fila,i);
}
void desce(Heap *fila, int i) {
    // se não for folha
    int maior = i;
    if (esq(i) < fila->tam && fila->v[esq(i)] > fila->v[maior])
        maior = esq(i);
    if (dir(i) < fila->tam && fila->v[dir(i)] > fila->v[maior])
        maior = dir(i);
    if (i != maior) {
        troca(&fila->v[i], &fila->v[maior]);
        desce(fila, maior);
    }
}
```

Qual complexidade? $O(\log n)$

Criar Fila

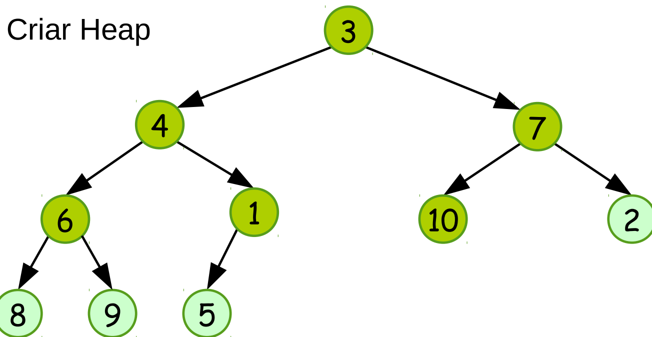


Criar Fila



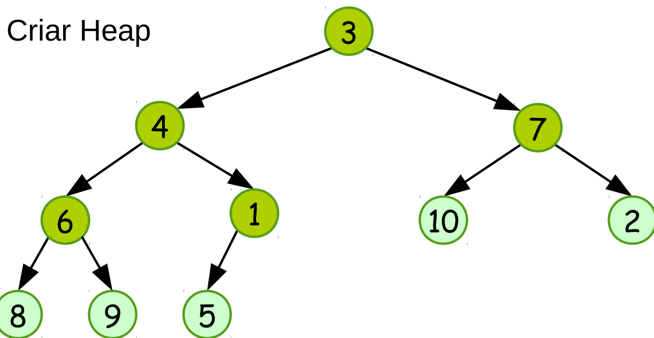
1. as folhas já são filas!

Criar Fila



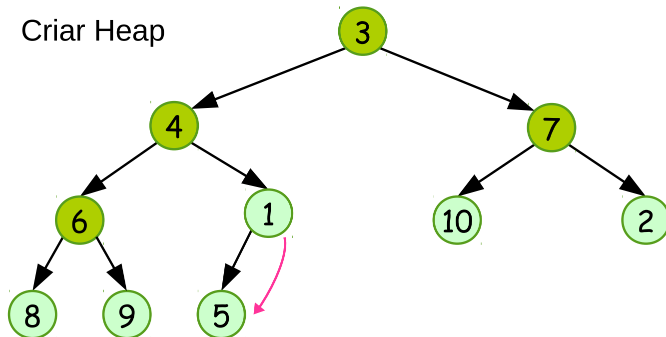
1. as folhas já são filas!

Criar Fila



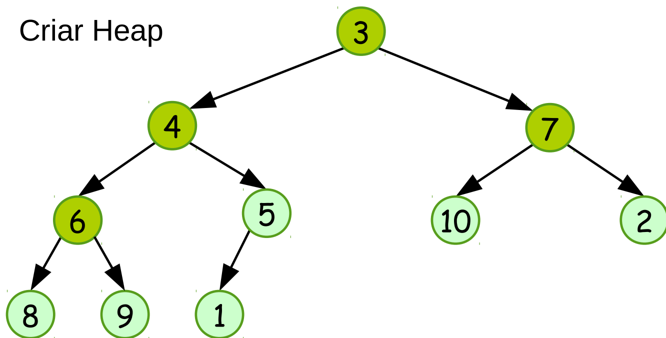
1. as folhas já são filas!

Criar Fila



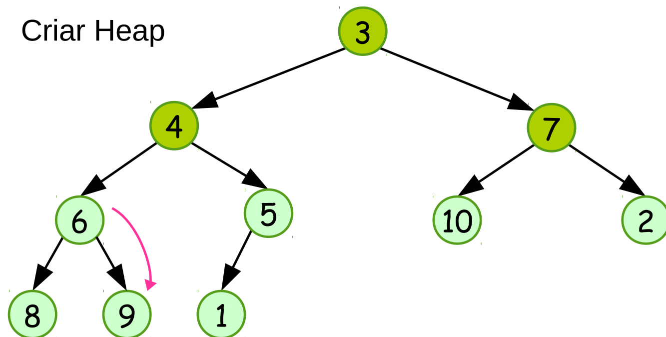
1. as folhas já são filas!
2. inserimos outros elementos na fila (em ordem inversa)

Criar Fila



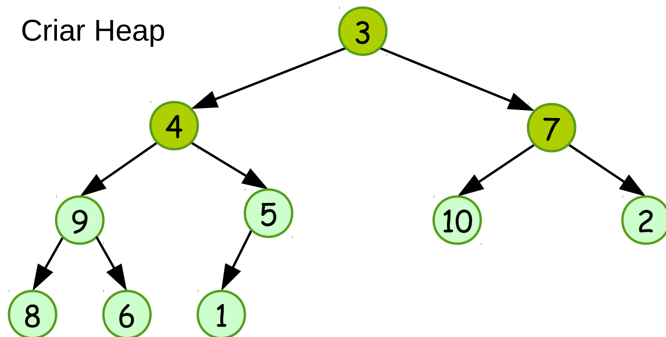
1. as folhas já são filas!
2. inserimos outros elementos na fila (em ordem inversa)

Criar Fila



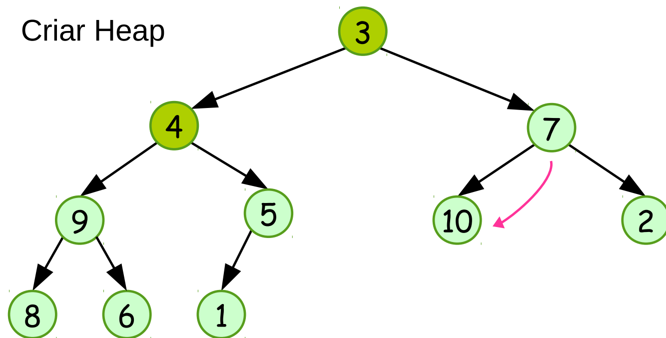
1. as folhas já são filas!
2. inserimos outros elementos na fila (em ordem inversa)

Criar Fila



