Prof: Ana Paula

Nome : Linekker Emmanuel

SI

LISTA 11

1)

```java
package src;

import java.util.*;

public class Teste {

    public static int pesquisaBinaria(int[] array, int x) {

        int dir = array.length - 1, esq = 0, meio;
        while (esq <= dir) {
            meio = (esq + dir) / 2;
            if (x == array[meio]) {
                return meio;
            } else if (x > array[meio]) {
                esq = meio + 1;
            } else {
                dir = meio - 1;
            }
        }
        return -1;

    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n, vet[];

        System.out.println("Digite o tamanho do Vetor");
        n = sc.nextInt();

        vet = new int[n];

        System.out.println("Preencha o vetor");
        for (int i = 0; i < vet.length; i++) {
            vet[i] = sc.nextInt();
        }

        System.out.println("digite o numero a pesquisar");
```

```
        int num = sc.nextInt();

        if (pesquisaBinaria(vet, num) == -1) {
            System.out.println("O elemento não existe");
        } else {

            System.out.println("O numero " + num + " esta na posição " +
pesquisaBinaria(vet, num) + " Encontrado");

        }

        sc.close();
    }
}
```

Questões 2,3 e 4 :

```
package src;

import java.util.*;

public class Teste {

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

        Scanner sc = new Scanner(System.in);

        ArvoreBinaria arvoreBinaria = new ArvoreBinaria();
        int num;

        int opcao = 0;
        while (opcao != 9) {
            System.out.println("\nMenu de opções:");
            System.out.println("1- Inserir um número na árvore binária");
            System.out.println("2- Remover um número da árvore binária");
            System.out.println("3- Pesquisar um número na árvore
binária");
            System.out.println("4- Mostrar o maior elemento da árvore
binária");
            System.out.println("5- Mostrar o menor elemento da árvore
binária");
            System.out.println("6- Mostrar todos os elementos da árvore,
usando o caminhamento central");
            System.out.println("7- Mostrar todos os elementos da árvore,
usando o caminhamento pós-ordem");
```

```
            System.out.println("8- Mostrar todos os elementos da árvore,
usando o caminhamento pré-ordem");
            System.out.println("9- Sair");

            System.out.print("Escolha uma opção: ");
            opcao = sc.nextInt();

            switch (opcao) {
                case 1:
                    System.out.print("Digite o número a ser INSERIDO na
árvore: ");
                    num = sc.nextInt();
                    arvoreBinaria.inserir(num);
                    break;

                case 2:
                    System.out.print("Digite o número a ser REMOVIDO na
árvore: ");
                    num = sc.nextInt();
                    arvoreBinaria.remover(num);
                    break;

                case 3:
                    System.out.print("Digite o número a ser PESQUISADO na
árvore: ");
                    num = sc.nextInt();
                    if (arvoreBinaria.pesquisar(num)) {
                        System.out.println("Número Encontrado");
                    } else {
                        System.out.println("Não Encontrado");
                    }
                    break;

                case 4:
                    int maior = arvoreBinaria.getMaior();
                    System.out.println("Maior elemento da árvore: " +
maior);
                    break;

                case 5:
                    int menor = arvoreBinaria.getMenor();
                    System.out.println("Menor elemento da árvore: " +
menor);
                    break;

                case 6:
                    System.out.println("Elementos da árvore (caminhamento
central):");
                    arvoreBinaria.caminharCentral();
```

```
                    break;

                case 7:
                    System.out.println("Elementos da árvore (caminhamento
pós-ordem):");
                    arvoreBinaria.caminharPos();
                    break;

                case 8:
                    System.out.println("Elementos da árvore (caminhamento
pré-ordem):");
                    arvoreBinaria.caminharPre();
                    break;

                case 9:
                    System.out.println("Fim do programa!");

                    break;

                default:
                    System.out.println("Opção inválida!");
                    break;

            }
        }
    }
}


package src;

public class ArvoreBinaria {
   private No raiz;

   ArvoreBinaria() {
      raiz = null;
   }

   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!");
   }
   return i;
}

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() {
   caminharCentral(raiz);
}

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

public void caminharPre() {
   caminharPre(raiz);
}

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

```
public void caminharPos() {
   caminharPos(raiz);
}

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

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!");
   } 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 i;
}

private No maiorEsq(No i, No j) {
   if (j.dir == null) {
      i.elemento = j.elemento;
      j = j.esq;
   } else {
      j.dir = maiorEsq(i, j.dir);
   }
   return j;
}

public int getMaior() throws Exception {
   return getMaior(raiz);
}

private int getMaior(No i) throws Exception {
```

```
      if (i == null) {
         throw new Exception("Árvore vazia");
      }
      while (i.dir != null) {
         i = i.dir;
      }
      return i.elemento;
   }

   public int getMenor() throws Exception {
      return getMenor(raiz);
   }

   private int getMenor(No i) throws Exception {
      if (i == null) {
         throw new Exception("Árvore vazia");
      }
      while (i.esq != null) {
         i = i.esq;
      }
      return i.elemento;
   }

