

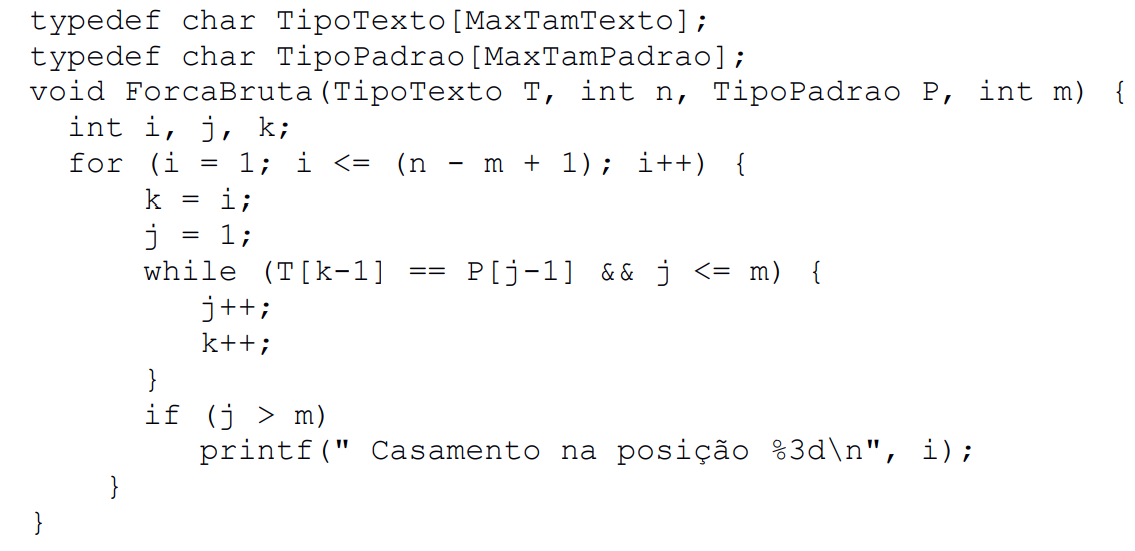
**UFMG – Engenharia de Sistemas**

**Lista de Exercicio 1**

**Análise de Complexidade de Algoritmos**

|  |  |  |
| --- | --- | --- |
| **DISCIPLINA:** | ESTRUTURA DE DADOS | |
| **PROFESSOR:** | [Carlos Henrique de Carvalho Teixeira](https://virtual.ufmg.br/20192/user/view.php?id=23049&course=9470) | |
| **ALUNO:** | Antonio Carlos da Anunciação | 2018019443 |

**1.** O Casamento de Padrões é um problema clássico em ciência da computação e é aplicado em áreas diversas como pesquisa genética, editoração de textos, buscas na internet, etc. Basicamente, ele consiste em encontrar as ocorrências de um padrão P de tamanho m em um texto T de tamanho n. Por exemplo, no texto T = “PROVA DE AEDSII” o padrão P = “OVA” é encontrado na posição 3 enquanto o padrão P = “OVO” não é encontrado. O algoritmo mais simples para o casamento de padrões é o algoritmo da “Força Bruta”, mostrado abaixo. Analise esse algoritmo e responda: Qual é a função de complexidade do número de comparações de caracteres efetuadas no melhor caso e no pior caso. Dê exemplos de entradas que levam a esses dois casos. Explique sua resposta!



*Para o melhor caso desse algoritmo o laço “while” vai iterar apenas “m” vezes, significando que encontrou o padrão nas “m” letras iniciais da palavra de entrada, o laço “for”sempre ira iterar (n-m+1), dessa maneira no melhor caso teremos:*

*f(n,m) = n – m + m*

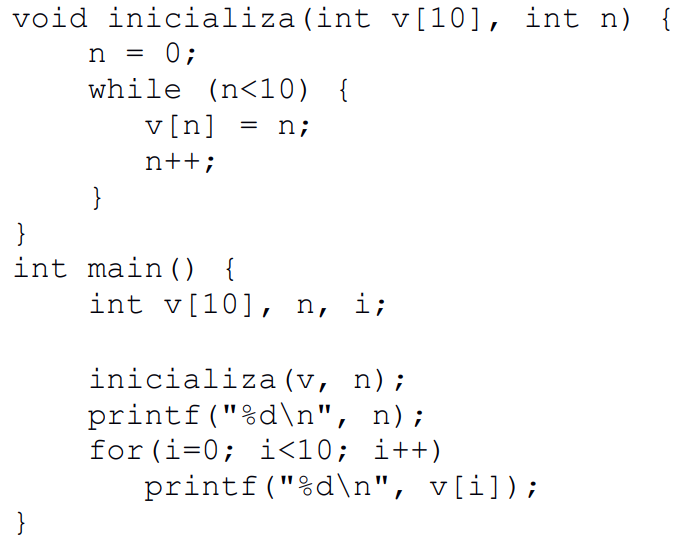
*f(n,m) = n*

*Para o pior caso desse algoritmo o laço “while” vai iterar até k = n-1, significando que encontrou o padrão nas “n-m” letras finais da palavra de entrada, o laço “for”sempre ira iterar (n-m+1), dessa maneira no melhor caso teremos:*

*f(n,m) = n – m + n – 1*

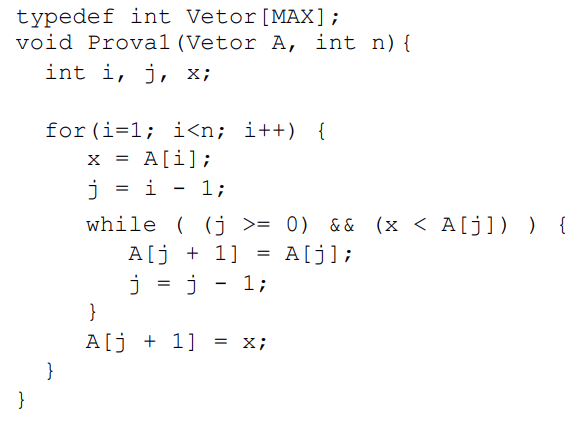
*f(n,m) = 2n – m – 1*

**2.** O que será impresso pelo programa abaixo. Explique a sua resposta.



*Uma sequencia de número de 0 a 10, esse programa recebe duas entradas vazias, um vetoe de tamanho 10 e um inteiro n sem atribuição, internamente é atribuido um valor inicial a esse inteiro e a partir esse inteiro em cada passo é guardado na posição “n” do vetor de entrada e incrementado 1, até que o valor final de n seja 9.*

**3.** Considere o algoritmo abaixo. O que ele faz? Qual é a função de complexidade do número de comparações de elementos do vetor no melhor caso e no pior caso? Que configuração do vetor de entrada A leva a essas duas situações? Explique / Demonstre como você chegou a esses resultados. (Dica: analise para cada valor de i quantas vezes o while é executado no melhor e no pior caso, e monte um somatório…).



*Esse algoritmo ordena um vetor de forma crescente. Sua Função de Complexidade O(n) será:*

*Para o pior caso, será quando a condição* x < A [ j ] *for verdadeira para todo* J >= 0*, j tera seu maior valor quando* i = n – 1*, está situação teremos quando o vetor estiver ordenado de forma decrescente.*

f(n) = n - 1 + (n – 1) – 1; f(n) = 2n – 3.

*O melhor caso será quando a condição do “while” for falsa para* x < A [ j ], *situação que nosso vetor está ordena de forma crescente.*

f(n) = n - 1

**4.** Sejam f(n), g(n) duas funções assintóticas positivas e a e b. Prove que as afirmativas abaixo são verdadeiras ou falsas, usando para isso as definições das notações assintóticas ou contraexemplos.

1. 2n+1 = O(2n)

2n+1 = 2.2n = c. 2n = O(2n), Verdadeiro.

1. 22n = O(2n)

22n= c.2n

22n= c, Logo é falsa porque c deve ser uma constante.

1. f(n) + g(n) = O(Max(f(n), g(n))

Verdadeiro, porque em um algoritmos com varias funções o complexidade do todo é a complexidade da função de maior complexidade.

**5.** Implemente uma função recursiva para computar o valor de 2n.

int potencia\_de\_2(int n) {

if( n == 0) return 1;

else return 2\*potencia\_de\_2(n - 1);

}

**6.** Resolva a seguinte questão sobre recursividade:

1. Escreva uma **função recursiva** int Palindromo(int esq, int dir, char palavra[ ]) que testa se uma determinada palavra é um palíndromo e retorna 1 em caso positivo e 0 em caso negativo. Um palíndromo é uma palavra que é lida da mesma forma da esquerda para direita ou da direita para esquerda (ex. ovo, arara). A palavra é passada para o função através de um vetor de caracteres limitada pelos os índices esq e dir, por exemplo:

Palindromo(0,4,"arara")

int Palindromo(int esq, int dir, char palavra[100]) {

if(palavra[esq] == palavra[dir]) {

if(esq >= dir) return 1;

else return Palindromo(esq+1,dir-1, palavra);

}

else return 0;

};

1. Calcule qual é a **função de complexidade** para o número de comparações de caracteres da sua função no melhor caso e no pior caso. Para isso, **determine e resolva** a equação de recorrência dessa função recursiva. Qual é a **ordem de complexidade** de sua função?

Para nossa função recursiva teremos teremos nossa equação de recorrencia da seguinte forma:

T(n) = T(n/2) + 1, n > 0

T(n) = 1, n ≤ 0

Por indução:

T(n) = n/2 + 1, ou seja: **O(n) = n**

O melhor caso sera quando a palavra não for um Palindromo, nos dando **O(n) = 1**, o pior caso é um palindromo de tamanho máximo do vetor, com: **O(n) = n.**

1. Qual seria a complexidade de uma implementação não recursiva dessa mesma função? Qual das duas implementações vocês escolheria? Justifique.

int Palindromo\_nao\_recursivo(int esq, int dir, char palavra[100]) {

while(palavra[esq] == palavra[dir]) {

esq+=1;

dir-=1;

if(esq >= dir) return 1;

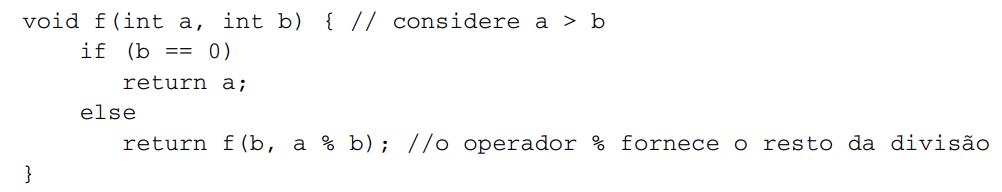
}

return 0;

};

Ambas tem a mesma complexidade no tempo, dessa maneira eu escolheria a de menor complexidade na memoria, evitando assim trabalhar com a função recursiva.

**7.** O que faz a função abaixo? Explique o seu funcionamento.



*Está função calcula o máximo divisor comum entre dois valores de entrada, na primeira iteração recursiva ela verifica se o menor valor de entrada é divisivel pelo maior valor de entrada, verificando se o resto da divisão é 0, caso não for uma disisão exata o algoritmo itera novamente entra o menor valor das entradas originais e o resto da divisão entre os valores originais, até que seja retornado um resto nulo.*

**8.** Vários algoritmos em computação usam a técnica de “Dividir para Conquistar”: basicamente eles fazem alguma operação sobre todos os dados, e depois dividem o problema em sub-problemas menores, repetindo a operação. Uma equação de recorrência típica para esse tipo de algoritmo é mostrada abaixo. Resolva essa equação de recorrência usando o Teorema Mestre.

T(n) = 2T(n/2) + n;

, **O(n)**

T(1) = 1;

O(n) = n, com n = 1, então O(1) = 1