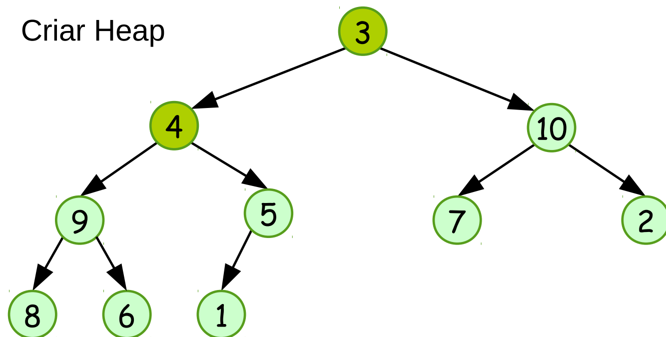
1. as folhas já são filas!
2. inserimos outros elementos na fila (em ordem inversa)

Criar Fila



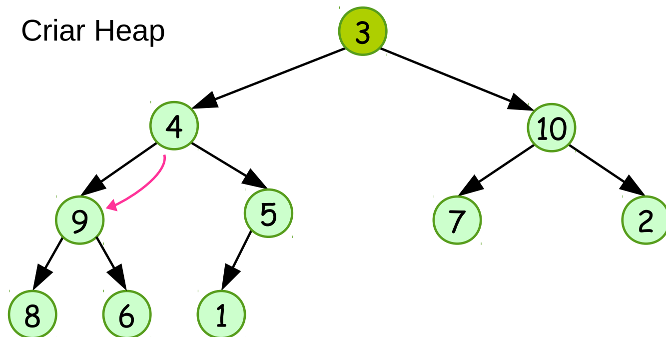
1. as folhas já são filas!
2. inserimos outros elementos na fila (em ordem inversa)

Criar Fila



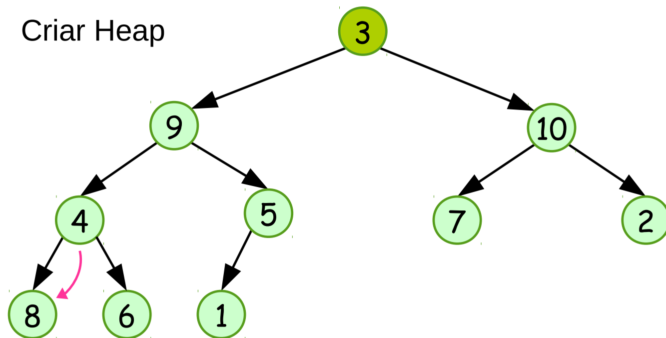
1. as folhas já são filas!
2. inserimos outros elementos na fila (em ordem inversa)

Criar Fila



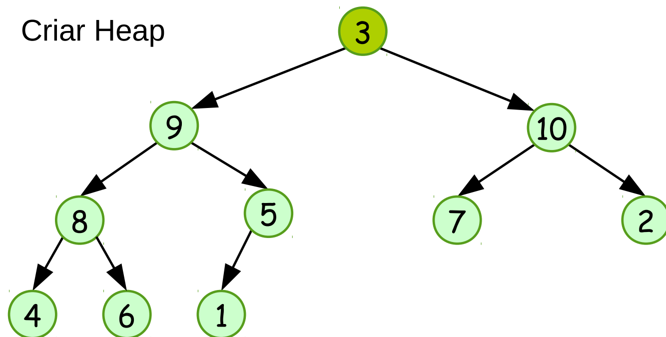
1. as folhas já são filas!
2. inserimos outros elementos na fila (em ordem inversa)

Criar Fila



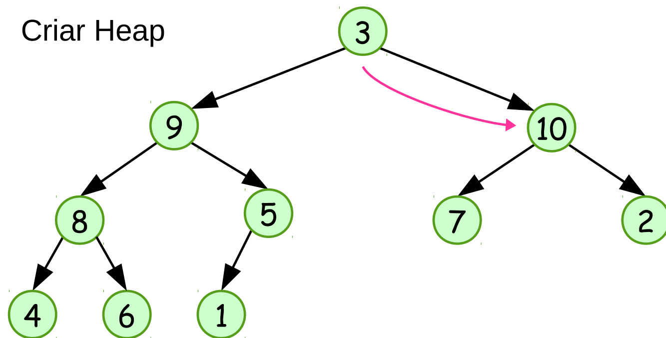
1. as folhas já são filas!
2. inserimos outros elementos na fila (em ordem inversa)

Criar Fila



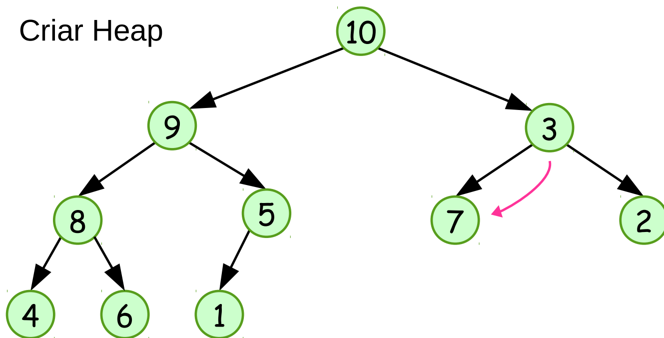
1. as folhas já são filas!
2. inserimos outros elementos na fila (em ordem inversa)

Criar Fila



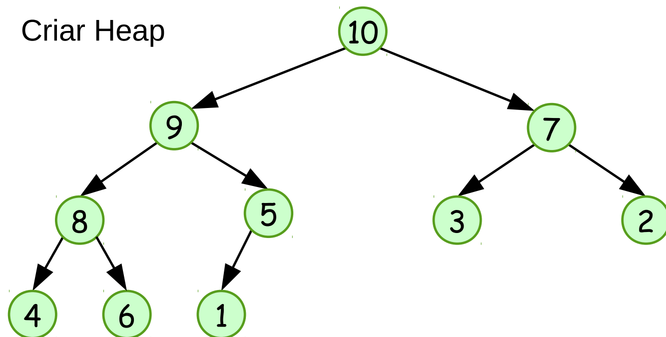
1. as folhas já são filas!
2. inserimos outros elementos na fila (em ordem inversa)

Criar Fila



1. as folhas já são filas!
2. inserimos outros elementos na fila (em ordem inversa)

Criar Fila



1. as folhas já são filas!
2. inserimos outros elementos na fila (em ordem inversa)

Criar fila em C

