TT12 – Unidade VII:

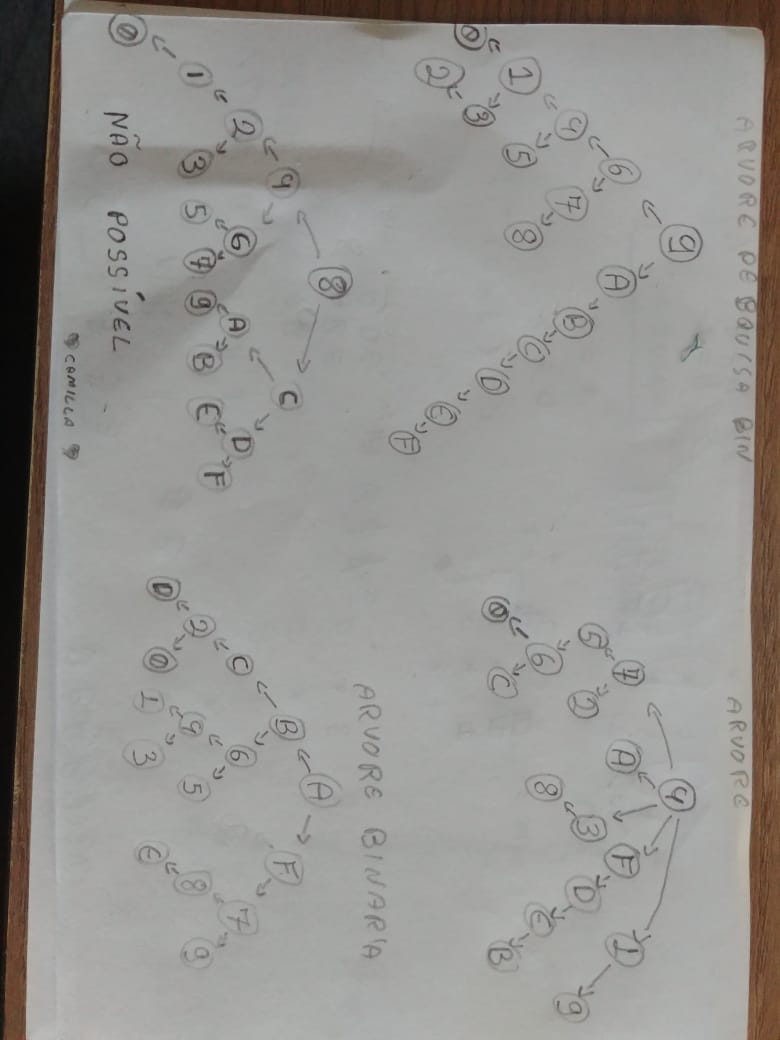
Árvores Binárias:

