



ALGORITMOS E COMPLEXIDADE (ARA0174)

AULA 4

DATA: 10/03/2022



Aula 4- ALGORITMOS E COMPLEXIDADE

**RELEMBRANDO PONTUALMENTE ALGUMAS
QUESTÕES TRABALHADAS NA AULA
PASSADA...**



Estácio

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Exercício 01

Qual a importância de estudarmos a complexidade dos algoritmos?

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Exercício 01 – Resposta

Qual a importância de estudarmos a complexidade dos algoritmos?

Tem como objetivo analisar o consumo de recursos necessários para a execução dos algoritmos e, assim, buscar formas de otimização do uso desses recursos

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Exercício 02

Conceitue a Notação O e o seu significado para a análise da complexidade de um algoritmo.

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Exercício 02– Resposta

Conceitue a Notação O e o seu significado para a análise da complexidade de um algoritmo.

Trata-se da notação que define o maior tempo de execução de um algoritmo sobre todas as entradas de tamanho n . Significa que ao avaliarmos a complexidade de algoritmo, devemos fazê-la esperando o cenário mais pessimista de sua execução

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Exercício 03

Defina a análise da complexidade de algoritmos do melhor caso, a do caso médio e a do pior caso.

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Exercício 03 – Resposta

Defina a análise da complexidade de algoritmos do melhor caso, a do caso médio e a do pior caso.

Melhor caso: Previsão do melhor cenário, executando o mínimo de instruções possíveis, tendo como reflexo o uso otimizado dos recursos computacionais,

Médio caso: Previsão de um consumo mediano de recursos computacionais

Pior caso: Previsão do pior cenário para a execução do algoritmo, prevendo que ele execute o máximo de instruções possíveis

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Exercício 04

Por que a análise da complexidade é feita sob a ótica do pior caso?

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Exercício 04 - Resposta

Por que a análise da complexidade é feita sob a ótica do pior caso?

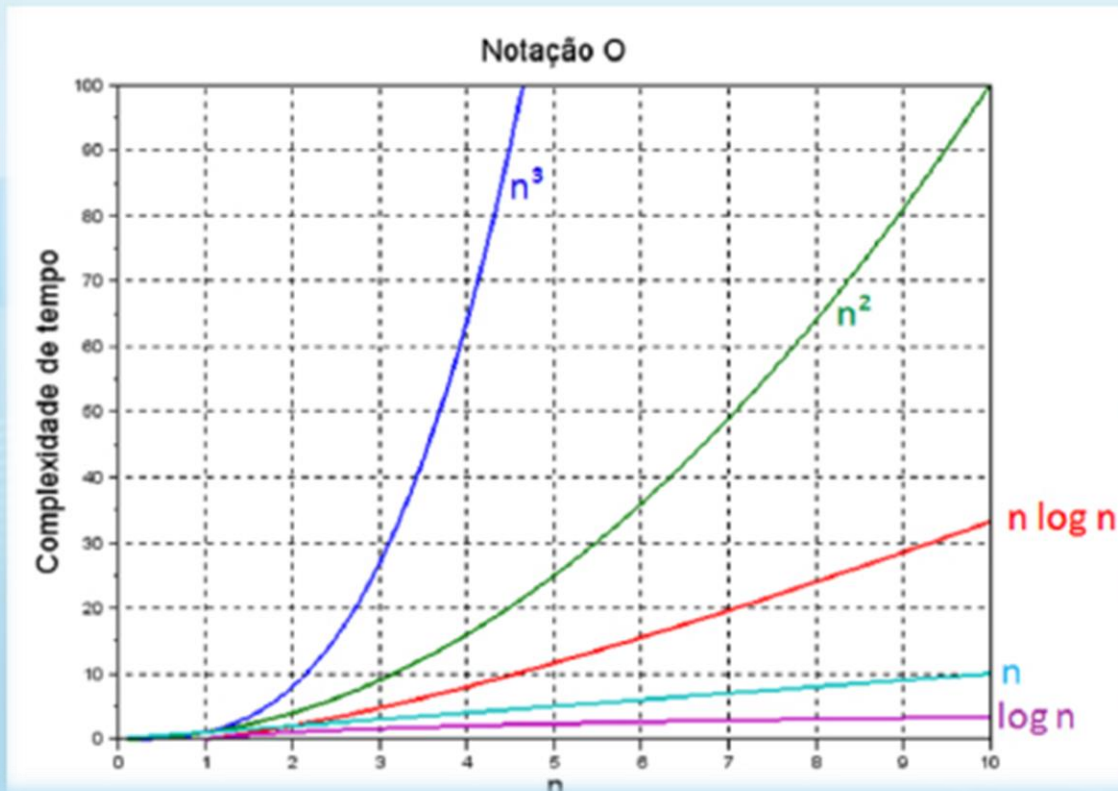
É importante prever o cenário de execução mais pessimista do algoritmo, para que não geremos falsas expectativas quanto a eficiência do programa.