```
void criar_fila(Heap *fila, int *dados, int tam) {
    int i;

    fila->v = dados;
    fila->tam = tam;
```

Criar fila em C

```
void criar_fila(Heap *fila, int *dados, int tam) {
    int i;

    fila->v = dados;
    fila->tam = tam;

    // insere no heap a partir do primeiro pai (em ordem inversa)
    for (i = pai(v->tam - 1); i >= 0; i--)
        desce(fila, i);
}
```

Criar fila em C

```
void criar_fila(Heap *fila, int *dados, int tam) {
    int i;

    fila->v = dados;
    fila->tam = tam;

    // insere no heap a partir do primeiro pai (em ordem inversa)
    for (i = pai(v->tam - 1); i >= 0; i--)
        desce(fila, i);
}
```

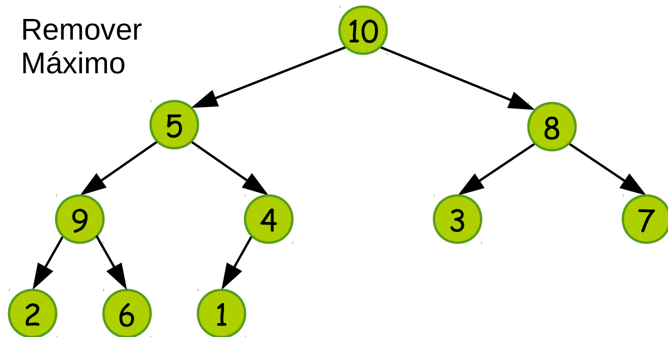
Qual complexidade?

Criar fila em C

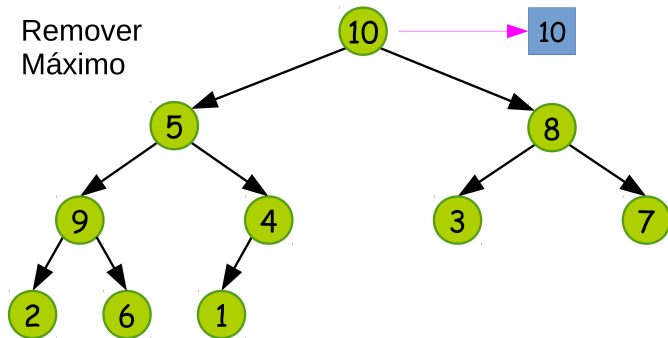
```
void criar_fila(Heap *fila, int *dados, int tam) {
    int i;

    fila->v = dados;
    fila->tam = tam;

    // insere no heap a partir do primeiro pai (em ordem inversa)
    for (i = pai(v->tam - 1); i >= 0; i--)
        desce(fila, i);
}
```

Qual complexidade?
Pode-se mostrar que esse algoritmo executa em $O(n)$.

Remover máximo

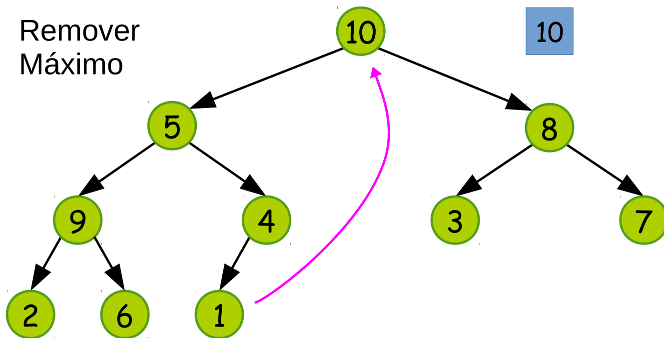


Remover máximo



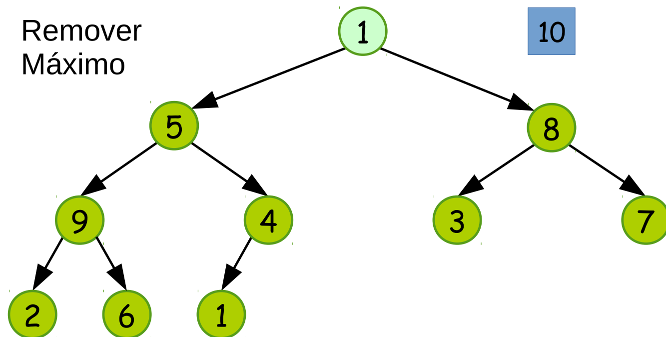
1. guardamos a raiz com o máximo

Remover máximo



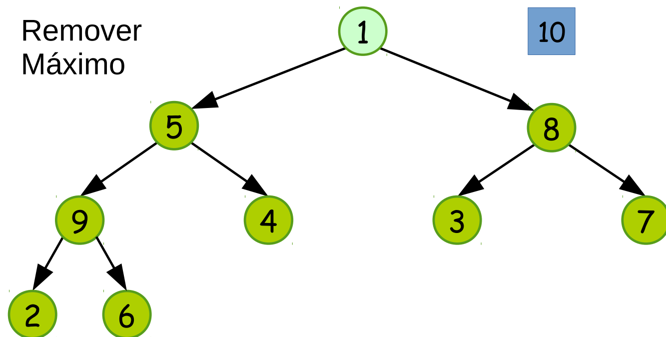
1. guardamos a raiz com o máximo
2. removemos o último elemento diminuímos a prioridade da raiz

Remover máximo



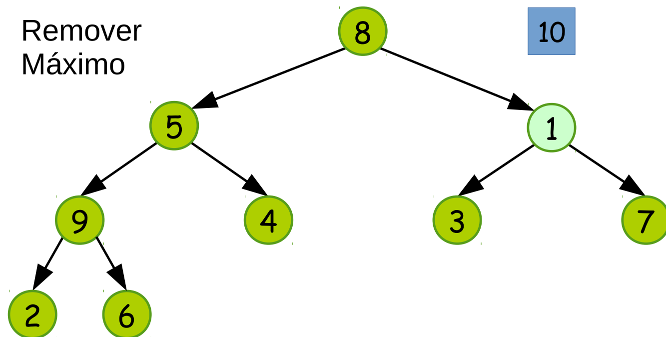
1. guardamos a raiz com o máximo
2. removemos o último elemento diminuímos a prioridade da raiz

Remover máximo



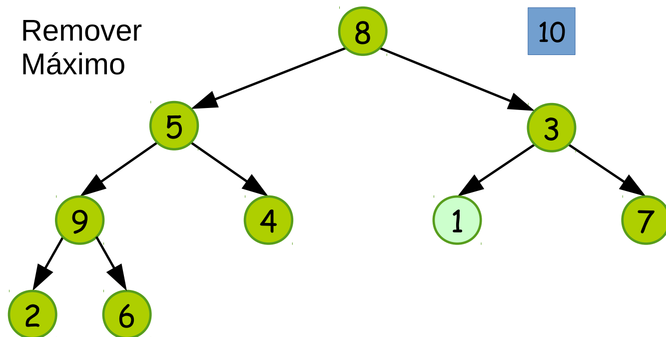
1. guardamos a raiz com o máximo
2. removemos o último elemento diminuímos a prioridade da raiz

Remover máximo



1. guardamos a raiz com o máximo
2. removemos o último elemento diminuímos a prioridade da raiz

Remover máximo



1. guardamos a raiz com o máximo
2. removemos o último elemento diminuímos a prioridade da raiz

Remover máximo em C

