2ª Avaliação Parcial
Curso: Engenharia de Computação
Disciplina: Estruturas de Dados
Prof. Jarbas Joaci de Mesquita Sá Junior
Universidade Federal do Ceará – UFC/Sobral

Nome _____ Data 07/2016.1

1ª) Explique os procedimentos necessários para inserir um elemento numa árvore binária de busca. Insira a seguinte sequência de chaves em uma árvore binária de busca: 9, 22, 13, 55, 44, 12, 56, 11, 17. (2,0 ponto).

2ª) Explique como funcionam os algoritmos *bubble-sort* e *quick-sort*. Quais são suas complexidades no pior e no melhor caso? (2,0 pontos)

3ª) Qual o objetivo das árvores AVL? Explique os procedimentos necessários para manter uma árvore AVL balanceada. Insira a seguinte sequência de chaves em uma árvore AVL: 9, 22, 13, 55, 44, 12, 56, 11, 17. (2,0 pontos).

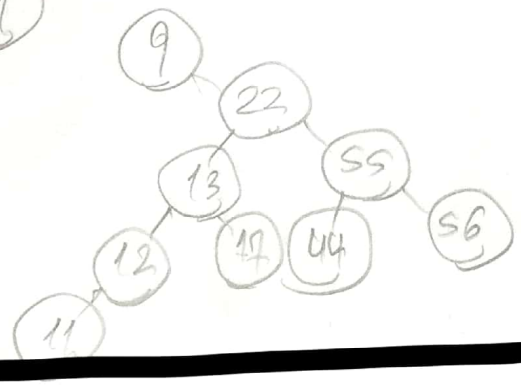
4ª) Construa uma árvore rubro-negra para a seguinte sequência de chaves: 1, 2, 3, 4, 5, 6, 7, 8. (2,0 pontos) remover 8, 4, 6

5ª) Construa uma árvore B de ordem $f = 3$ para a seguinte sequência de chaves: 1, 2, 3, 4, 5, 11, 12, 13, 14, 15, 6, 7, 8, 9, 10, 16, 17, 18, 19, 20, 21, 22. (2,0 pontos)

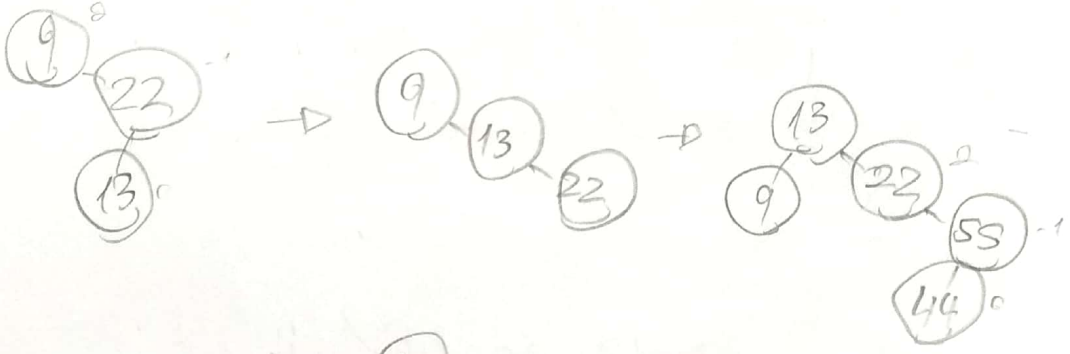
Obs: todos os nós podem armazenar no **máximo** $2f$-1 elementos. A raiz pode armazenar no **mínimo** 1 elemento e os outros nós no **mínimo** f-1 elementos.

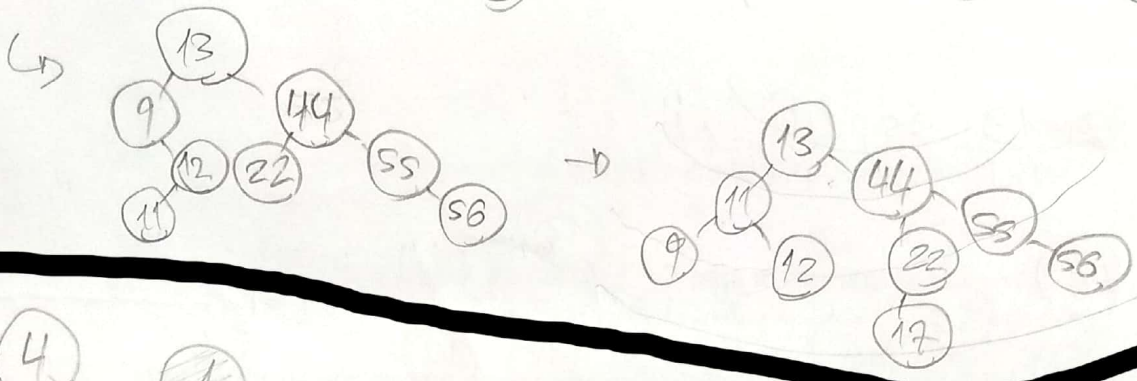
2AP Estruturas 2016.2

1

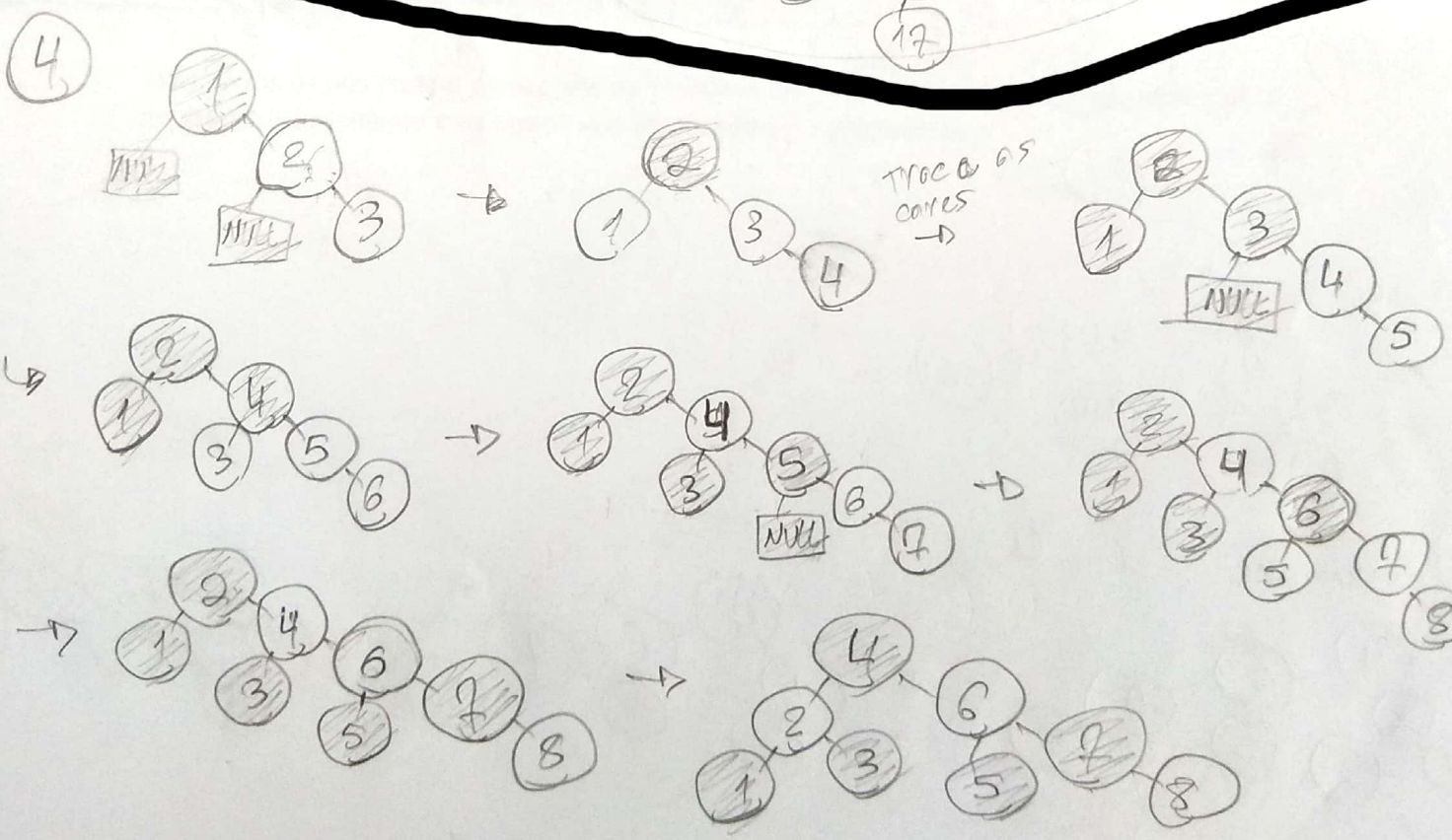


3





4



(2) BUBBLE SORT

É COMPOSTO POR UM LAÇO for PRINCIPAL ONDE i RECEBE O TAMANHO n DO VETOR V E VAI DECREMENTANDO ATÉ CHEGAR EM ZERO

↳ `for(i = n; i > 0; i--) { ... }`

- DENTRO DESTE HÁ OUTRO LAÇO for QUE VAI DE ZERO ATÉ i, ONDE HÁ UMA CONDIÇÃO SE V[j] > V[j+1] ESSES ELEMENTOS SÃO TROCADOS

