# 

**Curso:** Ciência de Computação **Período: 3**º **Turno:** Diurno

**Disciplina:** Estrutura de Dados **Professor:** Fermín Alfredo Tang Montané

**Atividade:** Avaliação AM1 **Data:** 01/04/2021

**Aluno:** João Vítor Fernandes Dias **Matrícula:** 00119110377

## [2,0 Pontos] defina os conceitos de Estrutura de dados e Tipo Abstrato de Dado. Ilustre com um exemplo em cada caso.

**R:**

* Estrutura de dados:
  + Definição: A estrutura de dados é o agrupamento de diversos dados, podendo eles ser atômicos ou até mesmo outra estrutura de dados, que estão ligados por algum sentido geral que os represente.
  + Exemplo 1: Endereço (pode conter: país, estado, cidade, rua, número da residência, CEP, etc.).
  + Exemplo 2: Data (pode conter (dependendo da precisão): ano, mês, dia, hora, minuto, segundo, etc.).
* Tipo Abstrato de Dado (TAD):
  + Definição: O Tipo Abstrato de Dado tem por definição a abstração e encapsulamento das funções e estruturas de dados que certa estrutura pode vir a precisar realizar. As funções por sua vez devem estar ocultas do usuário, permitindo assim o acesso indireto às funcionalidades que foram implementadas.
  + Exemplo: Caso queiramos um código que represente um cilindro e que possa fazer diversas operações relacionadas a ele, poderíamos fazer um programa específico para desempenhar esse requisito, porém seria exclusivo para esse caso. Podemos então generalizar e abstrair a ideia inicial proposta e criar um TAD que tenha funções gerais como criar o “tipo cilindro”, criar o ponteiro referente a esse tipo, inserir os dados nesse ponteiro, retornar os valores inseridos nele, retornar o valor da área, retornar o valor da altura, liberar o espaço alocado. Tendo todas essas funcionalidades, o usuário não precisará se preocupar em como elas foram implementadas, mas sim o que fazer com elas.

## [2,0 Pontos] dado que o tamanho da entrada de um algoritmo A é , e considerando que o número de operações realizada pelo mesmo é .

1. [1,0 ponto] complete os dados da tabela para diferentes tamanhos da entrada .

|  |  |
| --- | --- |
| *n* | *f(n³+n)* |
| *20* | *8020* |
| *100* | *1000100* |
| *500* | *125000500* |

**R:**

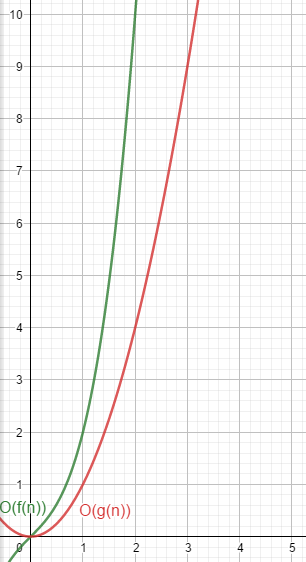
1. [1,0 ponto] Dado um algoritmo B, cujo número de operações é . Considerando o conceito de complexidade assintótica O ( ), podemos afirmar que o algoritmo B é sempre mais rápido que A? Justifique.

**R:**

Considerando o conceito de complexidade assintótica (notação big O) sobre a equação primeiro deveríamos remover os coeficientes, porém, como não há coeficiente, nenhum será removido. Agora analisaremos apenas o monômio de maior grau, que é , sendo assim .

Utilizando o mesmo conceito sobre a equação primeiro devemos remover os coeficientes, assim tendo , agora analisaremos apenas o monômio de maior ordem sendo ele o , sendo assim .

Então, se compararmos a equação com , teremos o seguinte comportamento:



Esse gráfico representa o grau de complexidade de cada algoritmo. Sendo o algoritmo A representado pela função e o algoritmo B representado pela função .

Temos que quanto menor for a complexidade, mais rápido será o algoritmo, assim podemos concluir que para números maiores que 0 o algoritmo B tende a ter menos complexidade, consequentemente, sendo mais rápido do que o algoritmo A.

