Questão 1:

```
package src;

import java.util.*;

public class Teste {

    public static void main(String[] args) throws Exception {

        Scanner sc = new Scanner(System.in);

        int num = 0;
        AVL a = new AVL();

        int opcao = 0;
        while (opcao != 7) {
            System.out.println("Menu de opções:");
            System.out.println("1- Inserir um número na árvore AVL");
            System.out.println("2- Remover um número da árvore AVL");
            System.out.println("3- Pesquisar um número na árvore AVL");
            System.out.println("4- Mostrar todos os elementos da árvore
AVL, usando o caminhamento central");
            System.out.println("5- Mostrar todos os elementos da árvore
AVL, usando o caminhamento pós-ordem");
            System.out.println("6- Mostrar todos os elementos da árvore
AVL, usando o caminhamento pré-ordem");
            System.out.println("7- Sair");
            System.out.print("Escolha uma opção: ");
            opcao = sc.nextInt();

            switch (opcao) {
                case 1:
                    System.out.print("Digite o número a ser inserido: ");
                    num = sc.nextInt();
                    a.inserir(num);
                    break;
                case 2:
                    System.out.print("Digite o número a ser removido: ");
                    num = sc.nextInt();
                    a.remover(num);
                    break;
                case 3:
                    System.out.print("Digite o número a ser pesquisado:
");
                    num = sc.nextInt();
                    if (a.pesquisar(num)) {
```

```
                        System.out.println("Número encontrado na
árvore.");
                    } else {
                        System.out.println("Número não encontrado na
árvore.");
                    }
                    break;
                case 4:
                    System.out.println("Elementos da árvore (caminhamento
central):");
                    a.caminharCentral();
                    break;
                case 5:
                    System.out.println("Elementos da árvore (caminhamento
pós-ordem):");
                    a.caminharPos();
                    break;
                case 6:
                    System.out.println("Elementos da árvore (caminhamento
pré-ordem):");
                    a.caminharPre();
                    break;
                case 7:
                    System.out.println("Encerrando o programa.");
                    break;
                default:
                    System.out.println("Opção inválida. Por favor,
escolha uma opção válida.");
                    break;
            }
        }

        sc.close();

    }
}

package src;

public class AVL {
    private No raiz;

    public AVL() {
        raiz = null;
    }

    public boolean pesquisar(int x) {
        return pesquisar(x, raiz);
    }
```

```
    private boolean pesquisar(int x, No i) {
        boolean resp;
        if (i == null) {
            resp = false;
        } else if (x == i.elemento) {
            resp = true;
        } else if (x < i.elemento) {
            resp = pesquisar(x, i.esq);
        } else {
            resp = pesquisar(x, i.dir);
        }
        return resp;
    }

    public void caminharCentral() {
        System.out.print("[ ");
        caminharCentral(raiz);
        System.out.println("]");
    }

    private void caminharCentral(No i) {
        if (i != null) {
            caminharCentral(i.esq);
            System.out.print(i.elemento + " ");
            caminharCentral(i.dir);
        }
    }

    public void caminharPre() {
        System.out.print("[ ");
        caminharPre(raiz);
        System.out.println("]");
    }

    private void caminharPre(No i) {
        if (i != null) {
            System.out.print(i.elemento + "(fator " + (No.getNivel(i.dir)
-
                    No.getNivel(i.esq)) + ") ");
            caminharPre(i.esq);
            caminharPre(i.dir);
        }
    }

    public void caminharPos() {
        System.out.print("[ ");
        caminharPos(raiz);
        System.out.println("]");
```