}
```

```
package src;

class No {
    public int elemento;
    public No esq;
    public No dir;

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

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

5)

```
package src;

import java.util.*;

public class Teste {

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

        Scanner sc = new Scanner(System.in);
        ArvoreBinaria arvore = new ArvoreBinaria();

        int opcao;
        do {
            System.out.println("\nMenu de Opções:");
            System.out.println("1- Inserir um nome na árvore binária");
            System.out.println("2- Remover um nome da árvore binária");
            System.out.println("3- Pesquisar um nome na árvore binária");
            System.out.println("4- Mostrar todos os elementos da árvore,
usando o caminhamento central");
            System.out.println("5- Mostrar todos os elementos da árvore,
usando o caminhamento pós-ordem");
            System.out.println("6- Mostrar todos os elementos da árvore,
usando o caminhamento pré-ordem");
            System.out.println("7- Sair");
            System.out.print("Escolha uma opção: ");
            opcao = sc.nextInt();
            sc.nextLine(); // Consumir a quebra de linha

            try {
                switch (opcao) {
                    case 1:
                        System.out.print("Digite o nome a ser inserido:
");
                        String nome = sc.nextLine();
                        arvore.inserir(nome);
                        System.out.println("Nome inserido com sucesso!");
                        break;
                    case 2:
                        System.out.print("Digite o nome a ser removido:
");
                        String nomeRemover = sc.nextLine();
                        arvore.remover(nomeRemover);
```

```
                        System.out.println("Nome removido com sucesso!");
                        break;
                    case 3:
                        System.out.print("Digite o nome a ser pesquisado:
");
                        String nomePesquisar = sc.nextLine();
                        boolean encontrado =
arvore.pesquisar(nomePesquisar);
                        if (encontrado) {
                            System.out.println("O nome está presente na
árvore.");
                        } else {
                            System.out.println("O nome não foi encontrado
na árvore.");
                        }
                        break;
                    case 4:
                        System.out.println("Elementos da árvore usando o
caminhamento central:");
                        arvore.caminharCentral();
                        break;
                    case 5:
                        System.out.println("Elementos da árvore usando o
caminhamento pós-ordem:");
                        arvore.caminharPos();
                        break;
                    case 6:
                        System.out.println("Elementos da árvore usando o
caminhamento pré-ordem:");
                        arvore.caminharPre();
                        break;
                    case 7:
                        System.out.println("Saindo do programa...");
                        break;
                    default:
                        System.out.println("Opção inválida! Digite um
número válido.");
                        break;
                }
            } catch (Exception e) {
                System.out.println("Erro: " + e.getMessage());
            }
        } while (opcao != 7);

        sc.close();

    }
}
```

