Lista de revisão 3 – Breno Henrique Silva Carvalho

Pesquisa Sequencial e Binaria

1)

```java
public class Main {
    public static boolean ExisteNota(int notas[], int notaProcurada) {
        for (int i : notas) {
            if (i == notaProcurada) {
                return true;
            }
        }
        return false;
    }
    public static int Pesquisar(int[] notas, int notaProcurada){
        for (int i = 0; i< notas.length; i++) {
            if(notaProcurada == notas[i])
            return i;
        }
        return -1;
    }


    Run | Debug
    public static void main(String[] args) {
        Scanner ent = new Scanner(System.in);
        int notas[] = new int[] { 11, 5, 3, 6, 12, 2 };

        System.out.print(s: "Digite a nota que deseja procurar: ");
        var nota = ent.nextInt();
        if(ExisteNota(notas, nota))
        {
            System.out.println("Indice: " + Pesquisar(notas, nota));
        }
        else
        {
            System.out.println(x: "Nota não encontrada");
        }

        ent.close();
    }
}
```

2) O algoritmo de busca linear é um algoritmo O(n). No melhor caso, o elemento a ser buscado é encontrado logo na primeira tentativa da busca. No pior caso, o elemento a ser buscado encontra-se na última posição e são feitas N comparações, sendo N o número total de elementos. No caso médio, o elemento é encontrado após (N+1)/2 comparações

3) A modificação necessária é a mudança para uma pesquisa binaria, onde apresenta-se mais eficiente que a busca linear.

4)

```java
public static boolean pesquisarNumero(int x, int[] notas) {

    for (int i = 0; i < notas.length; i++) {
        if (notas[i] == x) {
            System.out.println("Nota: " + x + " Encontrada");
            return true;
        }
    }
    return false;
}

public static boolean buscaBinaria(int p, int[] notas) {
    int inicio = 0;
    int fim = notas.length - 1;
    int meio;

    Arrays.sort(notas);

    while (inicio <= fim) {
        meio = (inicio + fim) / 2;
        if (notas[meio] == p) {
            System.out.println("Nota: " + p + " Encontrada");
            return true;
        } else if (notas[meio] < p)
            inicio = meio + 1;
        else if (notas[meio] > p)
            fim = meio - 1;
    }
    return false;

}
}
```

5)

```java
public class Questao5 {
    Run | Debug
    public static void main(String[] args) {
        Random gerador = new Random();
        Scanner ent = new Scanner(System.in);
        int pesquisa = 0;
        int vetor[] = new int[120];
        for (int i = 0; i < vetor.length; i++) {
            vetor[i] = gerador.nextInt(bound: 50);
        }

        BubbleSort(vetor);

        System.out.println(x: "Vetor ordenado: ");
        for (int i : vetor) {
            System.out.print(i + "  ");
        }
        System.out.println();

        System.out.println(x: "Digite a nota a ser pesquisada: ");
        pesquisa = ent.nextInt();

        System.out
                .println(pesquisarSequencial(pesquisa, vetor) + " comparações foram realizadas na Pesquisa Sequencial");
        System.out.println();
        System.out.println(buscaBinaria(pesquisa, vetor) + " comparações foram realizadas na pesquisa binaria");
        ent.close();
    }

    public static void BubbleSort(int[] vetor) {
        for (int i = 0; i < vetor.length - 1; i++) {
            for (int j = 0; j < vetor.length - 1 - i; j++) {
                if (vetor[j] > vetor[j + 1]) {
                    int aux = vetor[j];
                    vetor[j] = vetor[j + 1];
                    vetor[j + 1] = aux;
                }
            }
        }
    }
```

```java
public static int pesquisarSequencial(int x, int[] notas) {
    int contagem = 0;
    for (int i = 0; i < notas.length; i++) {
        contagem++;
        if (notas[i] == x) {
            System.out.println("Nota: " + x + " Encontrada");
            return contagem;
        }
    }
    return contagem;
}

public static int buscaBinaria(int p, int[] notas) {
    int inicio = 0;
    int fim = notas.length - 1;
    int meio;
    var contagem = 0;

    while (inicio <= fim) {
        contagem++;
        meio = (inicio + fim) / 2;
        if (notas[meio] == p) {
            System.out.println("Nota: " + p + " Encontrada");
            return contagem;
        } else if (notas[meio] < p) {
            contagem++;
            inicio = meio + 1;
        } else if (notas[meio] > p) {
            contagem++;
            fim = meio - 1;
        }
    }
    return contagem;

}
```

