**Lista01\_Complexidade de Algoritmos**

**1.Questão: Por muitas vezes damos atenção apenas ao pior caso dos algoritmos, explique o por quê.**

**RESPOSTA: A análise do pior caso dos algoritmos é importante para avaliar a viabilidade em cenários com grande entrada de dados. O estudo do pior caso, nos permite analisar se o algoritmo irá executar sua execução baseado nos tamanhos de entrada do domínio do problema e verificar se será eficiente para sua tarefa. A análise do pior caso permite analisar o limite do algoritmo de acordo com o tamanho de entrada, possibilitando entender sua viabilidade mesmo no pior cenário** **possível. Muitas vezes, damos mais atenção apenas ao pior caso dos algoritmos, porque, em geral, estamos interessados em determinar o tempo de execução mais longo para qualquer entrada de tamanho n.**

**2.Questão: Que tipo de crescimento melhor caracteriza cada uma dessas funções? (Constante, Linear, Polinomial, Exponencial)**

1. **(3/2)n - > exponencial**
2. **1- > constante**
3. **(3/2)n- > linear**
4. **2n3 - > cúbica**
5. **2n2 - > quadrática**
6. **3n2 - > quadrática**
7. **1000 - > constante**
8. **3n- > linear**

**3.Questão: Classifique as funções de acordo com o crescimento, do crescimento mais lento (na parte de cima) para o crescimento mais rápido (na parte de baixo)**

**RESPOSTA: C [1] < A [n] < E [n2] < B [n3] < D [(3/2)n] < F [2n]**

**4.Questão: Classifique as funções de acordo com o crescimento, do crescimento mais lento para o mais rápido.**

**D [64] < H [log8 n] < G [log2 n] < A [4n] < F [n\*log6 n] < E [n\*log2 n] < B [8n2] < C [6n3] < I [(8)2n]**

**5.Questão: Seja um algoritmo com complexidade de tempo a(n) = n2 – n + 549 e B um algoritmo com complexidade de tempo b(n) = 49n + 49. Qual algoritmo é melhor?**

**RESPOSTA: O melhor algoritmo é aquele que apresenta menor tempo de execução para o mesmo valor de n. Como O(a) = n2 (apresenta complexidade assintótica quadrática) e O(b) = n (apresenta complexidade assintótica linear), temos que o melhor algoritmo é b(n) = 49n + 49.**

**6.Questão: Considere um algoritmo de força bruta para calcular an, onde n ∈ ℕ. Pergunta-se:**

**a. Qual a complexidade desse algoritmo?**

**RESPOSTA: complexidade exponencial**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Algotitmo em C:**

**/\*\***

**<h1>Questao06</h1>**

**\* Dado um inteiro a, e um natural n, calcula a elevado a n.**

**\* <p>**

**\* <b>Nota:<b> Leia atentamente a documentação deste programa**

**\* para desfrutar dos recursos oferecidos pelo autor.**

**\***

**\* @author James Anderson**

**\* @version 1.0**

**\* @since 09/12/2022**

**\*/**

**#include <string.h>**

**#include <math.h>**

**#include <stdlib.h>**

**#include <stdio.h>**

**int main ()**

**{**

**//Declaração das variáveis.**

**int base, expoente;**

**double potencia = 1;**

**//Solicitação e leitura da base.**

**printf ("Digite a base: "); c1**

**scanf ("%d", &base); c2**

**//Solicitação e leitura do expoente.**

**printf ("Digite o expoente: "); c3**

**scanf ("%d", &expoente); c4**

**//Implementação do laço para o cálculo da potência.**

**for (int i = 1; i <= expoente; i++) { c5 \* (expoente + 1)**

**//Calcula a potência.**

**potencia \*= base; c6 \* expoente**

**}**

**printf ("O valor da potencia eh dado por: %.4f", potencia); c7**

**return 0;**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**b. Construa um algoritmo iterativo que calcule o valor em tempo O(log n). Para construção do algoritmo, você pode se basear na seguinte fórmula de exponenciação:**