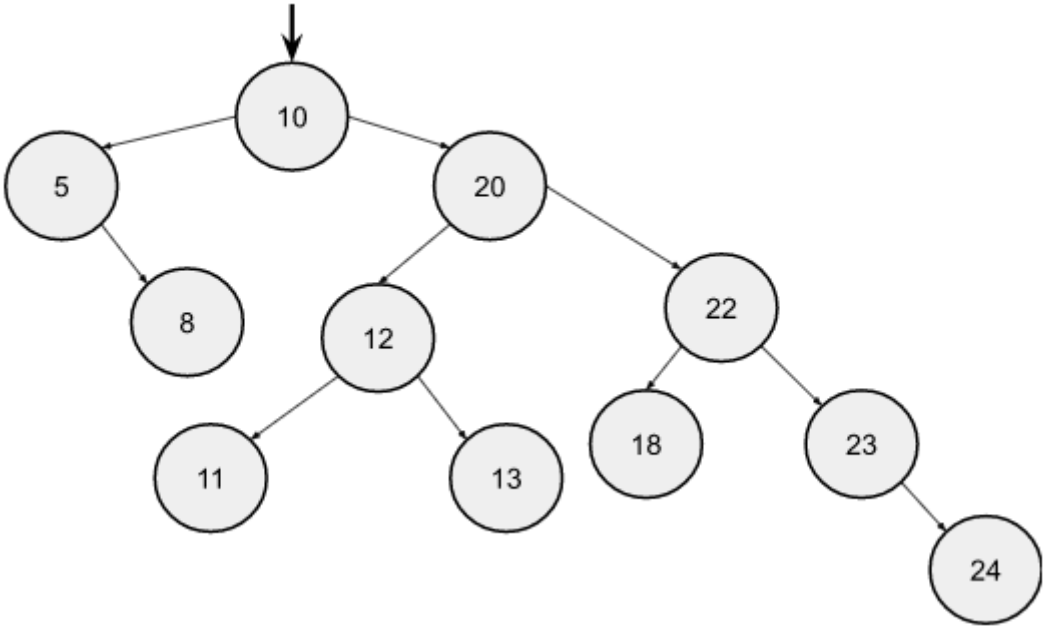
```java
package src;

public class ArvoreBinaria {
   private No raiz;

   public ArvoreBinaria() {
      raiz = null;
   }

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

   private No inserir(String x, No i) throws Exception {
      if (i == null) {
         i = new No(x);
      } else if (x.compareToIgnoreCase(i.elemento) < 0) {
         i.esq = inserir(x, i.esq);
      } else if (x.compareToIgnoreCase(i.elemento) > 0) {
         i.dir = inserir(x, i.dir);
      } else {
         throw new Exception("Erro!");
      }
      return i;
   }

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

   private boolean pesquisar(String x, No i) {
      boolean resp;
      if (i == null) {
         resp = false;
      } else if (x.compareToIgnoreCase(i.elemento) == 0) {
         resp = true;
      } else if (x.compareToIgnoreCase(i.elemento) < 0) {
         resp = pesquisar(x, i.esq);
      } else {
         resp = pesquisar(x, i.dir);
      }
      return resp;
   }

   public void caminharCentral() {
      caminharCentral(raiz);
```

```
}

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

public void caminharPre() {
   caminharPre(raiz);
}

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

public void caminharPos() {
   caminharPos(raiz);
}

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

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

private No remover(String x, No i) throws Exception {
   if (i == null) {
      throw new Exception("Erro!");
   } else if (x.compareToIgnoreCase(i.elemento) < 0) {
      i.esq = remover(x, i.esq);
   } else if (x.compareToIgnoreCase(i.elemento) > 0) {
      i.dir = remover(x, i.dir);
   } else if (i.dir == null) {
      i = i.esq;
   } else if (i.esq == null) {
      i = i.dir;
```

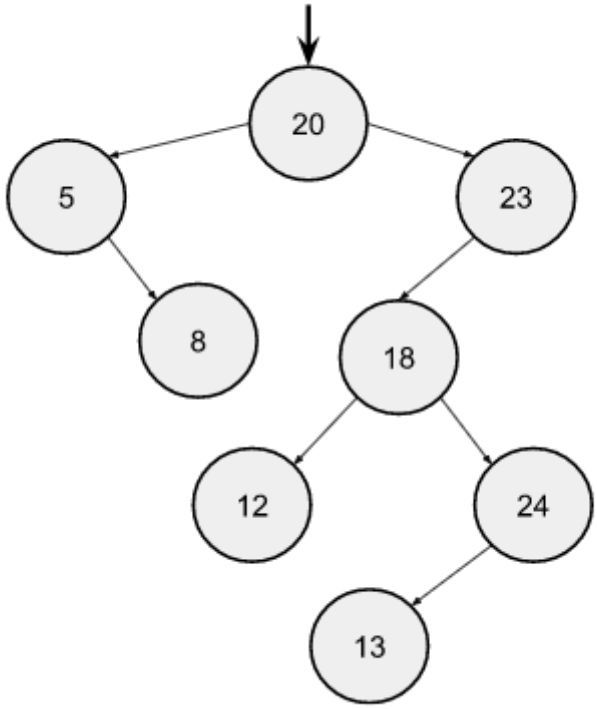
```
      } else {
         i.esq = maiorEsq(i, i.esq);
      }

      return i;
   }

   private No maiorEsq(No i, No j) {
      if (j.dir == null) {
         i.elemento = j.elemento;
         j = j.esq;
      } else {
         j.dir = maiorEsq(i, j.dir);
      }
      return j;
   }

}
```

```
package src;

class No {
    public String elemento;
    public No esq;
    public No dir;

    No(String elemento) {
        this(elemento, null, null);
    }

    No(String elemento, No esq, No dir) {
        this.elemento = elemento;
        this.esq = esq;
        this.dir = dir;
    }
}
```

6) a)



b)