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Exercício 05

Considerando o gráfico abaixo, podemos perceber que ele exibe uma comparação entre as grandezas da Notação O para alguns tipos de algoritmos. Pesquise e comente a eficiência de cada tipo de algoritmo, no tratamento de entradas para grandes valores de n .

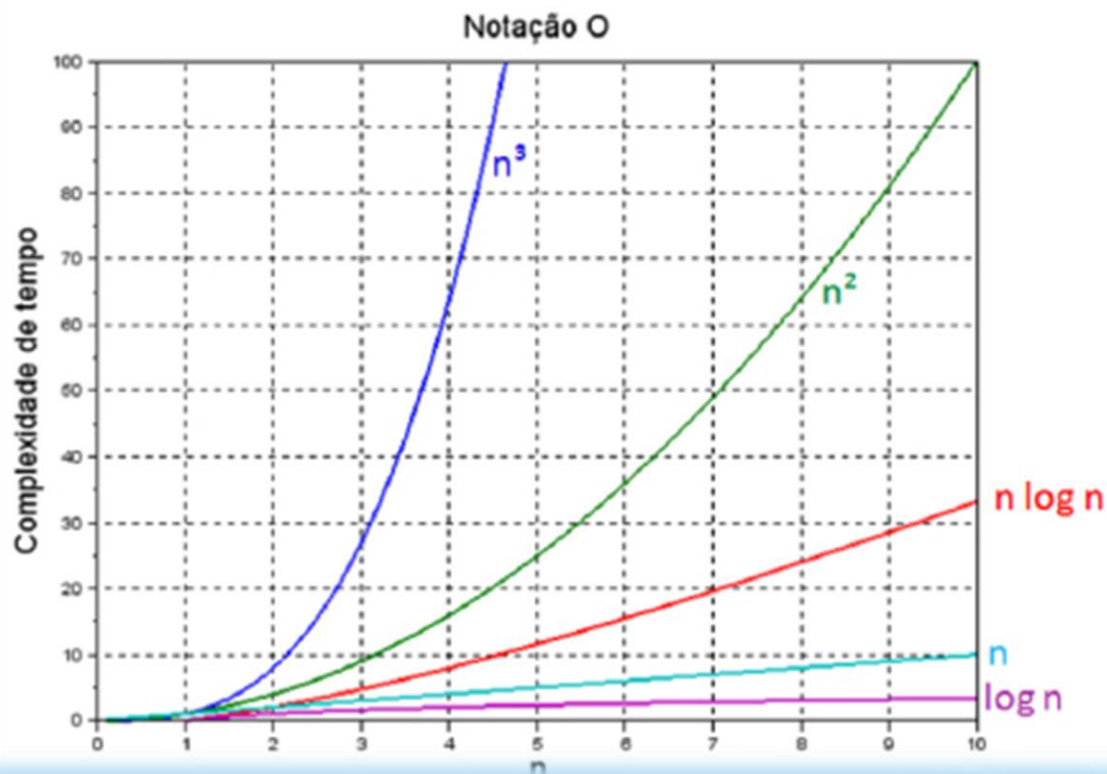


Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Exercício 05 – Resposta

Considerando o gráfico abaixo, podemos perceber que ele exibe uma comparação entre as grandezas da Notação O para alguns tipos de algoritmos. Pesquise e comente a eficiência de cada tipo de algoritmo, no tratamento de entradas para grandes valores de n .



