# SIN5013 - Exemplo de Prova do Conteúdo Semestral

**1.** Encontre as equações de recorrência dos seguintes métodos (não é necessário resolver, mas é preciso identificar precisamente suas equações de recorrência [desejamos identificar **exatamente** quantas vezes os **comandos hachurados** serão executados, **no pior caso**, em relação a n]). Não é necessário resolver as equações, apenas identifica-las:

```
static int busca (int[] A, int x, int ini, int fim){
  if (ini>fim) return -1;
  int meio = (fim+ini)/2;
  if (A[meio]==x) return meio;
  int i1 = busca(A, x, ini, meio-1);
  int i2 = busca(A, x, meio+1, fim);
  if (i1>i2) return i1;
  return i2;
}
```

(neste caso n = fim-ini+1)
Equação de Recorrência:

**T(0) = 0**
**T(n) = 2*T((n-1)/2) + 1**

```
void hanoi2(char ori, char dst, char aux, int n) {
  if(n <= 3) {
    System.out.print("Move de " + ori + " para " + dst);
  }else {
    hanoi2(ori, aux, dst, n/3);
    hanoi2(ori, dst, aux, n/3);
    hanoi2(aux, dst, ori, n/3);
  }
}
```

Equação de Recorrência:

**T(n) = 1  para n <=3**
**T(n) = 3*T(n/3)**

**2. Resolva a equação de recorrência** a seguir e identifique sua **complexidade assintótica θ** (não é preciso provar/demonstrar a complexidade assintótica):

**T(n) = 2*T(n/2) + 1 , T(1) = 1**

**1º T(n) = 2*T(n/2) + 1**
**2º T(n) = 2*(2*T(n/4) + 1) + 1 = 4*T(n/4) + 2 + 1**
**3º T(n) = 4*(2*T(n/8) + 1) + 2 + 1 = 8*T(n/8) + 4 + 2 + 1**
**iº T(n) = $2^i$*T(n/$2^i$) + $2^{(i-1)}$ ... + 4 + 2 + 1**

**T(n/2) = 2*T(n/4) + 1**
**T(n/4) = 2*T(n/8) + 1**

**T(n/$2^i$) = T(1)**
**n/$2^i$ = 1**
**n = $2^i$**
**i = log(n)**

**para i = log(n)**
**iº T(n) = $2^{log n}$*T(n/$2^{log n}$) + $2^{(log(n)-1)}$ ... + 4 + 2 + 1**
**T(n) = n*T(1) + $2^{log(n)}$*$2^{-1}$ + ... + 4 + 2 + 1**
**T(n) = n*1 + n/2 + ... + 4 + 2 + 1  (soma de PG)**

$$S_n = \frac{a_1(q^n - 1)}{q - 1} = \frac{a_1(1 - q^n)}{1 - q}.$$

**T(n) = 2*n-1  =>  T(n) ∈ $\theta(n)$**

**3.** Escreva um programa (utilizando um ou mais métodos/funções em código ou pseudo-código) que dado um grafo não direcionado com n nós **retorne o número de componentes conexos do grafo**. Este grafo pode ser representado por: (a) uma matriz de adjacências booleana com n x n células correspondendo aos n nós do grafo; ou representado por: (b) um arranjo de nós sendo que cada nó possui um arranjo de vizinhos; neste caso, considere que Nó é uma classe ou estrutura que possui os seguintes campos [ou atributos]: numVizinhos (campo do tipo inteiro com o número de vizinhos do respectivo nó), vizinhos (arranjo de ponteiros para Nós com numVizinhos elementos), visitado (campo booleano). Escolha a representação que te convém, se desejar, considere que todas as variáveis são globais/*static*. Assuma que as variáveis já estão devidamente inicializadas/preenchidas.