David de Sá Vieira de Faria – 699415

**Obs: todos códigos disponíveis no diretório ./Exercicios\_praticos/Unidade07a**

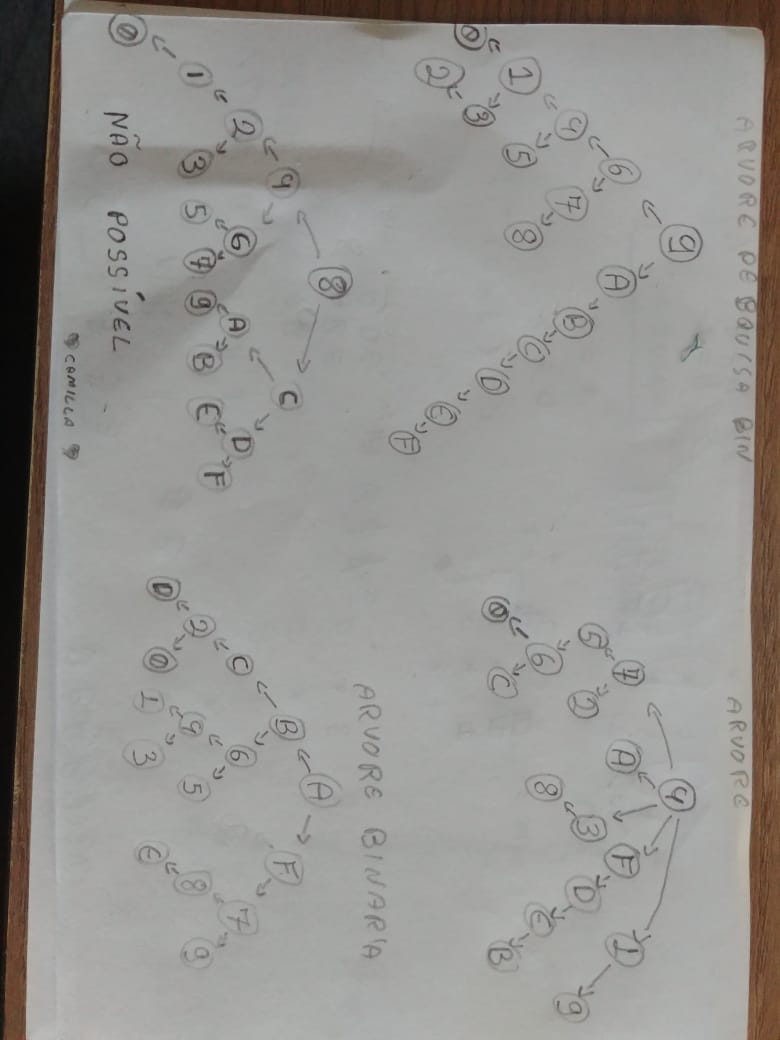
**Exercício Resolvido (1) – Slide 12:**

Crie uma árvore com os dígitos hexadecimais.

