



ALGORITMOS E COMPLEXIDADE (ARA0174)

AULA 10

DATA: 05/05/2022



Aula 10- ALGORITMOS E COMPLEXIDADE

APRESENTAÇÃO DO GABARITO PROVA AV1



Estácio

Aula 10 - ALGORITMOS E COMPLEXIDADE

APRESENTAÇÃO DO GABARITO PROVA AV1

Questão 01) Analise as afirmativas abaixo a respeito de um algoritmo recursivo:

- I. Deve conter pelo menos uma estrutura de repetição FOR.
- II. Deve conter pelo menos uma estrutura de repetição WHILE.
- III. Deve conter pelo menos uma estrutura de seleção.

Assinale a alternativa correta. (0,5 pt)

- () Todas as afirmativas estão corretas.
- (**X**) Somente a afirmativa III está correta.
- () Somente as afirmativas I e II estão corretas.
- () Somente a afirmativa I está correta.
- () Somente as afirmativas II e III estão corretas.

Aula 10 - ALGORITMOS E COMPLEXIDADE

APRESENTAÇÃO DO GABARITO PROVA AV1

Questão 02) Um algoritmo tem complexidade $O(3m^3+2mn^2+n^2+10m+m^2)$. Mostre a maneira simplificada de representar a complexidade desse algoritmo (1,0 pt)

Mostre o passo a passo dessa simplificação.

$$O(3m^3+2mn^2+n^2+10m+m^2)$$

$$O(\cancel{3m^3}+\cancel{2mn^2}+n^2+\cancel{10m}+m^2)$$

$$O(m^3+mn^2+n^2+m+m^2)$$

Resposta:

$$O(m^3+mn^2)$$

Aula 10 - ALGORITMOS E COMPLEXIDADE

APRESENTAÇÃO DO GABARITO PROVA AV1

Questão 03) Abaixo temos uma função chamada *mergeSort*, pertencente ao algoritmo de ordenação mergesort (ordenação por mistura), e o mesmo apresenta **ERRO(S)** em alguma(s) linha(s). Reescreva a(s) linha(s) que precisa(m) ser corrigida(s) para que esta função funcione corretamente. (1,0 pt)

```
void mergeSort (int VEC[], int inicio, int fim) {  
    int meio;  
    if (inicio < fim) {  
        meio = (inicio + fim);  
        mergeSort(VEC, inicio, meio);  
        mergeSort(VEC, meio+1, fim);  
        merge(VEC, inicio, meio, meio-1, fim);  
    }  
}
```

Reescreva AQUI!

$\text{meio} = (\text{inicio} + \text{fim}) / 2;$

$\text{merge}(\text{VEC}, \text{inicio}, \text{meio}, \text{meio}+1, \text{fim});$

Aula 10 - ALGORITMOS E COMPLEXIDADE

APRESENTAÇÃO DO GABARITO PROVA AV1

Questão 03) Abaixo temos uma função chamada *mergeSort*, pertencente ao algoritmo de ordenação mergesort (ordenação por mistura), e o mesmo apresenta ERRO(S) em alguma(s) linha(s). Reescreva a(s) linha(s) que precisa(m) ser corrigida(s) para que esta função funcione corretamente. (1,0 pt)

```
void mergeSort (int VEC[], int inicio, int fim) {  
    int meio;  
    if (inicio < fim) {  
        meio = (inicio + fim);  
        mergeSort(VEC, inicio, meio);  
        mergeSort(VEC, meio+1, fim);  
        merge(VEC, inicio, meio, meio+1, fim);  
    }  
}
```