Árvores binárias

1)

```java
public void Inserir(String nome, String endereco, int idade, double renda) throws Exception {
    _raiz = Inserir(nome, endereco, idade, renda, _raiz);
    tamanho++;
}

public No Inserir(String nome, String endereco, int idade, double renda, No i) throws Exception {
    if (i == null) {
        i = new No(nome, endereco, idade, renda);
    } else if (nome.compareTo(i._nome) < 0) {
        i._noEsq = Inserir(nome, endereco, idade, renda, i._noEsq);
    } else if (nome.compareTo(i._nome) > 0) {
        i._noDir = Inserir(nome, endereco, idade, renda, i._noDir);
    } else {
        throw new Exception(message: "Erro ao inserir Nó");
    }

    return i;
}
```

2)

```java
public void PesquisarIdade(int idade) {
    PesquisarIdade(idade, _raiz);
}

private void PesquisarIdade(int idade, No i) {

    if (i ≠ null) {

        if (i._idade == idade) {

            System.out.println(i._nome);
            PesquisarIdade(idade, i._noDir);
            PesquisarIdade(idade, i._noEsq);
        } else if (i._idade > idade) {
            PesquisarIdade(idade, i._noEsq);
        } else if (i._idade < idade) {
            PesquisarIdade(idade, i._noDir);
        }
    }
}
```

3)

```java
public int PesquisarRenda(double renda_min, double renda_maxima) {
    quantidade_pessoas = 0;
    PesquisarRenda(renda_min, renda_maxima, _raiz);
    return quantidade_pessoas;
}

private void PesquisarRenda(double renda_min, double renda_maxima, No i) {

    if (i != null) {

        if (i._renda > renda_min && i._renda < renda_maxima) {
            quantidade_pessoas++;
            PesquisarRenda(renda_min, renda_maxima, i._noEsq);
            PesquisarRenda(renda_min, renda_maxima, i._noDir);
        } else if (i._renda < renda_min && i._renda < renda_maxima) {

            PesquisarRenda(renda_min, renda_maxima, i._noEsq);
        } else if (i._renda > renda_min && i._renda > renda_maxima) {

            PesquisarRenda(renda_min, renda_maxima, i._noDir);
        }
    }
}
```

4)

```java
public void PesquisarNome(String nome) throws Exception {
    boolean achou = false;

    PesquisarNome(nome, _raiz, achou);
    if(!achou)
        throw new Exception(message: "Pessoa nao encontrada");

}

private void PesquisarNome(String nome, No i, boolean achou) {
    if (i != null) {
        if (i._nome.equalsIgnoreCase(nome)) {
            System.out.println("Nome: " + i._nome);
            System.out.println("Endereço: " + i._endereco);
            System.out.println("Idade: " + i._idade);
            achou = true;
        } else if (nome.compareTo(i._nome) < 0) {
            PesquisarNome(nome, i._noDir,achou);
        } else if (nome.compareTo(i._nome) > 0) {
            PesquisarNome(nome, i._noEsq,achou);
        }
    }
}
```

5)

```
public int[] PreencherVetor(ArvoreBinaria arvore) {
    var vetor = new int[arvore.getTamanho()];
    var raiz = arvore.getRaiz();
    percorrerArvore(raiz, vetor, indice: 0);

    return vetor;

}

void percorrerArvore(No raiz, int[] vetor, int indice) {
    if (raiz != null) {
        vetor[indice] = raiz._idade;
        indice++;
        percorrerArvore(raiz._noEsq, vetor, indice);
        indice++;
        percorrerArvore(raiz._noDir, vetor, indice);
        indice++;
        indice--;
    }

    return;
}
```

7)

Arvore AVL

1) São o recurso de balanceamento seja ele para calcular o balanceamento da própria arvore binaria para utilização em rotações, sejam simples ou dupla.

2) usada em sub-árvores em que o pai e o filho estão desbalanceados para a esquerda

3) usada em sub-árvores em que o pai e o filho estão desbalanceados para a direita

4) usada em sub-árvores em que o pai está desbalanceado para a direita e o filho para a esquerda

5) usada em sub-árvores em que o pai está desbalanceado para a esquerda e o filho para a direita