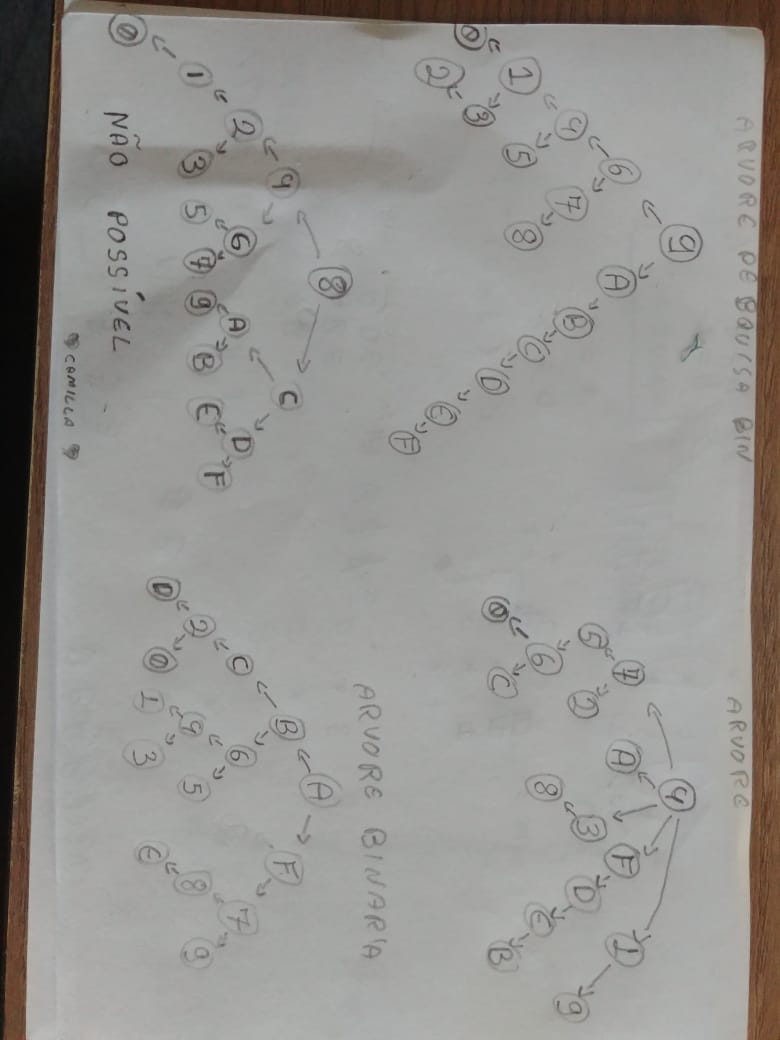


**Exercício Resolvido (2) – Slide 16:**

Crie uma árvore binária com os dígitos hexadecimais.