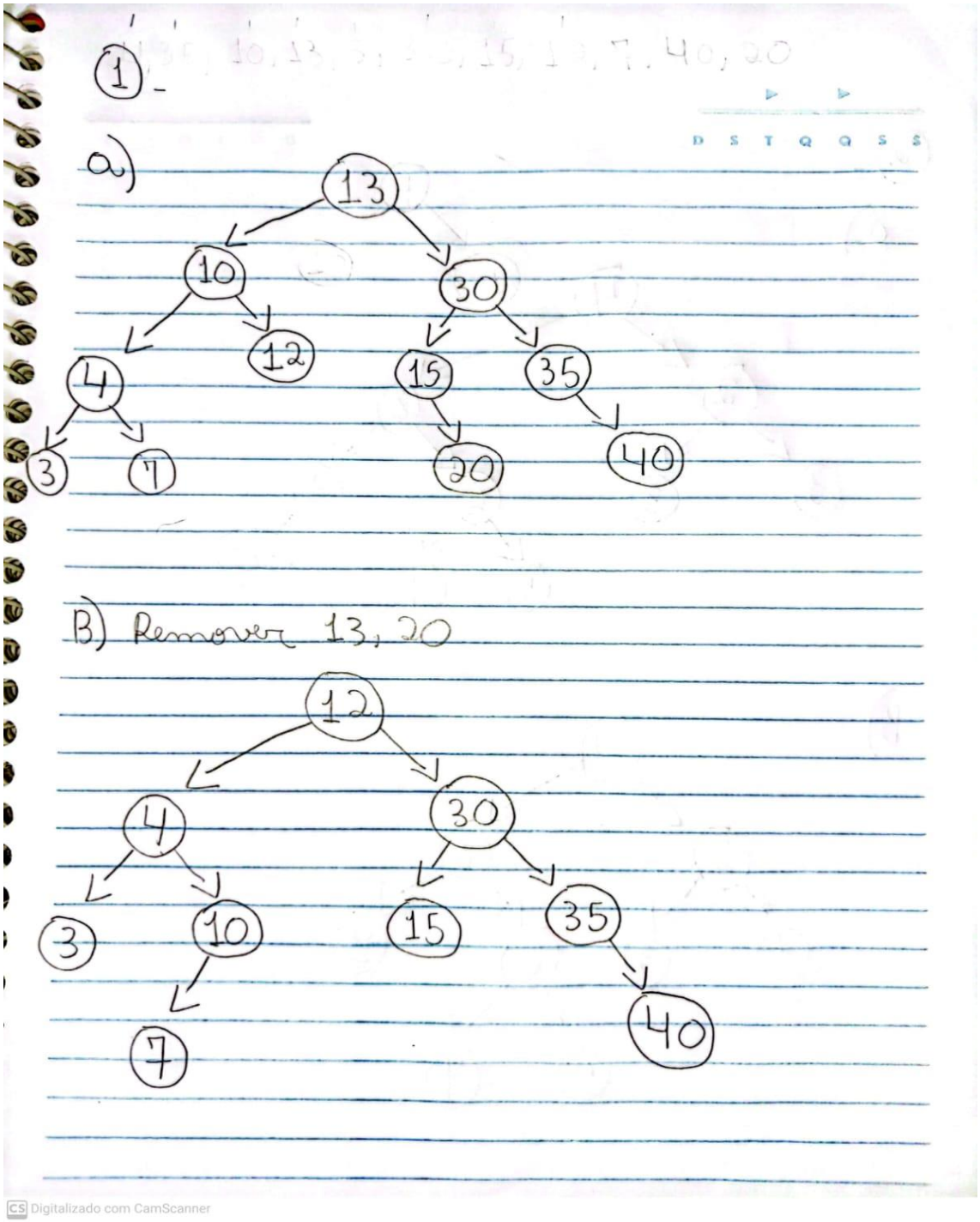
```
}

private void caminharPos(No i) {
    if (i != null) {
        caminharPos(i.esq);
        caminharPos(i.dir);
        System.out.print(i.elemento + " ");
    }
}

public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}

private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro ao inserir!");
    }
    return balancear(i);
}

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) {
        throw new Exception("Erro ao remover!");
    } else if (x < i.elemento) {
        i.esq = remover(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = remover(x, i.dir);
    } else if (i.dir == null) {
        i = i.esq;
    } else if (i.esq == null) {
        i = i.dir;
    } else {
        i.esq = maiorEsq(i, i.esq);
    }
    return balancear(i);
}

private No maiorEsq(No i, No j) {
```

```
        // Encontrou o maximo da subarvore esquerda.
        if (j.dir == null) {
            i.elemento = j.elemento; // Substitui i por j.
            j = j.esq; // Substitui j por j.ESQ.
            // Existe no a direita.
        } else {
            // Caminha para direita.
            j.dir = maiorEsq(i, j.dir);
        }
        return j;
    }

    private No balancear(No no) throws Exception {
        if (no != null) {
            int fator = No.getNivel(no.dir) - No.getNivel(no.esq);
            // Se balanceada
            if (Math.abs(fator) <= 1) {
                no.setNivel();
                // Se desbalanceada para a direita
            } else if (fator == 2) {
                int fatorFilhoDir = No.getNivel(no.dir.dir) -
No.getNivel(no.dir.esq);
                // Se o filho a direita tambem estiver desbalanceado
                if (fatorFilhoDir == -1) {
                    no.dir = rotacionarDir(no.dir);
                }
                no = rotacionarEsq(no);
                // Se desbalanceada para a esquerda
            } else if (fator == -2) {
                int fatorFilhoEsq = No.getNivel(no.esq.dir) -
No.getNivel(no.esq.esq);
                // Se o filho a esquerda tambem estiver desbalanceado
                if (fatorFilhoEsq == 1) {
                    no.esq = rotacionarEsq(no.esq);
                }
                no = rotacionarDir(no);
            } else {
                throw new Exception(
                        "Erro no No(" + no.elemento + ") com fator de
balanceamento ("
                                + fator + ") invalido!");
            }
        }
        return no;
    }

    private No rotacionarDir(No no) {
        System.out.println("Rotacionar DIR(" + no.elemento + ")");
        No noEsq = no.esq;
```

```
        No noEsqDir = noEsq.dir;
        noEsq.dir = no;
        no.esq = noEsqDir;
        no.setNivel(); // Atualizar o nivel do no
        noEsq.setNivel(); // Atualizar o nivel do noEsq
        return noEsq;
    }

    private No rotacionarEsq(No no) {
        System.out.println("Rotacionar ESQ(" + no.elemento + ")");
        No noDir = no.dir;
        No noDirEsq = noDir.esq;
        noDir.esq = no;
        no.dir = noDirEsq;
        no.setNivel(); // Atualizar o nivel do no
        noDir.setNivel(); // Atualizar o nivel do noDir
        return noDir;
    }
}


package src;
public class No {
    public int elemento;
    public No esq;
    public No dir;
    public int nivel; // Numero de niveis abaixo do no

    public No(int elemento) {
        this(elemento, null, null, 1);
    }

    public No(int elemento, No esq, No dir, int nivel) {
        this.elemento = elemento;
        this.esq = esq;
        this.dir = dir;
        this.nivel = nivel;
    }

    /**
     * Calculo do numero de niveis a partir de um vertice
     */
    public void setNivel() {
        this.nivel = 1 + Math.max(getNivel(esq), getNivel(dir));
    }

    /**
     * Retorna o numero de niveis a partir de um vertice
     *
```

```
     * @param No no que se deseja o nivel.
     */
    public static int getNivel(No no) {
        return (no == null) ? 0 : no.nivel;
    }
}
```

Questão 2:



Questão 3: