1. Faça os exercícios do slide 11 do arquivo "5-inducaoI.pdf"

2. Suponha um algoritmo A e um algoritmo B, com funções de complexidade de tempo $a(n) = n^2 - n + 549$ e $b(n) = 49n + 49$, respectivamente. Determine que valores de n pertencentes ao conjuntos dos números naturais para os quais A leva menos tempo para executar do que B.

3. O que significa dizer que f(n) é O(f(n))?

4. Indique se as afirmativas a seguir são verdadeiras ou falsas e justifique suas respostas.
(a) $2n+1 = O(2n)$
(b) $22n = O(2n)$
(c) $f(n) = O(u(n))$ e $g(n) = O(v(n)) \Rightarrow f(n) + g(n) = O(u(n) + v(n))$
(d) $(n + 1)5$ é $O(n^5)$
(e) $n^3 \log n$ é $\Omega(n^3)$

5. Explique a diferença entre O(1) e O(2)?

6. Qual algoritmo você prefere: um algoritmo que requer n^5 passos ou um algoritmo que requer 2^n passos?

7. Resolva as equações de recorrência:
(a) $T(n) = T(n - 1) + c$, sendo c constante e $n > 1$; $T(1)=0$.
(b) $T(n) = T(n - 1) + 2n$, para $n > 1$; $T(0)=1$.
(c) $T(n) = cT(n - 1)$, sendo c, k constantes e $n > 0$; $T(0)=k$.

8. Faça os exercícios dos slides 25 e 26 do arquivo "5-inducaoI.pdf"

9. Escreva um método em Java para obter o maior e o segundo maior elemento de um arranjo de valores int sem usar nenhum laço. Apresente também a análise desse algoritmo (função de complexidade).

10. Considere o trecho de programa abaixo:

```
...
 for(i = 0; i < n; i++)
  for( j = 0; j < m; j++)
    {
    // trecho de programa cujo custo é O(3)
    }
...
```

Qual a complexidade assintótica temporal do trecho de programa acima?

11. Considere o trecho de programa abaixo:

```
...
  for(i = 0; i < n; i++)
   {
     // trecho de programa cujo custo é O(1)
   }
  for( j = 0; j < m; j++)
     {
     // trecho de programa cujo custo é O(3)
     }
...
```

Qual a complexidade assintótica temporal do trecho de programa acima?

12. Considere o trecho de programa abaixo:

```
...
  for(i=n-1; i > 0 ; i--)
     for(j=1; j <= i; j++)
           {
           // trecho de programa cujo custo é O(1)
           }
...
```

Qual a complexidade assintótica temporal do trecho de programa acima?

13. Bill tem um algoritmo buscaEmMatriz para encontrar um elemento x em matriz A n × n. O algoritmo faz iterações sobre as linhas de A e usa o algoritmo de busca sequencial em cada linha até que x seja encontrado ou que todas as linhas tenham sido examinadas. Qual o tempo de execução no pior caso do algoritmo buscaEmMatriz em função de n? O algoritmo é linear O(n)? Justifique.

14. Dados os três métodos abaixo que calculam o valor máximo e mínimo de um arranjo:

// METODO ITERATIVO

```
public static int[] maximoMinimoIterativo(int[] A){
        int[] res = new int[2];
        res[0] = A[0]; // res[0] contem o maximo
        res[1] = A[0]; // res[1] contem o minimo
        for (int i=1;i<A.length;i++){
              if (A[i]>res[0]) res[0] = A[i];
              else {
                    if (A[i]<res[1]){
                          res[1] = A[i];
                    }
              }
        }
        return res;
}
```

// METODO RECURSIVO (INDUÇÃO FRACA)

```
public static int[] maximoMinimoIndFraca(int[] A,int elementos){
        // Criterio de parada: só sobrou um número a ser avaliado
        if (elementos==1){
              int[] res = new int[2];
              res[0] = A[0]; // res[0] contem o maximo
              res[1] = A[0]; // res[1] contem o minimo
              return res;
        }else{
              elementos--;
              int[] res = maximoMinimoIndFraca(A,elementos);
              if (A[elementos]>res[0]){
                    res[0] = A[elementos];
              }else{
                    if (A[elementos]<res[1]){
                          res[1] = A[elementos];
                    }
              }
              return res;
        }
}
```

// METODO RECURSIVO (INDUÇÃO FORTE)

```
public static int[] maximoMinimoIndForte(int[] A,int ini, int fim){
        // Criterio de parada: soh sobrou um numero a ser avaliado
        if (ini==fim){
              int[] res = new int[2];
```

```
                res[0] = A[ini]; // res[0] contem o maximo
                res[1] = A[fim]; // res[1] contem o minimo
                return res;
        }else{
                int meio = (ini+fim)/2;
                int[] res = maximoMinimoIndForte(A,ini,meio);
                int[] res2 = maximoMinimoIndForte(A,meio+1,fim);
                if (res2[0]>res[0]) res[0] = res2[0];
                if (res2[1]<res[1]) res[1] = res2[1];
                return res;
        }
}
```

14a) Escreva a função T(n) (número de vezes que as comparações que estão nas linhas destacadas dos "if"s são executadas para cada um dos métodos, em relação ao tamanho do arranjo de entrada), considerando sempre o pior caso de execução de cada método. Se o método for recursivo, escreva T(n) como uma função de recorrência.

14b) Calcule a complexidade assintótica (θ) para cada uma das funções T(n) acima (no caso de T(n) ser uma equação de recorrência, apresente o cálculo e não apenas o resultado).

15. Utilize o teorema mestre para calcular a complexidade assintótica da seguinte equação de recorrência: T(n) = 4*T(n/2) + n*log(n) ; T(1) = 1

Teorema Mestre

Sejam $a \geq 1$ e $b \geq 2$ constantes, seja $f(n)$ uma função e seja $T(n)$ definida para os inteiros não-negativos pela relação de recorrência

$$T(n) = aT(n/b) + f(n)$$

Então $T(n)$ pode ser limitada assintoticamente da seguinte maneira:

1. Se $f(n) \in O(n^{\log_b a - \epsilon})$ para alguma constante $\epsilon > 0$, então $T(n) \in \Theta(n^{\log_b a})$
2. Se $f(n) \in \Theta(n^{\log_b a})$, então $T(n) \in \Theta(n^{\log_b a} \log n)$
3. Se $f(n) \in \Omega(n^{\log_b a + \epsilon})$, para alguma constante $\epsilon > 0$ e se $af(n/b) \leq cf(n)$, para alguma constante $c < 1$ e para n suficientemente grande, então $T(n) \in \Theta(f(n))$

16. Demonstre que $n \in o(n^2)$

17. Calcule a complexidade assintótica (θ) da função de recorrência T(n) a seguir, NÃO UTILIZE O TEOREMA MESTRE. Exiba o cálculo e não apenas o resultado.

T(1) = 10
T(n) = 3*T(n/3) + n