```
int remover_max(Heap *fila) {
    int max = fila->v[0];
    fila->v[0] = fila->v[fila->tam-1];
    (fila->tam)--;
    desce(fila, 0);
    return max;
}
```

Remover máximo em C

```
int remover_max(Heap *fila) {
    int max = fila->v[0];
    fila->v[0] = fila->v[fila->tam-1];
    (fila->tam)--;
    desce(fila, 0);
    return max;
}
```

Qual complexidade?

Remover máximo em C

```
int remover_max(Heap *fila) {
    int max = fila->v[0];
    fila->v[0] = fila->v[fila->tam-1];
    (fila->tam)--;
    desce(fila, 0);
    return max;
}
```

Qual complexidade? $O(\log n)$

Heap-Sort

Ordenando elementos

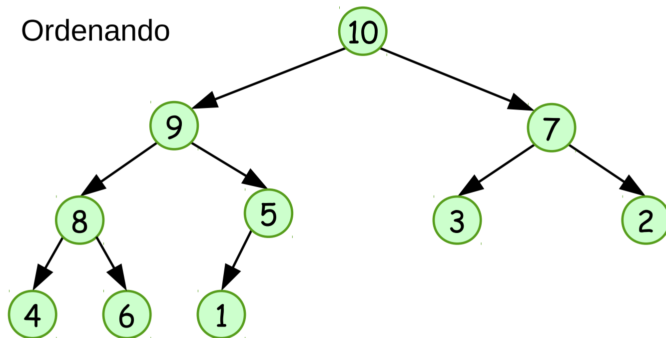
Pergunta: como podemos obter todos os elementos de uma fila de prioridade ordenados?

Heap-Sort

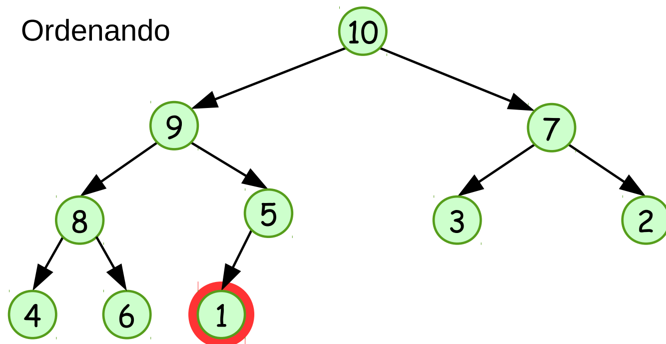
Ordenando elementos

Pergunta: como podemos obter todos os elementos de uma fila de prioridade ordenados?
Resposta: basta ir retirando os máximos

Ordenando