| ***Representação 1*** | ***Representação 2*** |
|---|---|
| int numNos = n;<br>boolean adjacencias[][] = new boolean[n][n];<br>boolean visitados[] = new boolean[n]; | int numNos = n;<br>No nos[] = new No[numNos] |

```
import java.util.LinkedList;
import java.util.Random;

public class Grafo {
    static int numNos;
    static boolean[][] adjacencias;
    static boolean[] visitados;
    static No nos[];

    public Grafo(int n, int m){
        numNos = n;
        adjacencias = new boolean[n][n];
        visitados = new boolean[n];
        nos = new No[n];

        //CRIANDO UM GRAFO COM N NOS E M ARESTAS
        Random r = new Random();
        int maxIter = 1000000;
        int arestas = 0;
        int iter = 0;
        int x, y;
        while (arestas < m && iter < maxIter){
            x = r.nextInt(n);
            y = r.nextInt(n);
            iter++;
            if (x != y && adjacencias[x][y] == false){
                arestas++;
                adjacencias[x][y] = true;
                adjacencias[y][x] = true;
            }
        }
        System.err.println("Grafo criado. " + n + " nos; " + arestas + " arestas; " + iter +
" iteracoes.");
        for (int i=0;i<n;i++){
            nos[i] = new No();
        }
```

```
        for (int i=0;i<n;i++){
            int vizinhosI = 0;
            for (int j=0;j<n;j++){
                if (adjacencias[i][j]) vizinhosI++;
            }
            nos[i].numVizinhos = vizinhosI;
            nos[i].vizinhos = new No[vizinhosI];
            int atual = 0;
            for (int j=0;j<n;j++){
                if (adjacencias[i][j]) {
                    nos[i].vizinhos[atual] = nos[j];
                    atual++;
                }
            }
        }
    }


    public static void main(String[] args){
        Grafo g = new Grafo(800,600);
        System.out.println("Componentes1: " + calcularComponentesConexos1());
        System.out.println("Componentes2: " + calcularComponentesConexos2());
        System.out.println("Componentes3: " + calcularComponentesConexos3());
        System.out.println("Componentes4: " + calcularComponentesConexos4());
    }

    // Calcula o numero de componentes conexos de maneira recursiva utilizando a matriz de
adjacencias
    private static int calcularComponentesConexos4() {
        for (int i=0;i<numNos;i++) visitados[i] = false;
        int componentes = 0;
        for (int atual=0;atual<numNos;atual++){
            if (!visitados[atual]){
                componentes++;
                visitados[atual] = true;
                visitarVizinhos(atual);
            }
        }
        return componentes;
    }

    private static void visitarVizinhos(int atual) {
        for (int v=0;v<numNos;v++){
            if (!visitados[v] && adjacencias[atual][v]){
                visitados[v] = true;
                visitarVizinhos(v);
            }
        }
    }
```

```
   // Calcula o numero de componentes conexos de maneira iterativa utilizando a matriz de
adjacencias
   private static int calcularComponentesConexos3() {
      int componentes = 0;
      LinkedList<Integer> lista = new LinkedList<Integer>();
      int atual = 0;
      for (int i=0;i<numNos;i++){
         atual = i;
         if (!visitados[atual]){
            componentes++;
            visitados[atual] = true;
            lista.add(atual);
            while (!lista.isEmpty()){
               atual = lista.remove();
               for (int v=0;v<numNos;v++){
                  if (!visitados[v] && adjacencias[atual][v]){
                        visitados[v] = true;
                        lista.add(v);
                  }
               }
            }
         }
      }
      return componentes;
   }

   // Calcula o numero de componentes conexos de maneira iterativa utilizando o arranjo de
vizinhos
   private static int calcularComponentesConexos2() {
      for (int i=0;i<numNos;i++){
         nos[i].visitado = false;
      }
      int componentes = 0;
      LinkedList<No> lista = new LinkedList<No>();
      No atual;
      for (int i=0;i<numNos;i++){
         atual = nos[i];
         if (!atual.visitado){
            componentes++;
            atual.visitado = true;
            lista.add(atual);
            while (!lista.isEmpty()){
               atual = lista.remove();
               for (int v=0;v<atual.numVizinhos;v++){
                  if (!atual.vizinhos[v].visitado){
                        atual.vizinhos[v].visitado = true;
                        lista.add(atual.vizinhos[v]);
                  }
               }
            }
         }
      }
      return componentes;
   }
```

```
    // Calcula o numero de componentes conexos de maneira recursiva utilizando o arranjo de vizinhos
    private static int calcularComponentesConexos1() {
        int componentes = 0;
        for (int i=0;i<numNos;i++){
            if (!nos[i].visitado){
                componentes++;
                nos[i].visitado = true;
                visitarVizinhos(nos[i]);
            }
        }
        return componentes;
    }

    private static void visitarVizinhos(No atual) {
        for (int v=0;v<atual.numVizinhos;v++){
            if (!atual.vizinhos[v].visitado){
                atual.vizinhos[v].visitado = true;
                visitarVizinhos(atual.vizinhos[v]);
            }
        }
    }
}

class No{
    int numVizinhos;
    No[] vizinhos;
    boolean visitado;
}
```