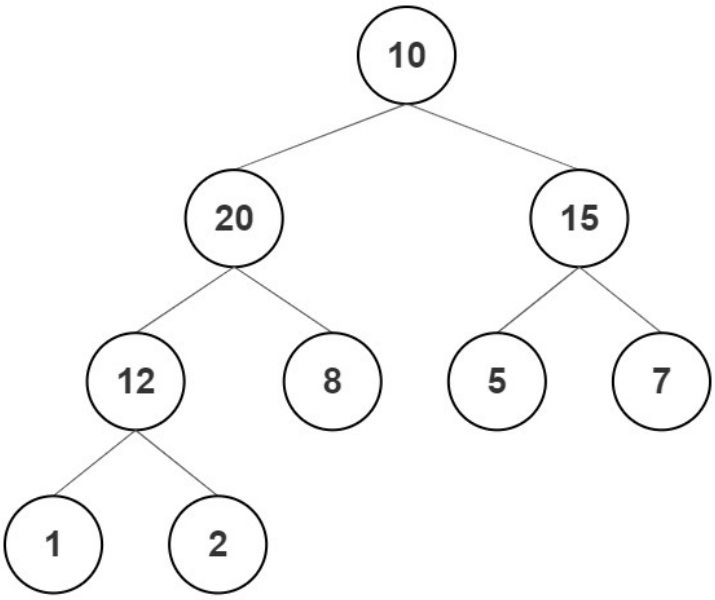
Pós-Graduação em Computação – UFF
Estruturas de Dados e Algoritmos – Profº Ígor Machado Coelho
Lista 3 – Árvores
Aluno: Ítalo Leite Ferreira Portinho

➔ Questão 1:

Árvore completa: 10, 20, 15, 12, 8, 5, 7, 1, 2



a) Percurso em pré-ordem (nó, esq, direita):

10, 20, 12, 1, 2, 8, 15, 5, 7

b) Percurso em pós-ordem(esq, dir, nó):

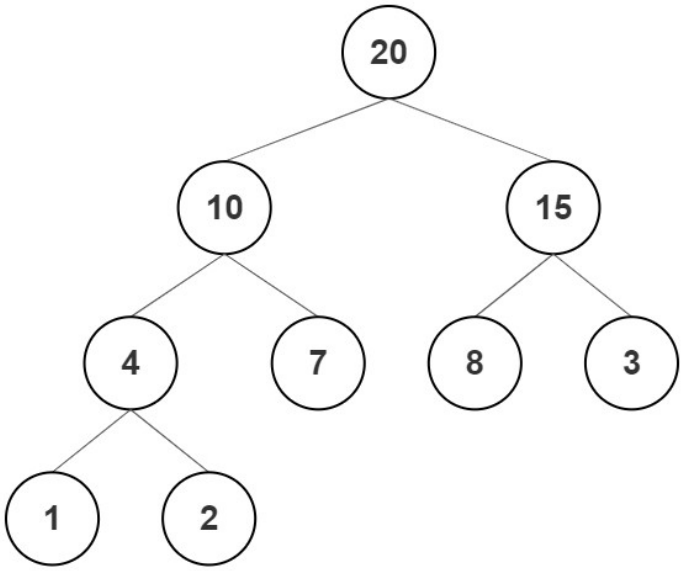
1, 2, 12, 8, 20, 5, 7, 15, 10

c) Percurso em-ordem(esq, nó, dir):

1, 12, 2, 20, 8, 10, 5, 15, 7

➔ Questão 2:

MAX-heap = 20, 10, 15, 4, 7, 8, 3, 1, 2

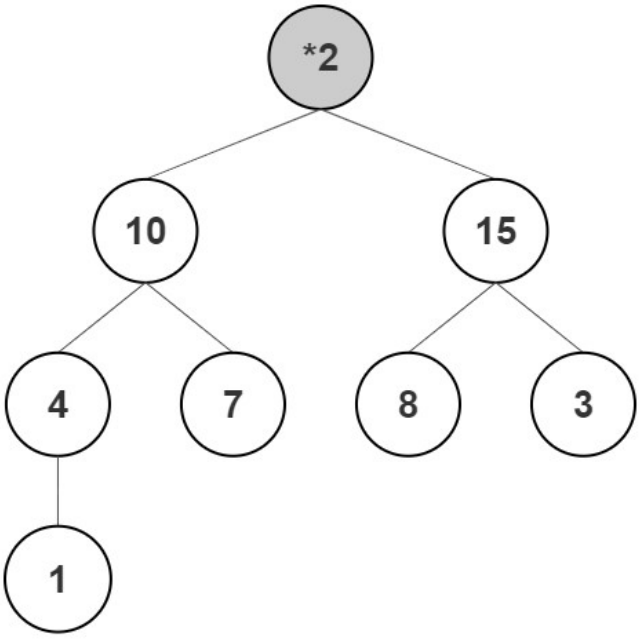


a) Remover elemento de maior prioridade (20):

Passo 1: Colocar o último elemento (2) no lugar do 20, e realizar trocas sucessivas com o maior filho até atender a condição do heap (pai > filhos).

Vetor = *2, 10, 15, 4, 7, 8, 3, 1

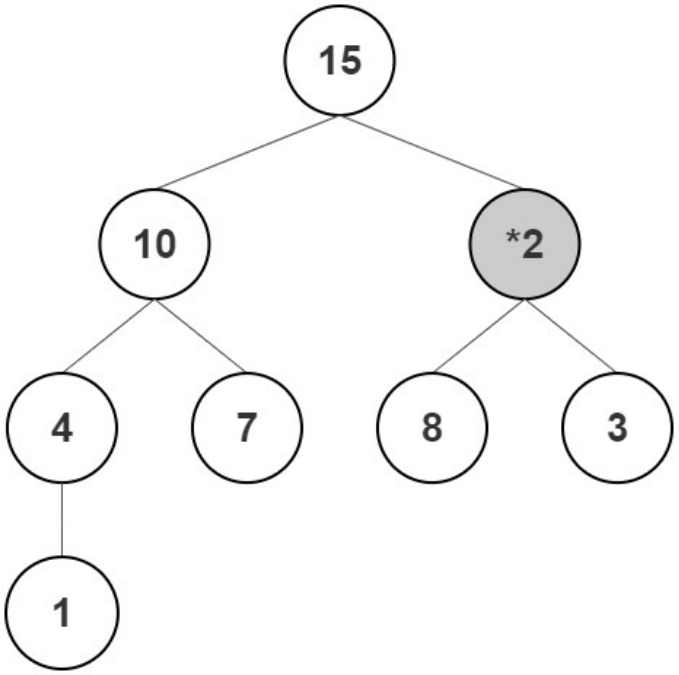
Árvore:



Passo 2: trocar o elemento 2 com seu maior filho (15):

Vetor: 15, 10, *2, 4, 7, 8, 3, 1

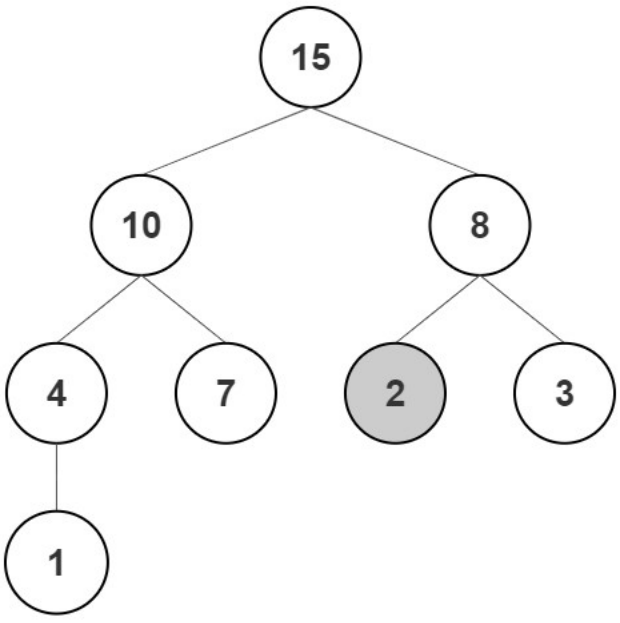
Árvore:



Passo 3: trocar o elemento 2 com seu maior filho (8). É o passo final, pois 2 é menor que pai e não possui filhos para testar.

Vetor: 15, 10, 8, 4, 7, 2, 3, 1

Árvore:

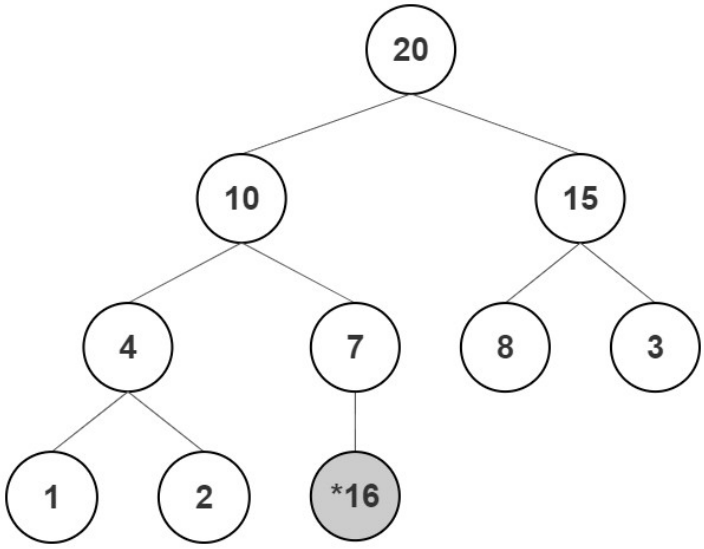


b) Inserir elemento 16 no heap inicial:

Passo 1: Inserir 16 na última posição.

Vetor: 20, 10, 15, 4, 7, 8, 3, 1, 2, *16

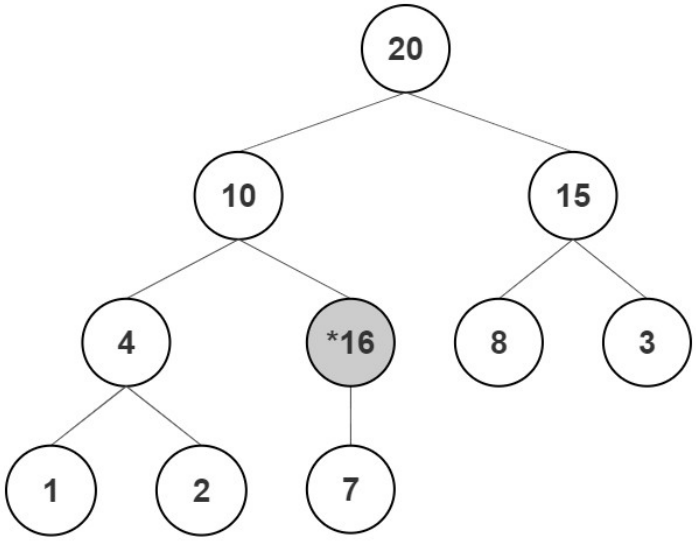
Árvore:



Passo 2: 16 é menor que seu pai (7), então troca de posição.

Vetor: 20, 10, 15, 4, *16, 8, 3, 1, 2, 7

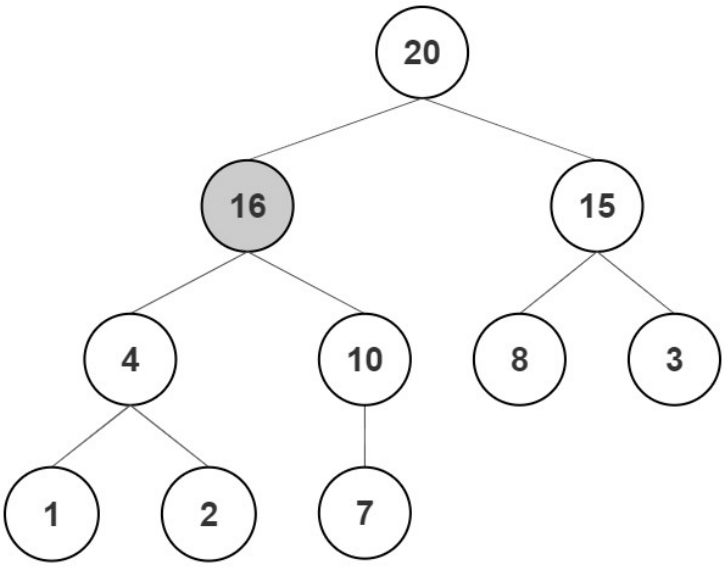
Árvore:



Passo 3: 16 é menor que seu pai (10), então troca de posição. É o passo final, pois agora 16 é menor que seu novo pai (20).

Vetor: 20, 16, 15, 4, 10, 8, 3, 1, 2, 7

Árvore:



➔ Questão 3:

a) Escreva um algoritmo para computar a soma das folhas:

```
int soma_folhas(NoArvore* no, int soma) {
    if (no->esq == 0 && no->dir == 0) {
        soma = soma + no->dado;
    } else {
        soma = soma_folhas(no->esq, soma);
        soma = soma_folhas(no->dir, soma);
    }
    return soma;
}
```

b) Escreva um algoritmo para efetuar um percurso de pós-ordem:

```
void pos_ordem(NoArvore* no) {
    if (no->esq) {
        pos_ordem(no->esq);
    }
    if (no->dir) {
        pos_ordem(no->dir);
    }
    printf("%d ", no->dado);
    return;
}
```

c) Escreva um algoritmo para efetuar um percurso em-ordem:

```
void em_ordem(NoArvore* no) {
    if (no->esq) {
        em_ordem(no->esq);
    }
    printf("%d ", no->dado);
    if (no->dir) {
        em_ordem(no->dir);
    }
    return;
}
```

d) Escreva um algoritmo para efetuar um percurso em pré-ordem:

```
void pre_ordem(NoArvore* no) {
    printf("%d ", no->dado);
    if (no->esq) {
        pre_ordem(no->esq);
    }
    if (no->dir) {
        pre_ordem(no->dir);
    }
    return;
}
```

e) Escreva um algoritmo para computar a altura de um dado nó:

```
int calculaAltura(NoArvore* no, int altura) {
    if (!no) {
        return 0;
    }
    int altura_esq = calculaAltura(no->esq, altura);
    int altura_dir = calculaAltura(no->dir, altura);
    int max_altura = std::max(altura_esq, altura_dir) + 1;

    return max_altura;
}
```

f) Escreva um algoritmo para computar o balanceamento de um nó (O balanceamento de um NÓ é definido como a altura de sua subárvore esquerda menos a altura de sua subárvore direita - Fonte: http://wiki.icmc.usp.br/images/f/f0/AVL.pdf):

```
int balanceamento_no(NoArvore* no) {
    int altura_sub_esquerda = calculaAltura(no->esq, 0);
    int altura_sub_direita = calculaAltura(no->dir, 0);
    return altura_sub_esquerda - altura_sub_direita;
}
```

g) Escreva um algoritmo para percorrer a árvore em níveis:

```
void print_nivel(NoArvore* no, int nivel) {
    if (!no)
        return;
    if (nivel == 1) {
        printf(" %d ", no->dado);
    } else if (nivel > 1) {
        print_nivel(no->esq, nivel - 1);
        print_nivel(no->dir, nivel - 1);
    }
    return;
}
// Função auxiliar
void percorre_niveis(NoArvore* no) {
    int altura = calculaAltura(no, 0);
    for (int i = 1; i <= altura; i++) {
        print_nivel(no, i);
    }
}
```

h) Escreva um algoritmo para computar o produto dos nós:

```
int produto_nos(NoArvore* no, int produto) {
    produto = produto * no->dado;
    if (no->esq) {
        produto = produto_nos(no->esq, produto);
    }
    if (no->dir) {
        produto = produto_nos(no->dir, produto);
    }
    return produto;
}
```