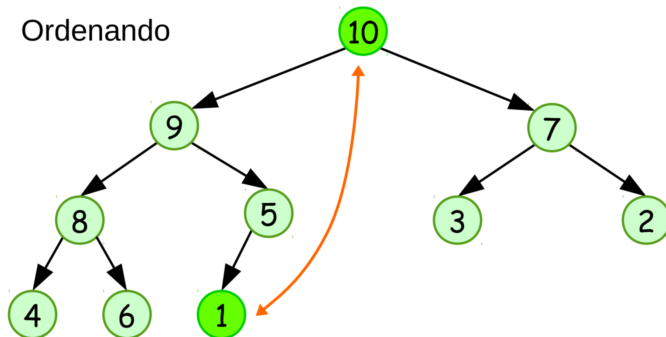


Ordenando



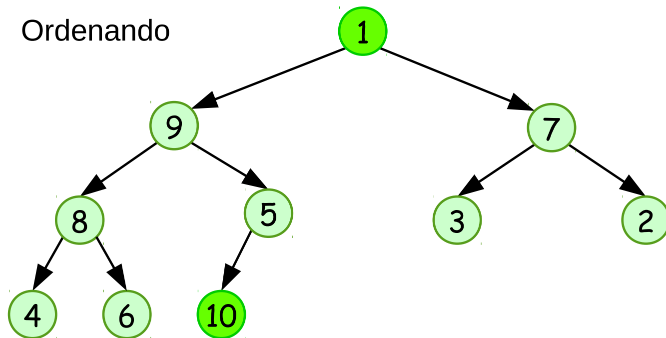
Para cada elemento:

Ordenando



Para cada elemento:

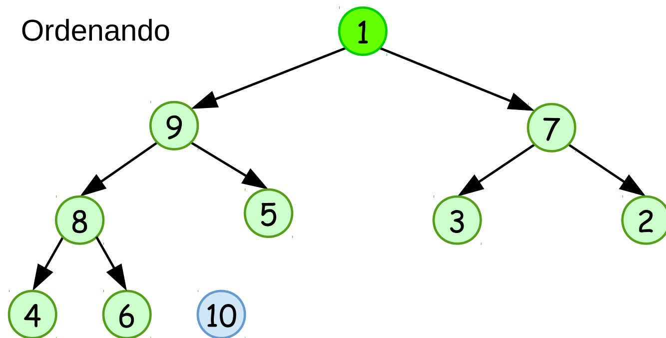
Ordenando



Para cada elemento:

1. trocamos último com a raiz

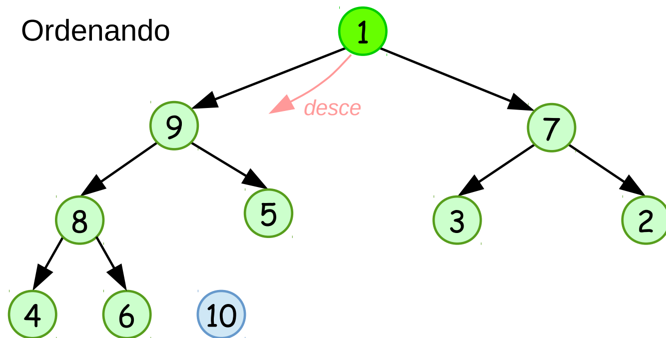
Ordenando



Para cada elemento:

1. trocamos último com a raiz

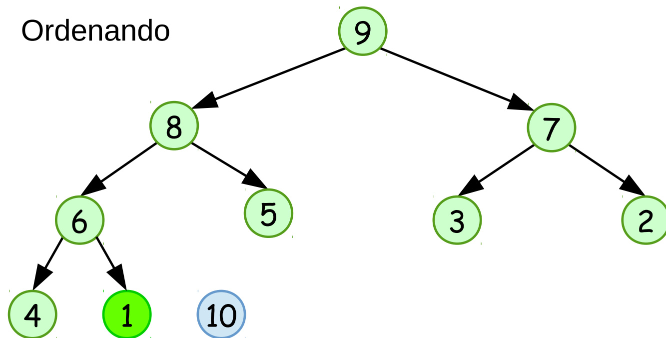
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

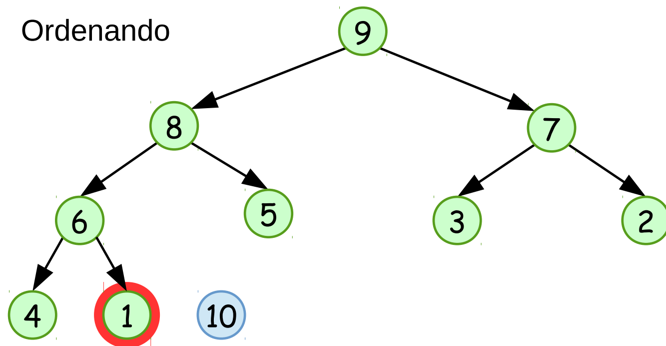
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

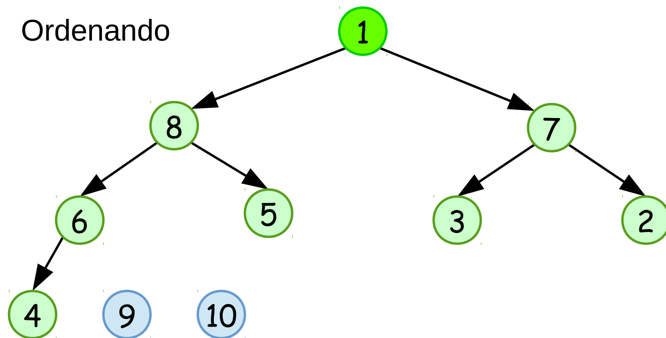
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

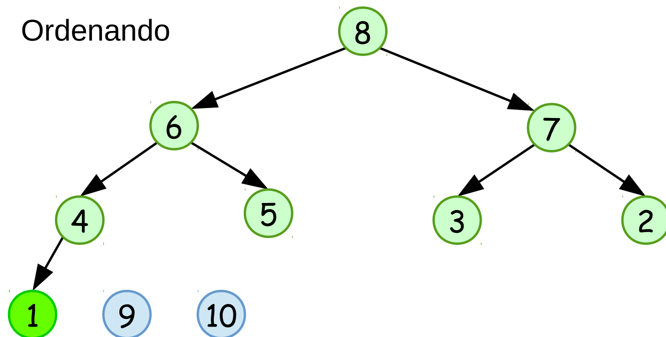
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

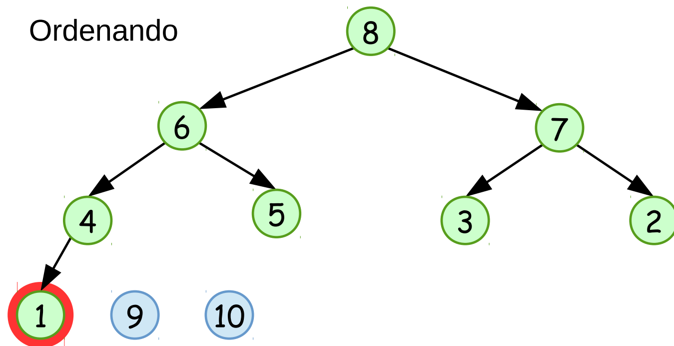
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

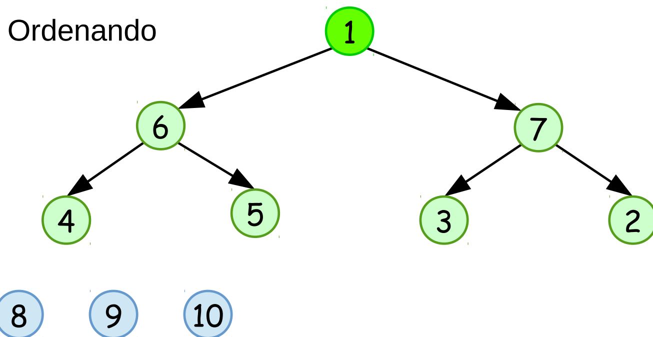
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

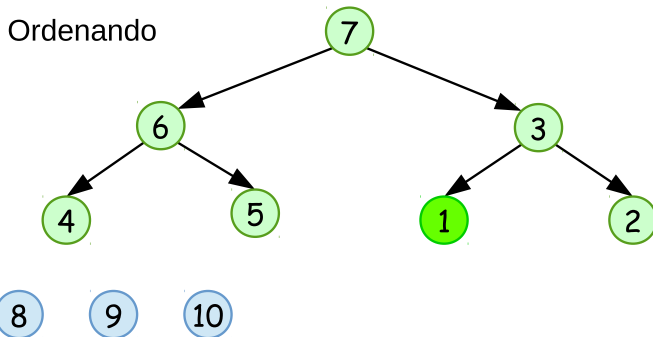