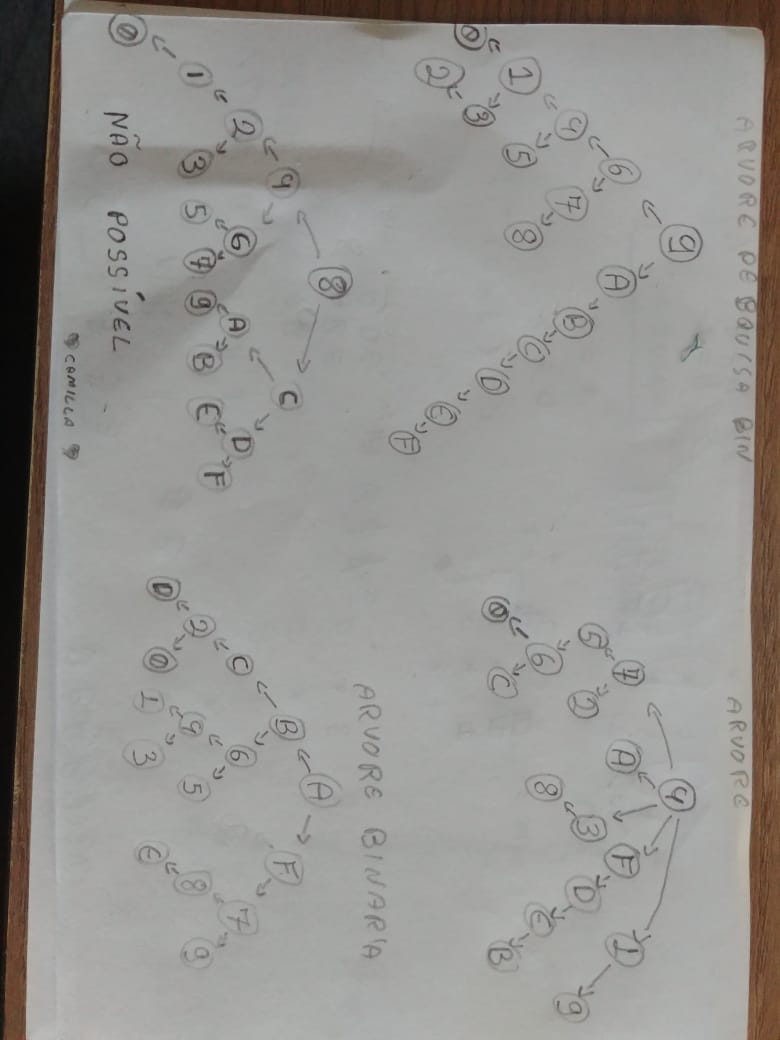
****

**Exercício Resolvido (3) – Slide 20:**

Crie uma árvore binária de pesquisa com os dígitos hexadecimais. ****

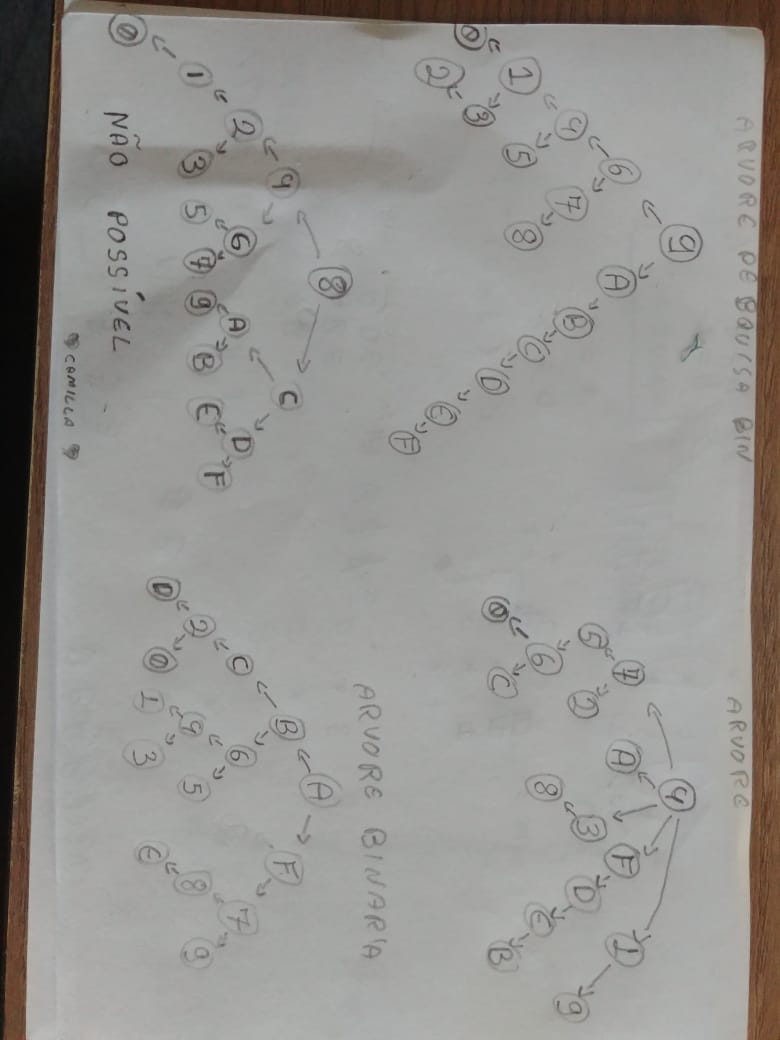
**Exercício Resolvido (4) – Slide 23:**

Crie uma árvore binária completa com os dígitos hexadecimais.

****Não é possível pela fato de estar sobrando uma folha.

**Exercício Resolvido (5) – Slide 25:**

Crie uma árvore binária completa com os dígitos hexadecimais não nulos.

****

**Exercício – Slide 33:**

Seja nossa Fila, faça um método que inverta a ordem dos seus elementos.

|  |
| --- |
| void inverter () {      Celula fim = ultimo;      while (primeiro != fim){          Celula nova = new Celula (primeiro.prox.elemento);          nova.prox = fim.prox;          fim.prox = nova;          Celula tmp = primeiro.prox;          primeiro.prox = tmp.prox;          nova = tmp = tmp.prox = null;          if (ultimo == fim)              ultimo = ultimo.prox;      } // end while      fim = null;  } // end inverter() |

**Exercício (1) – Slide 235:**

Faça um método que retorna a altura da árvore. Em seguida, insira vários elementos de forma aleatória. Para cada inserção, mostre na tela o número de elementos da árvore, o logaritmo (base 2) desse número e a altura.