Código mergesort trabalhado

```
void mergeSort (int VEC[], int inicio, int fim) {  
    // O primeiro passo é dividir o vetor em duas partes.  
    // (meio a meio) então crio a variável "meio" para me  
    // auxiliar para dividir o vetor.  
    int meio;  
  
    // Se inicio for menor que fim, pegando "por exemplo um vetor com 8 posições",  
    // seria 0 < 7.  
    if (inicio < fim) {  
        // Aqui divido o meu vetor em 2 partes.  
        // Mais se o vetor tiver o número de posições ímpar? Po exemplo: 8 posições  
        // Utilizo a função floor (Arrendando o valor inteiro),  
        // ou seja 7/2 = 3,5, com o floor vai ficar 3.  
  
        // Como estamos utilizando a questão da recursividade  
        // Vai voltando a função e passando pela análise if(inicio < fim)  
        // dividindo o vetor até chegar a 1.  
  
        meio = floor((inicio + fim)/2);  
  
        // Faz a ordenação da primeira parte ou primeira metade.  
        // Por exemplo: vetor com 8 posições que vai de 0 a 7.  
        // inicio = 0 e meio = 3  
        mergeSort(VEC, inicio, meio);  
  
        // Faz a ordenação da segunda parte ou segunda metade.  
        // Por exemplo: vetor com 8 posições que vai de 0 a 7.  
        // meio+1 = 3+1 = 4 e fim=7.  
        mergeSort(VEC, meio+1, fim);  
  
        // Realizará a ordenação das duas partes.  
        merge(VEC, inicio, meio, meio+1, fim);  
    }  
}
```

Aula 10 - ALGORITMOS E COMPLEXIDADE

APRESENTAÇÃO DO GABARITO PROVA AV1

Questão 04) Sobre o comportamento assintótico do algoritmo de ordenação mergesort, marque a alternativa que apresenta, corretamente, sua complexidade. (0,5 pt)

() $O(n^2)$ () $O(n^3)$ () $O(2^n)$ () $O(\log n)$ (**X**) $O(n \log n)$

Aula 10 - ALGORITMOS E COMPLEXIDADE

APRESENTAÇÃO DO GABARITO PROVA AV1

Questão 05) Correlacione as colunas apropriadamente, não havendo repetição de itens: (1,0 pt)

(A) FUNÇÃO

(**D**) Ordenação por intercalação, separando e juntando os elementos.

(B) DIVIDIR PARA CONQUISTAR

(**B**) O problema é separado em pequenas partes a serem solucionadas.

(C) RECURSIVIDADE

(**A**) Bloco de código a ser executado quando chamado retornando um valor.

(D) MERGESORT

(**C**) O problema é dividido em dois casos (duas partes): Trivial (base) e geral.

Aula 10 - ALGORITMOS E COMPLEXIDADE

APRESENTAÇÃO DO GABARITO PROVA AV1

Questão 06) Considere os seguintes algoritmos e suas complexidades na notação O:

- Algoritmo A: $O(\log n)$ - Algoritmo B: $O(n^2)$ - Algoritmo C: $O(n \cdot \log n)$

Considerando-se o pior caso de execução destes algoritmos, é correto afirmar que o algoritmo:
(1,0 pt)

- ☐ A é o menos eficiente.
- ☐ C é o menos eficiente.
- ☐ A não é o mais eficiente nem o menos eficiente.
- ☒ B é o menos eficiente.
- ☐ C é o mais eficiente.

Aula 10 - ALGORITMOS E COMPLEXIDADE

APRESENTAÇÃO DO GABARITO PROVA AV1

Questão 06) Considere os seguintes algoritmos e suas complexidades na notação O:

- Algoritmo A: $O(\log n)$

- Algoritmo B: $O(n^2)$

- Algoritmo C: $O(n \cdot \log n)$

Considerando-se o pior caso de execução destes algoritmos, é correto afirmar que o algoritmo: (1,0 pt)

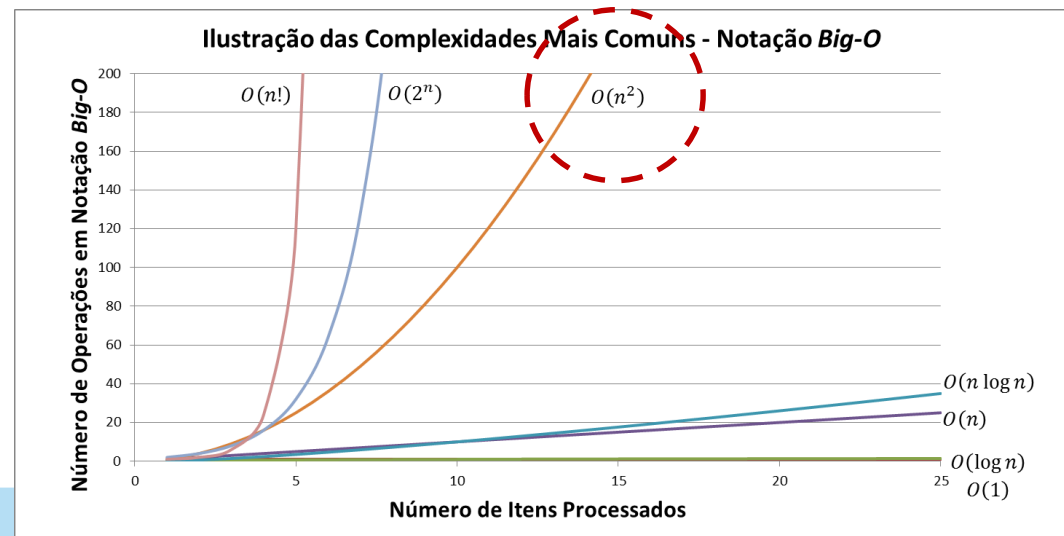
() A é o menos eficiente.

() C é o menos eficiente.

() A não é o mais eficiente nem o menos eficiente.

(**X**) B é o menos eficiente.

() C é o mais eficiente.



Aula 10 - ALGORITMOS E COMPLEXIDADE

APRESENTAÇÃO DO GABARITO PROVA AV1

Questão 07) Qual a finalidade da análise de um algoritmo?. (Marque a ÚNICA alternativa correta) (0,5 pt)

- () Digitação de menos código e consequentemente, menos trabalho.
- () Obter uma saída de dados independente da entrada de dados.
- (**X**) Correção e melhor desempenho do algoritmo.
- () Poder utilizar qualquer linguagem de programação.
- () Uso exclusivo da recursão no algoritmo.

Aula 10 - ALGORITMOS E COMPLEXIDADE

APRESENTAÇÃO DO GABARITO PROVA AV1

Questão 08) Considere o código abaixo no qual possui uma função recursiva:


```
#include<iostream>
#include<conio.h>

using namespace std;

int retorna(int a);

int main() {
    cout << "\n\n Retorna: " << retorna(4);
    getch();
}

int retorna(int a) {
    if (a == 1) {
        return -a;
    } else {
        return 2*a + retorna(a - 1);
    }
}
```



Responda:

Qual o valor retornado pela função? (1,0 pt)
(Mostrar o passo a passo trabalhando com o código acima)

$\text{return } 2*4 + \overset{3}{\text{retorna}(4 - 1)};$
 $\text{return } 2*4 + \overset{2}{2*3} + \text{retorna}(3 - 1);$
 $\text{return } 2*4 + 2*3 + \overset{1}{2*2} + \text{retorna}(2 - 1);$
 $\text{return } 2*4 + 2*3 + 2*2 + (-1);$
 $\text{return } 8 + 6 + 4 + (-1); \rightarrow \text{return } 17;$

Aula 10 - ALGORITMOS E COMPLEXIDADE

APRESENTAÇÃO DO GABARITO PROVA AV1

Questão 09) Qual a complexidade do algoritmo abaixo? (0,5 pt)
(Complexidade da função size é $O(1)$)

```
#include<iostream> Faça as marcações/destaque as complexidades
#include<vector>
using namespace std;

bool exemplo4(vector<int> A, vector<int> B) {
    // size() = Função que retorna o número de elementos do vetor.
    int TAM1 = A.size();
    int TAM2 = B.size();
    for(int i=0; i<TAM1; i++){
        for(int j=0; j<TAM2; j++) {
            if( A[i] == A[j]) {
                return true;
            }
        }
    }

    return false;
}

int main() {
    std::vector<int> A;
    std::vector<int> B;
    A.push_back(5); A.push_back(15); A.push_back(25);
    B.push_back(10); B.push_back(20); B.push_back(30);
    cout << exemplo4(A, B); }
```

Mostre a análise passo a passo.

Aula 10 - ALGORITMOS E COMPLEXIDADE

APRESENTAÇÃO DO GABARITO PROVA AV1

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

PASSOS PARA O CÁLCULO:

- 1 - LEVAR EM CONSIDERAÇÃO APENAS AS REPETIÇÕES DO CÓDIGO.
- 2 - VERIFICAR A COMPLEXIDADE DAS FUNÇÕES/MÉTODOS PRÓPRIOS DA LINGUAGEM (SE UTILIZADO).
- 3 - IGNORAR AS CONSTANTES E UTILIZAR O TERMO DE MAIOR GRAU.