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

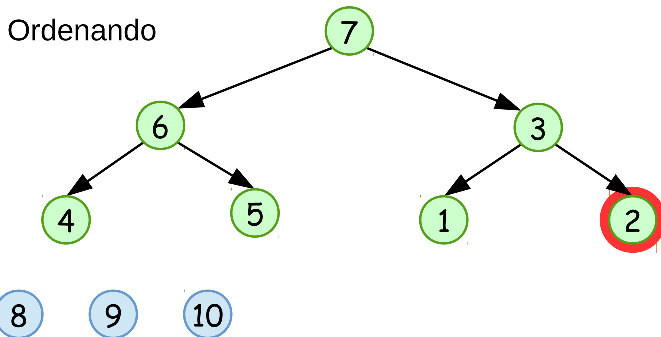
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

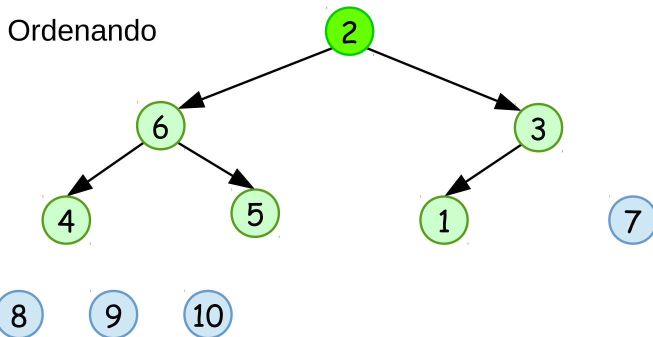
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

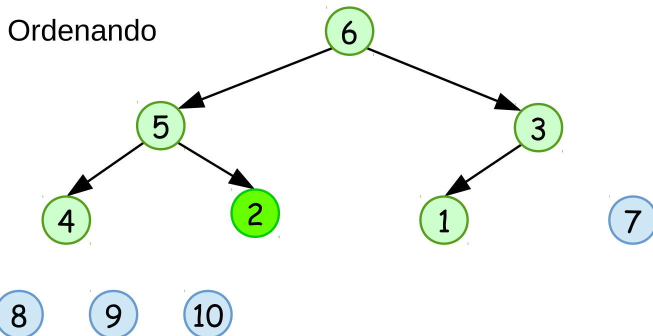
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

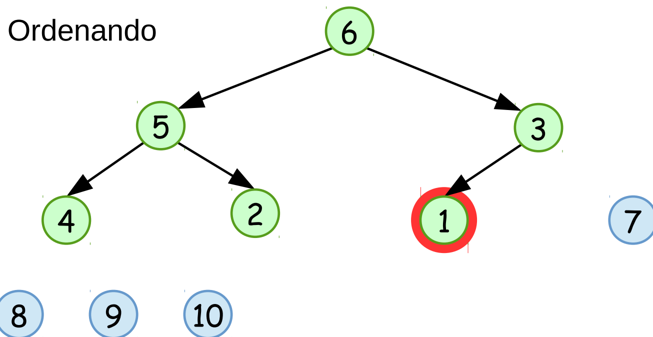
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

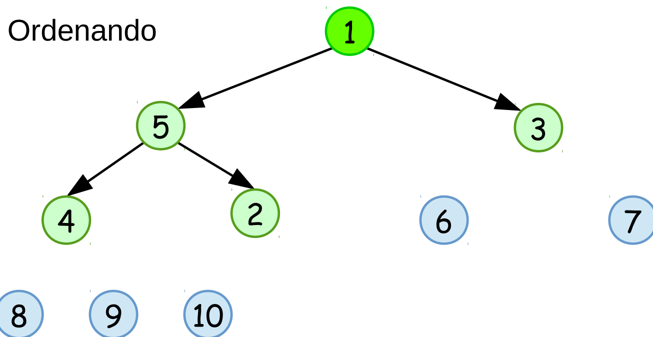
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

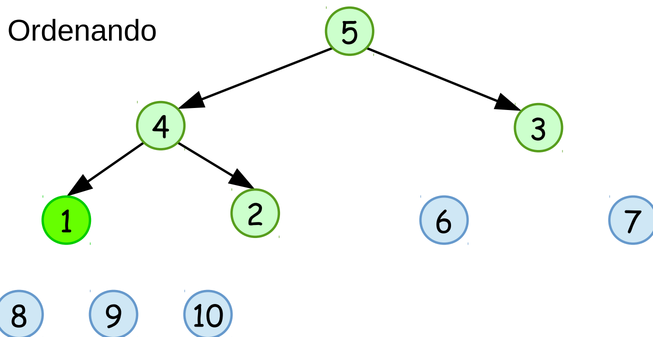
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

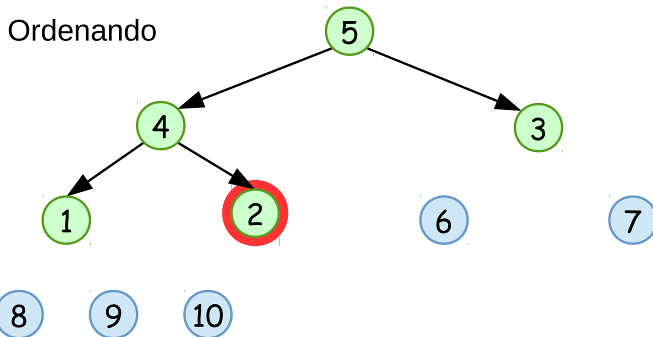
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

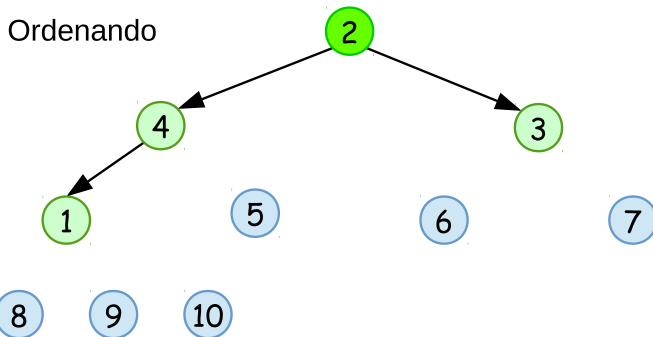
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

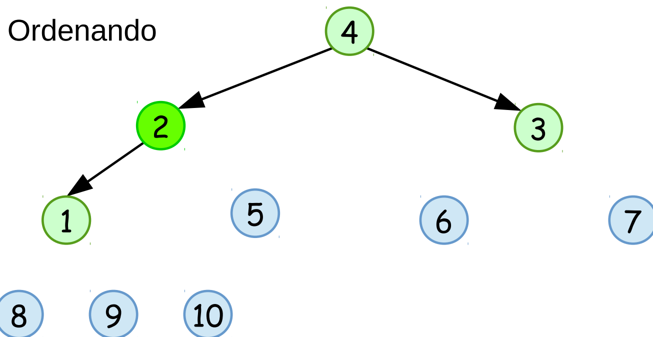
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