**Exercício (2) – Slide 236:**

Faça um método que retorne a soma dos elementos existentes na árvore.

    public int sumElementos(){

        return sumElementos(raiz);

    } // end nElementos()

    public int sumElementos(No i){

        int n = 0;

        if(i != null){

            n = i.elemento;

            n += sumElementos(i.esq);

            n += sumElementos(i.dir);

        } // end if

        return n;

    } // end nElementos()

**Exercício (3) – Slide 237:**

Faça um método que retorne o número de elementos pares existentes na árvore.

    public int sumElementosPares(No i){

        int n = 0;

        if(i != null){

            n = (i.elemento % 2 == 0) ? i.elemento : 0;

            n += sumElementosPares(i.esq);

            n += sumElementosPares(i.dir);

        } // end if

        return n;

    } // end nElementos()

**Exercício (4) – Slide 238:**

Faça um método estático que recebe dois objetos do tipo árvore binária e retorne um booleano indicando se as duas árvores são iguais.

    private static boolean igual2(No i1, No i2) {

        boolean resp;

        if (i1 != null && i2 != null) {

            resp = (i1.elemento == i2.elemento) && igual2(i1.esq, i2.esq) && igual2(i1.dir, i2.dir);

        } else if (i1 == null && i2 == null) {

            resp = true;

        } else {

            resp = false;

        }

        return resp;

    }

**Exercício (5) – Slide 239:**

Faça um método que retorna true se a árvore contém algum número divisível por onze.

    public boolean div11(No i){

        boolean resp = false;

        if(i != null){

            resp = (i.elemento % 11 == 0) || div11(i.esq) || div11(i.dir);

        } // end if

        return resp;

    } // end nElementos()

**Exercício (6) – Slide 240:**

Um algoritmo de ordenação é o TreeSort que insere os elementos do array em uma árvore binária e utiliza um "mostrar" para ordenar os elementos do array. Implemente o TreeSort e faça a análise de complexidade do mesmo.

    public static void treeSort(int[] array){

        ArvoreBinaria tree = new ArvoreBinaria();

        for(int i = 0; i < array.length; i++){

            try{

                tree.inserir(array[i]);

            } catch(Exception e){}

        } // end inserir

        tree.showArvore();

    } // end treeSort()

**Análise de complexidade:**

|  |  |
| --- | --- |
| **Melhor caso** | **Pior caso** |
| Ө(1) + Ө(n) + Ө(n) = Ө(n) | Ө(lg(n)) + Ө(n) + Ө(n) = Ө(lg(n)) |

**Exercício (7) – Slide 241:**

Faça o método No toArvoreBinaria(Celula primeiro, CelulaDupla primeiro) que recebe o nó cabeça de uma lista simples e o de outra dupla. Em seguida, crie uma árvore binária contendo os elementos intercalados das duas listas e retorne o endereço do nó raiz da árvore criada.

    public No toArvoreBinaria(Celula\_ListaSimples i, Celula\_ListaDupla j){

        while((i != null) || (j != null)){

            try {

                inserir(i.elemento);

            } catch (Exception e) {}

            try {

                inserir(j.elemento);

            } catch (Exception e) {}

            if(i != null)

                i = i.prox;

            if(j != null)

                j = j.prox;

        } // end for

        return raiz;

    } // end toArvoreBinaria()

**Exercício (8) – Slide 352:**

O método remover privado e recursivo apresentado em nossa árvore recebe e um valor e retorna um No. Altere tal método para que o mesmo retorne void.

**Exercício (10) – Slide 444:**

Você foi contratado para desenvolver uma agenda de contatos (atributos nome, telefone, email e CPF) para um escritório de advocacia

* Crie uma classe Contato contendo os atributos String nome, telefone e email e int CPF
* Crie uma classe Celula contendo os atributos Contato contato e Celula prox
* Crie uma classe No contendo os atributos Celula primeiro e ultimo, No esq e dir, e char letra
* Crie uma classe Agenda contendo o atributo No raiz, os métodos inserir(Contato contato), remover(String nome), pesquisar(String nome) e pesquisar(int cpf). Para cada método, mostre o melhor e pior caso

**Obs: Codigo fonte pode ser encontrado na pasta Exercicio10**

|  |  |  |
| --- | --- | --- |
|  | **Melhor caso** | **Pior caso** |
| Inserir() | **O()** |  |
| Remover() |  |  |
| Pesquisar(String nome) |  |  |
| Pesquisar(int cpf) |  |  |