## [2,0 Pontos] dado que a função de eficiência de um algoritmo é , se um passo neste algoritmo leva 1 nanosegundo quanto tempo leva o algoritmo para processar uma entrada de tamanho 10.000.

|  |  |  |
| --- | --- | --- |
| f (n) = | 3n\*log(n) | |
| f (10000) = | 3\*10000\*log2(10000) | |
| f (10000) = | 30000\*log2(10000) | |
| f (10000) = | 30000\*13,28771... | |
| f (10000) = | 398631,3714 | |
| f (10000) ≅ | | 398631 |

1. Calcule o número de operações.

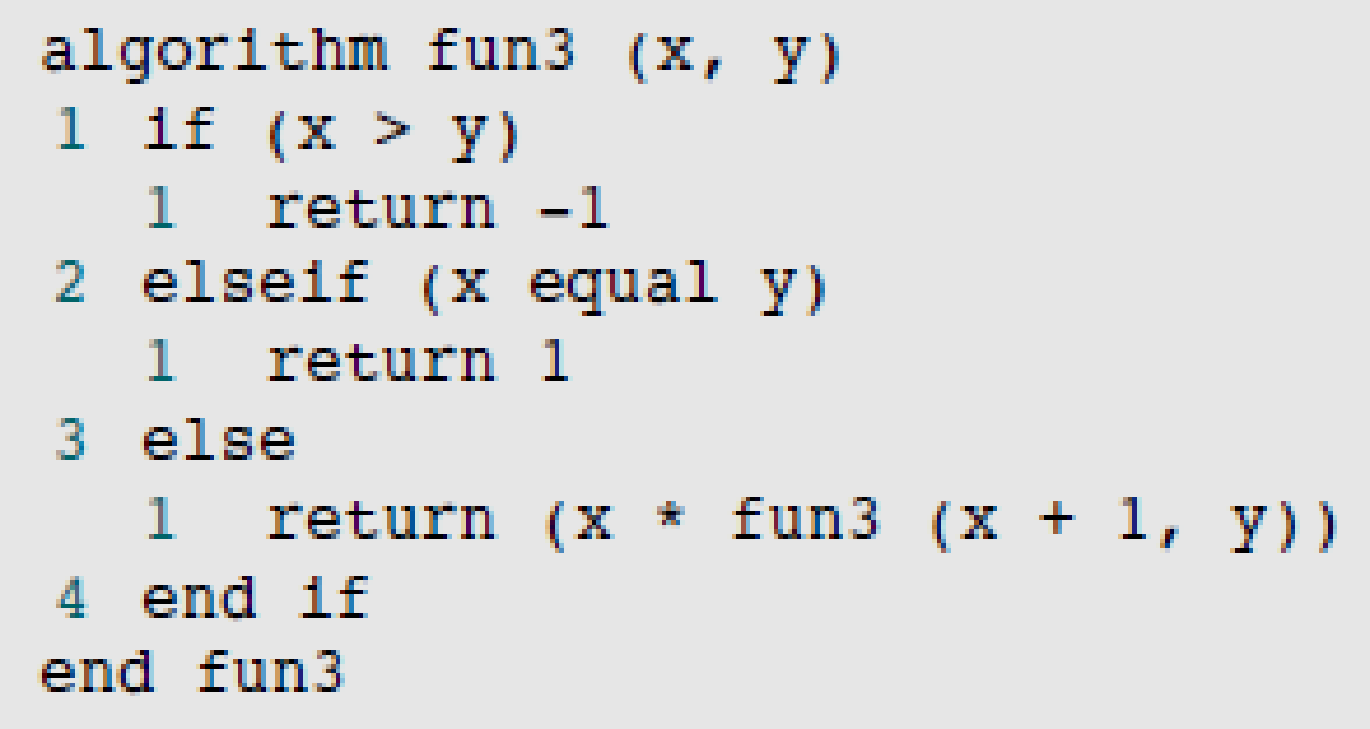
**R:**

1. Calcule o tempo em unidades adequadas.

|  |  |  |
| --- | --- | --- |
| passos | 398631 | passos |
| ns=passos | 398631 | ns |
| μs=ns\*10-3 | 398,631 | μs |
| ms=ns\*10-6 | 0,398631 | ms |
| s=ns\*10-9 | 0,000398631 | s |

**R:**

## [2,0 Pontos] considere o seguinte algoritmo para a função recursiva

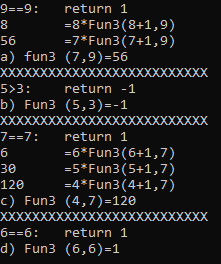


Qual seria o valor retornado, no caso em que chamada a fosse:

1. fun3 (7, 9);
2. fun3 (5, 3);
3. fun3 (4, 7);
4. fun3 (6, 6).

Em cada caso mostre a sequência de chamadas recursivas.

**R:**



## [2,0 Pontos] faça um algoritmo para uma função recursiva que calcule o valor da série descrita para um valor maior do que zero, a ser fornecido como parâmetro:

**R:**

#include <stdio.h>

int filterInput (){

int num;

do{

printf("\nType a positive Integer:\t");

scanf("%d", &num);

} while (num<0);

return num;

}

float serie (int n){

if (n>0){

return ((1.0/n+n) + serie(n-1));

} else {

return 0;

}

}

int main(){

printf("%.3f", serie(filterInput()));

}