$O(n^3)$ e $O(n^2)$: Custo computacional elevado a medida que o n aumenta
 $O(n \log n)$ e $O(n)$: Aumento linear do custo, sendo $O(n \log n)$ mais caro
 $O(\log n)$: É o algoritmo de menor custo, pois chega num ponto que o custo estabiliza

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CONTINUAÇÃO...

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 4:

```
#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();
    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); }
```

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 4:

Qual a complexidade deste código?



```
#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();
    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); }
```

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 4:

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

1º PASSO: ACHAR AS REPETIÇÕES

```
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;
}
```

Nesse caso estou trabalhando com 2 variáveis TAM1 e TAM2, ou seja, não posso chamar tudo de “n”. Para diferenciar vou chamar um de “n” e outro de “m”.

```
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); }
```


Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 4:

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

1º PASSO: ACHAR AS REPETIÇÕES

```
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;
}
```

$O(n)$ $O(m)$

```
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); }
```

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 4:

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

2º PASSO: VERIFICAR A COMPLEXIDADE DAS FUNÇÕES/MÉTODOS PRÓPRIOS DA LINGUAGEM (SE UTILIZADO).

```
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]) {
                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); }
```

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 4:

```
#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); }
```

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 4:

DEPOIS DE TER ENCONTRADO A COMPLEXIDADE DE TUDO DO MEU CÓDIGO. O QUE FOR CONSTANTE EU "IGNORO".

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 4:

```
#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();
    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); }
```

$O(n)$ $O(m)$

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

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 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

```
#include<iostream>
#include<vector>
#include<algorithm> // std::sort
#include<locale>
using namespace std;

// Essas duas funções fazem a mesma coisa.
// Recebe o vector de "idades", e verifica se no "vector"
// a menor idade aparece pelo menos duas vezes
bool exemplo5(vector<int> idades) {
    // size() = Função que retorna o número de elementos do vetor.
    int TAM = idades.size();
    int v[]={24, 33, 18, 5, 1, 5};
    int menor_idade = 100; // Inicia-se com 100 anos

    // Pega-se a menor idade, percorre-se no "for", e encontra-se
    // a menor idade
    for(int i=0; i<TAM; i++){
        if(idades[i] < menor_idade) {
            menor_idade = idades[i]; }
    }
    int cont = 0;

    // Percorro esse "for" e vejo se essa menor idade
    // está presente nele, e cada vez que ele estiver presente,
    // eu incremento essa variável "cont".
    for(int i=0; i<TAM; i++){
        if(v[i] == menor_idade) {
            cont++; }
    }
    // Se o meu "cont" for mais de um, quer dizer que a minha
    // menor idade foi repetida. Logo retorno TRUE.
    return cont > 1;
}
```

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 5 E
EXEMPLO 6
(COMPARAÇÃO DE
DUAS FUNÇÕES
QUE FAZEM A
MESMA COISA SÓ
QUE CONSTRUÍDAS
DE FORMA
DIFERENTE):