Questão 09) Qual a complexidade do algoritmo abaixo? (0,5 pt)

(Complexidade da função size é $O(1)$)

```
#include<iostream>
#include<vector>
using namespace std;

bool exemplo4(vector<int> A, vector<int> B) {
    // size() = Função que retorna o número de elementos do vetor.
    int TAM1 = A.size(); ← O(1)
    int TAM2 = B.size(); ← O(1)
    for(int i=0; i<TAM1; i++){ ← O(n)
        for(int j=0; j<TAM2; j++){ ← O(m)
            if( A[i] == A[j]) { ← O(1)
                return true; ← O(1)
            }
        }
    }

    return false; ← O(1)
}

int main() {
    std::vector<int> A;
    std::vector<int> B;
    A.push_back(5); A.push_back(15); A.push_back(25);
    B.push_back(10); B.push_back(20); B.push_back(30);
    cout << exemplo4(A, B); }
```

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 4:

AGORA TEMOS QUE ESCREVER EM UMA ÚNICA COMPLEXIDADE.

LOGO:

$O(n)*O(m)$.

3º PASSO: IGNORAR AS CONSTANTES E UTILIZAR O TERMO DE MAIOR GRAU.

Esse é o termo final: $O(n)*O(m)$.

Aula 10 - ALGORITMOS E COMPLEXIDADE

```
#include<iostream>
#include<locale.h>
using namespace std;

int dec2hex(int dec);

int main() {
    setlocale(LC_ALL, "portuguese");
    int dec;
    cout << "Digite um no. inteiro: ";
    cin >> dec;
    cout << "\n\n(Leia da direita para a esquerda)";
    cout << "\n Resultado: ";
    dec2hex(dec);
    return 0;
}

int dec2hex(int dec) {
    int n, resto;
    static int vetor[10], i=0, k;

    if (dec/16!=0) {
        n = dec/16;
        resto = dec%16;

        vetor[i]=resto;
        cout << resto << " ";
        i=i+1;

        return dec2hex(n);
    }
```

```
    } else {
        resto = dec%16;
        vetor[i]=resto;
        cout << resto << " ";
    }
    cout << "\n\n Exibindo na ordem: ";
    for (k=i; k>=0; k--) {
        if ( vetor[k] >= 10 ) {
            switch( vetor[k] ) {
                case 10 : cout << "A " ; break;
                case 11 : cout << "B " ; break;
                case 12 : cout << "C " ; break;
                case 13 : cout << "D " ; break;
                case 14 : cout << "E " ; break;
                case 15 : cout << "F " ; break;
            }
            cout << vetor[k] << " ";
        }
    }
}
```

Aula 10 - ALGORITMOS E COMPLEXIDADE

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

REFINANDO A ANÁLISE (ALGUMAS PARTICULARIDADES):

CASO 1:

Tem-se dois algoritmos cuja quantidade de operações está apresentado abaixo:

Algoritmo A: $f(n) = n^2$

Algoritmo B: $f(n) = n + 1.000$

Não pertencem à mesma ordem, pois o primeiro tem comportamento quadrático, e o segundo, comportamento linear.

Aula 10 - ALGORITMOS E COMPLEXIDADE

REFINANDO A ANÁLISE (ALGUMAS PARTICULARIDADES):

CASO 1:

Se $n = 10$, a quantidade de operações executadas por cada algoritmo é:

Algoritmo A: $f(n) = n^2$

$f(10) = 10^2 = 100$ operações

Algoritmo B: $f(n) = n + 1.000$

$f(10) = 10 + 1.000 = 1.010$ operações

Conclui-se que, o algoritmo A é 10 vezes mais rápido do que o algoritmo B.

Aula 10 - ALGORITMOS E COMPLEXIDADE

REFINANDO A ANÁLISE (ALGUMAS PARTICULARIDADES):

CASO 2:

Se $n = 1000$, a quantidade de operações executadas por cada algoritmo é:

Algoritmo A: $f(n) = n^2$

$f(1000) = 1000^2 = 1.000.000$ operações

Algoritmo B: $f(n) = n + 1.000$

$f(1000) = 1000 + 1.000 = 2.000$ operações

Conclui-se que, o algoritmo A é 500 vezes mais lento do que o algoritmo B.