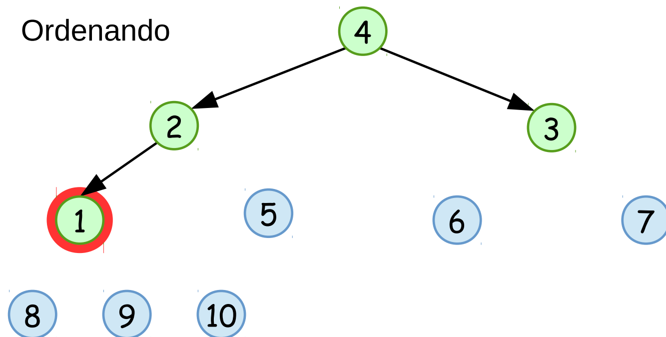
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

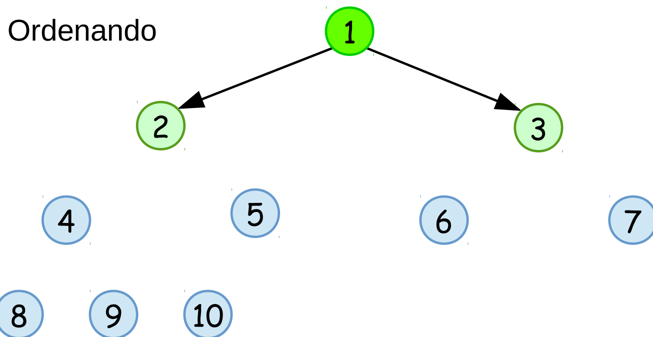
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

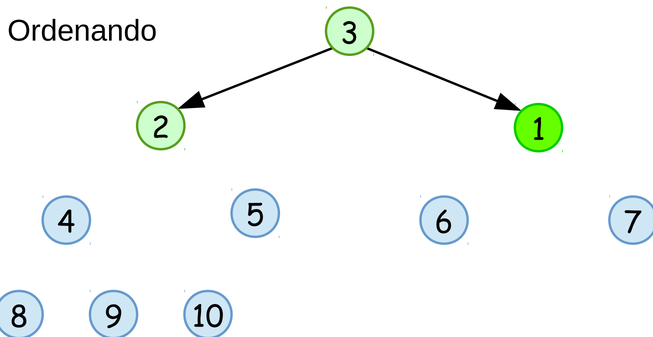
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

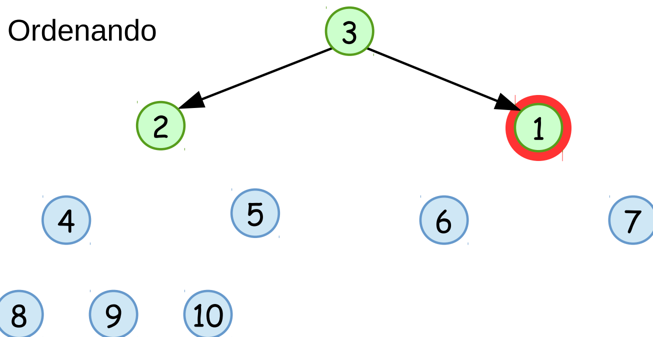
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

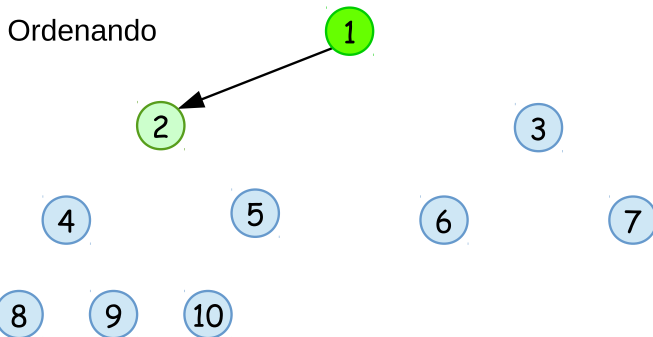
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

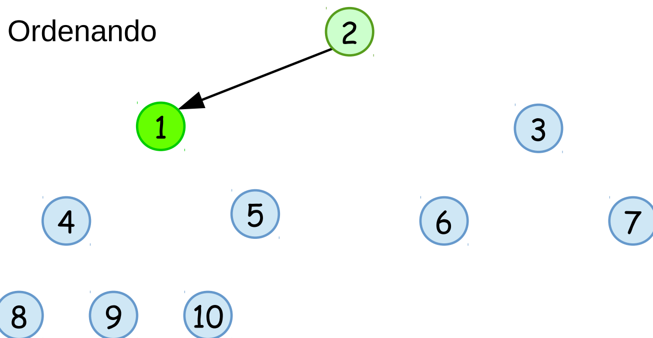
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

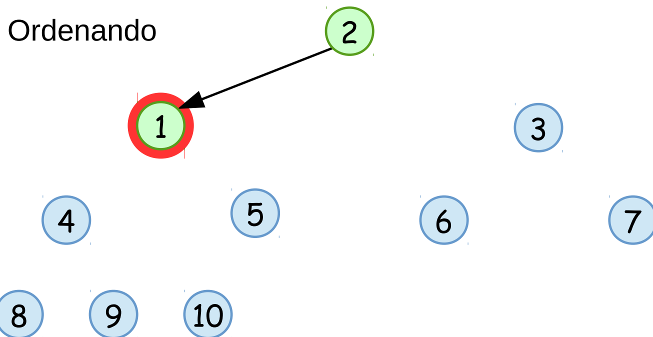
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

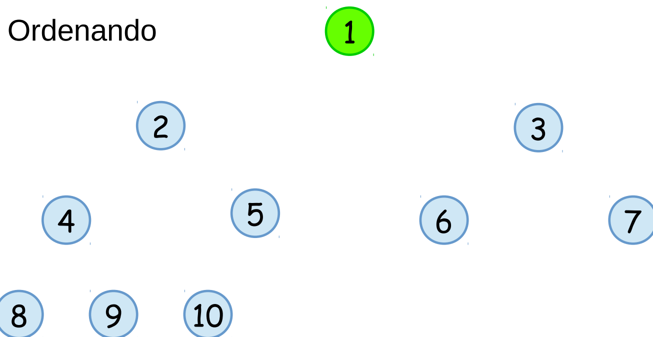
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

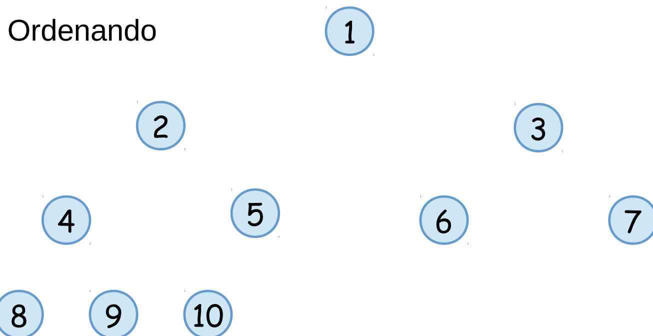
Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

Ordenando



Para cada elemento:

1. trocamos último com a raiz e descemos

Implementação do Heap-Sort