PARTE 1

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

```
bool exemplo6(vector<int> idades) {  
    //Pego meu vector de "idades" e dou um  
    //"sort()", que ordenará do menor para po maior.  
    sort(idades.begin(), idades.end());  
  
    // Se eu quero saber se a menor idade aparece  
    // pelo menos duas vezes, verifico a posição 0  
    // do meu vector, e verifico também a posição 1  
    // do me vector, e comparo e vejo se são iguais !!!  
    return idades[0] == idades[1];  
}  
  
int main() {  
    setlocale(LC_ALL, "portuguese");  
    std::vector<int> id;  
    id.push_back(15); id.push_back(23); id.push_back(5); id.push_back(42);  
    id.push_back(8); id.push_back(5);  
    cout << "\n Função 1: " << exemplo5(id) << " Função 2: " << exemplo6(id); }
```

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 5 E
EXEMPLO 6
(COMPARAÇÃO DE
DUAS FUNÇÕES
QUE FAZEM A
MESMA COISA SÓ
QUE CONSTRUÍDAS
DE FORMA
DIFERENTE):

PARTE 2

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

```
#include<iostream>
#include<vector>
#include<algorithm> // std::sort
#include<locale>
using namespace std;
```

```
// Essas duas funções fazem a mesma coisa.
// Recebe o vector de "idades", e verifica se no "vector"
// a menor idade aparece pelo menos duas vezes.
```

```
bool exemplo5(vector<int> idades) {
    // size() = Função que retorna o número de elementos do vetor.
    int TAM = idades.size();
    int v[]={24, 33, 18, 5, 1, 5};
    int menor_idade = 100; // Inicia-se com 100 anos

    // Pega-se a menor idade, percorre-se no "for", e encontra-se
    // a menor idade
    for(int i=0; i<TAM; i++){
        if(idades[i] < menor_idade) {
            menor_idade = idades[i]; }
    }
    int cont = 0;

    // Percorro esse "for" e vejo se essa menor idade
    // está presente nele, e cada vez que ele estiver presente,
    // eu incremento essa variável "cont".
    for(int i=0; i<TAM; i++){
        if(v[i] == menor_idade) {
            cont++; }
    }

    // Se o meu "cont" for mais de um, quer dizer que a minha
    // menor idade foi repetida. Logo retorno TRUE.
    return cont > 1;
}
```

Qual a complexidade deste código?

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 5 E EXEMPLO 6 (COMPARAÇÃO DE DUAS FUNÇÕES QUE FAZEM A MESMA COISA SÓ QUE CONSTRUÍDAS DE FORMA DIFERENTE):

PARTE 1

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Qual a complexidade deste código? **CÁLCULO DA COMPLEXIDADE DE ALGORITMO:**

```
bool exemplo6(vector<int> idades) {  
    //Pego meu vector de "idades" e dou um  
    //"sort()", que ordenará do menor para po maior.  
    sort(idades.begin(), idades.end());  
  
    // Se eu quero saber se a menor idade aparece  
    // pelo menos duas vezes, verifico a posição 0  
    // do meu vector, e verifico também a posição 1  
    // do me vector, e comparo e vejo se são iguais !!!  
    return idades[0] == idades[1];  
}
```

```
int main() {  
    setlocale(LC_ALL, "portuguese");  
    std::vector<int> id;  
    id.push_back(15); id.push_back(23); id.push_back(5); id.push_back(42);  
    id.push_back(8); id.push_back(5);  
    cout << "\n Função 1: " << exemplo5(id) << " Função 2: " << exemplo6(id); }  
}
```

EXEMPLO 5 E
EXEMPLO 6
(COMPARAÇÃO DE
DUAS FUNÇÕES
QUE FAZEM A
MESMA COISA SÓ
QUE CONSTRUÍDAS
DE FORMA
DIFERENTE):