**/\*\***

**<h1>Questao07</h1>**

**\* Dados tres números naturais, calcular o tempo O(log n).**

**\* <p>**

**\* <b>Nota:<b> Leia atentamente a documentação deste programa**

**\* para desfrutar dos recursos oferecidos pelo autor.**

**\***

**\* @author James Anderson**

**\* @version 1.0**

**\* @since 13/12/2022**

**\*/**

**#include <string.h>**

**#include <math.h>**

**#include <stdlib.h>**

**#include <stdio.h>**

**int main ()**

**{**

**//Declaração das variáveis.**

**int x, y, n;**

**double tempo;**

**//Solicitação e leitura de x.**

**printf ("Digite x: ");**

**scanf ("%d", &x);**

**//Solicitação e leitura de y.**

**printf ("Digite y: ");**

**scanf ("%d", &y);**

**//Solicitação e leitura de n.**

**printf ("Digite n: ");**

**scanf ("%d", &n);**

**//Cálculo do tempo.**

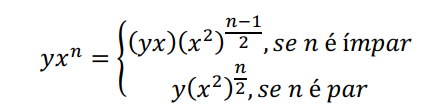
**if(n%2 == 0){**

**tempo = x\*y\*pow((pow(x,2)), ((n-1)/2));**

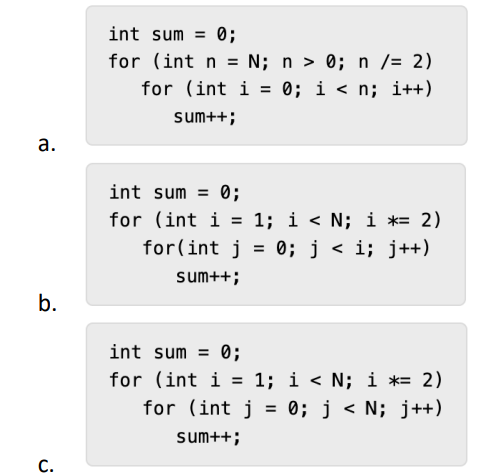
**} else tempo = y\*pow((pow(x,2)), (n/2));**

**printf ("O valor da potencia eh dado por: %.4f", tempo);**

**return 0;**



**7.Questão: Estime a complexidade assintótica de cada um dos algoritmos abaixo.**



**RESPOSTA:**

**7A**

**Complexidade linear: (N + N/2 + N/4 + N/8 + .....)**

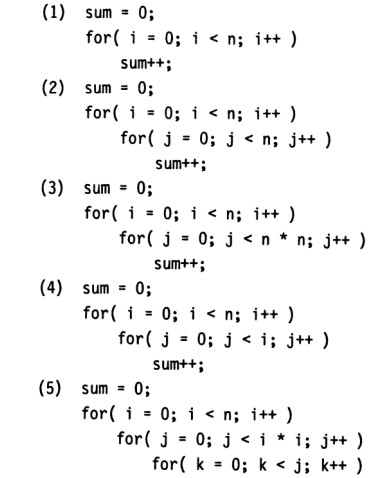
**7B**

**Complexidade linear: (1 + 2 + 4 + 8 + ......)**

**7C**

**Complexidade n\*log n: (o primeiro for executa log n e o segundo n vezes)**

**8.Questão: Diga a complexidade assintótica dos seguintes algoritmos.**



**RESPOSTA:**

**8.1**

**Linha 1: c1**

**Linha 2: c2\*n**

**Linha 3: c3\* (n – 1)**

**Complexidade: Linear, O(n) = n**

**8.2**

**Linha 1: c1**

**Linha 2: c2\*n**

**Linha 3: c3\*n2**

**Linha 4: c4\*(n2 – 1)**

**Complexidade: Quadrática, O(n) = n2**

**8.3**

**Linha 1: c1**

**Linha 2: c2\*n**

**Linha 3: c3\*n3**

**Linha 4: c4\*(n3 – 1)**

**Complexidade: Cúbica, O(n) = n3**

**8.4**

**Linha 1: c1**

**Linha 2: c2\*n**

**Linha 3: c3\*n2**

**Linha 4: c4\* (n2 - 1)**

**Complexidade: Quadrática, O(n) = n2**

**8.5**

**Linha 1: c1**

**Linha 2: c2\*n**

**Linha 3: c3\*n3**

**Linha 4: c4\* (n5 – 1)**

**Complexidade: Polinomial, O(n) = n5**