```
void heap_sort(int v[], int n) {
    Heap fila;
    int i;
```

Implementação do Heap-Sort

```
void heap_sort(int v[], int n) {
    Heap fila;
    int i;

    criar_fila(&fila, v, n);
```

Implementação do Heap-Sort

```
void heap_sort(int v[], int n) {
    Heap fila;
    int i;

    criar_fila(&fila, v, n);

    for (i = n-1; i > 0; i--)
        v[i] = remove_max(fila);
}
```

Implementação do Heap-Sort

```
void heap_sort(int v[], int n) {
    Heap fila;
    int i;

    criar_fila(&fila, v, n);

    for (i = n-1; i > 0; i--)
        v[i] = remove_max(fila);
}
```

Qual complexidade?

Implementação do Heap-Sort

```
void heap_sort(int v[], int n) {
    Heap fila;
    int i;

    criar_fila(&fila, v, n);

    for (i = n-1; i > 0; i--)
        v[i] = remove_max(fila);
}
```

Qual complexidade? $O(n \log n)$

Exercício

1. Implemente as operações `inserir` e `aumentar_prioridade`.
2. (desafio) Implemente uma estrutura de dados que contenha as operações "inserir", "remover máximo" e "remover mínimo". O tempo de criação deve ser $O(n)$ e o tempo de cada operação deve ser $O(\log(n))$.

Exercício 2 - Ordenação estável

Um algoritmo de ordenação é considerado estável se dois elementos com a mesma chave são mantidos na mesma ordem em que o vetor original. Por exemplo, vamos ordenar por mês as datas:
5/7/2015, 2/7/2015, 7/9/2010, 6/7/1999
Temos mais de uma ordenação possível:
estável: 5/7/2015, 2/7/2015, 6/7/1999,7/9/2010
não estável: 2/7/2015, 5/7/2015, 6/7/1999, 7/9/2010

1. Justifique porque a ordenação por seleção é estável.
2. Quais algoritmos de ordenação que vimos são estáveis e quais não são. Tente dar um exemplo para cada exemplo que não é estável.