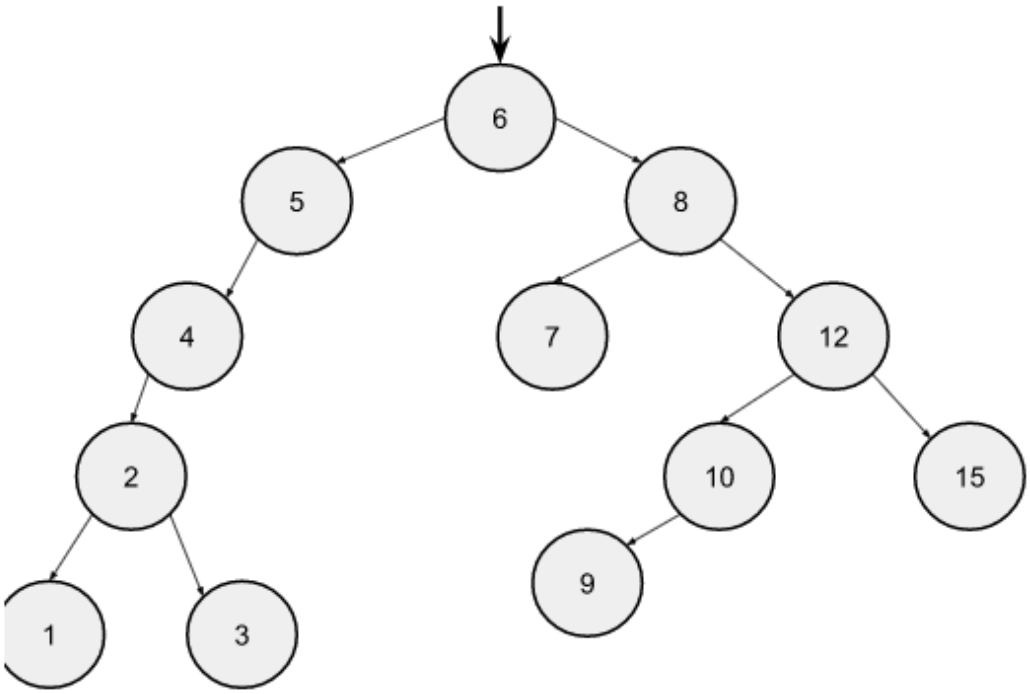


c) Pré-Ordem: 10 5 8 20 12 11 13 22 18 23 24

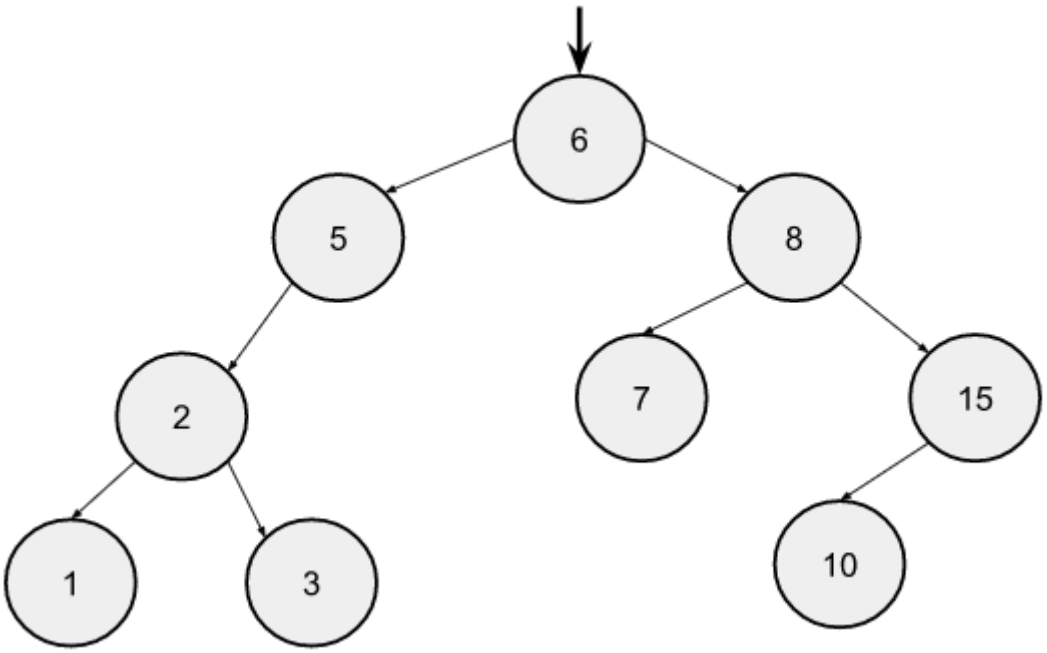
Pós-Ordem:

Central: 5 8 10 11 12 13 18 20 22 23 24

7) a)



b)



c) Pré-Ordem: 6 5 4 2 1 3 8 7 12 10 9 15

Pós-Ordem:

Central: 1 2 3 5 6 7 8 10 15