↳
```
for (j=0; j<i; j++) {
    if (V[j] > V[j+1]) {
        temp = V[j];
        V[j] = V[j+1];
        V[j+1] = temp;
    }
}
```

*ESSE PROCESSO LEVA OS MAIORES ELEMENTOS PARA O FINAL DO VETOR (EM SUAS POSIÇÕES CORRETAS) DEPOIS ESSES ELEMENTOS SÃO ISOLADOS LÁ, JÁ QUE j SÓ VAI ATÉ O TAMANHO DE i E i SEMPRE É DECREMENTADO, ENTÃO O PROCESSO CONTINUA PARA O RESTO DO VETOR ATÉ QUE TODOS OS ELEMENTOS ESTEJAM ORDENADOS, O PROCESSO NÃO É MUITO EFICIENTE NA PRÁTICA.

COMPLEXIDADE

↳ PIOR CASO: $O(n^2)$

↳ MELHOR CASO: $\Omega(n^2)$

2 (continuação 1)

05) QUICKSORT → DIVIDIR PARA CONQUISTAR

O ALGORITIMO É UMA FUNÇÃO QUE RECEBE COMO PARAMETRO O TAMANHO n DO VETOR E UM PONTEIRO PARA UM VETOR V

↳ rapida (n, *V) {...}

INICIAMOS ESCOLHENDO UM PIVÔ X, QUE NESSE CASO SERÁ O PRIMEIRO ELEMENTO DO VETOR V[0]

↳ x = V[0];

CRIAMO UMA VARIAVEL a QUE SERÁ INICIALIZADA COM 1

↳ a = 1;

E TAMBÉM OUTRA VARIAVEL b QUE SERÁ INICIALIZADA COM n-1

↳ b = n-1;

DENTRO DE UM LAÇO INFINITO (while (1) {...}) FAREMOS O SEGUINTE:

- CRIAMO UM LAÇO QUE VAI INCREMENTA O VALOR DE a ENQUANTO V[a] ≤ X E ENQUANTO a < n

↳ while (V[a] ≤ x && a < n) { a++

- CRIAMO UM LAÇO QUE DECREMENTA O VALOR DE b ENQUANTO V[b] > X

↳ while (V[a] > x) { b--; }

- TESTAMOS A CONDIÇÃO SE (a < b)

SE ESSA CONDIÇÃO FOR VERDADEIRA TROCAMOS OS VALORES V[a] E V[b] DE LUGAR E

2 (continuação 2)

E INCREMENTAMOS EM a E DECREMENTAMOS b.
SE A CONDIÇÃO NÃO FOR VERDADEIRA NÓS SAÍMOS DO LAÇO INFINITO

```
if (a<b) {
    temp = V[a];
    V[a] = V[b];
    V[b] = temp;
    a++;
    b--;
} else {
    break;
}
```

* QUANDO A FUNÇÃO SAI DO do *if* OS OUTROS DOIS *while* SÃO TESTADOS NOVAMENTE E O VALOR DE a E DE b PODEM SER ALTERADOS

QUANDO SAIRMOS DO *while (1)* NESSE PONTO A POSIÇÃO CORRETA DO PIVÔ SERÁ A POSIÇÃO b DO VETOR, OU SEJA V[b], ENTÃO TROCAMOS V[0]=x E V[b] DE LUGAR.

```
V[0] = V[b];
V[b] = x;
```

ENTÃO APLICAMOS A FUNÇÃO RECURSIVAMENTE À ESQUERDA DA POSIÇÃO b DO VETOR E À DIREITA

```
rapida(b, V); //esquerda do vetor principal
rapida(n-a, &V[a]); //direita do vetor principal
```

NO FINAL O VETOR ESTARÁ ORDENADO.

CONTINUA...

2 (continuação 3)

A COMPLEXIDADE DO QUICKSORT

↳ PIOR CASO: $O(n^2)$

↳ MELHOR CASO: $\Omega(n \log(n))$

↳ CASO MÉDIO: $\Omega(n \log(n))$