PARTE 2

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

```
#include<iostream>
#include<vector>
#include<algorithm> // std::sort
#include<locale>
using namespace std;
```

```
// Essas duas funções fazem a mesma coisa.
// Recebe o vector de "idades", e verifica se no "vector"
// a menor idade aparece pelo menos duas vezes
bool exemplo5(vector<int> idades) {
    // size() = Função que retorna o número de elementos do vetor.
    int TAM = idades.size();
    int v[]={24, 33, 18, 5, 1, 5};
    int menor_idade = 100; // Inicia-se com 100 anos

    // Pega-se a menor idade, percorre-se no "for", e encontra-se
    // a menor idade
    for(int i=0; i<TAM; i++){ ← O(n)
        if(idades[i] < menor_idade) {
            menor_idade = idades[i]; }
    }
    int cont = 0;

    // Percorro esse "for" e vejo se essa menor idade
    // está presente nele, e cada vez que ele estiver presente,
    // eu incremento essa variável "cont".
    for(int i=0; i<TAM; i++){ ← O(n)
        if(v[i] == menor_idade) {
            cont++; }
    }

    // Se o meu "cont" for mais de um, quer dizer que a minha
    // menor idade foi repetida. Logo retorno TRUE.
    return cont > 1;
}
```

1º PASSO: ACHAR AS REPETIÇÕES

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:
EXEMPLO 5 E
EXEMPLO 6
(COMPARAÇÃO DE DUAS FUNÇÕES QUE FAZEM A MESMA COISA SÓ QUE CONSTRUÍDAS DE FORMA DIFERENTE):

PARTE 1

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

1º PASSO: ACHAR AS REPETIÇÕES

NÃO TEM !!!

```
bool exemplo6(vector<int> idades) {  
    //Pego meu vector de "idades" e dou um  
    //"sort()", que ordenará do menor para po maior.  
    sort(idades.begin(), idades.end());  
  
    // Se eu quero saber se a menor idade aparece  
    // pelo menos duas vezes, verifico a posição 0  
    // do meu vector, e verifico também a posição 1  
    // do me vector, e comparo e vejo se são iguais !!!  
    return idades[0] == idades[1];  
}
```

```
int main() {  
    setlocale(LC_ALL, "portuguese");  
    std::vector<int> id;  
    id.push_back(15); id.push_back(23); id.push_back(5); id.push_back(42);  
    id.push_back(8); id.push_back(5);  
    cout << "\n Função 1: " << exemplo5(id) << " Função 2: " << exemplo6(id); }
```

CÁLCULO DA
COMPLEXIDADE
DE ALGORITMO:

EXEMPLO 5 E
EXEMPLO 6
(COMPARAÇÃO DE
DUAS FUNÇÕES
QUE FAZEM A
MESMA COISA SÓ
QUE CONSTRUÍDAS
DE FORMA
DIFERENTE):

PARTE 2

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

```
#include<iostream>
#include<vector>
#include<algorithm> // std::sort
#include<locale>
using namespace std;
```

2º PASSO: VERIFICAR A COMPLEXIDADE DAS FUNÇÕES/MÉTODOS PRÓPRIOS DA LINGUAGEM (SE UTILIZADO).

```
// Essas duas funções fazem a mesma coisa.
// Recebe o vector de "idades", e verifica se no "vector"
// a menor idade aparece pelo menos duas vezes
bool exemplo5(vector<int> idades) {
    // size() = Função que retorna o número de elementos do vetor.
    int TAM = idades.size(); ← O(1)
    int v[]={24, 33, 18, 5, 1, 5};
    int menor_idade = 100; // Inicia-se com 100 anos

    // Pega-se a menor idade, percorre-se no "for", e encontra-se
    // a menor idade
    for(int i=0; i<TAM; i++){ ← O(n)
        if(idades[i] < menor_idade) {
            menor_idade = idades[i]; }
    }
    int cont = 0;

    // Percorro esse "for" e vejo se essa menor idade
    // está presente nele, e cada vez que ele estiver presente,
    // eu incremento essa variável "cont".
    for(int i=0; i<TAM; i++){ ← O(n)
        if(v[i] == menor_idade) {
            cont++; }
    }

    // Se o meu "cont" for mais de um, quer dizer que a minha
    // menor idade foi repetida. Logo retorno TRUE.
    return cont > 1;
}
```

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:
EXEMPLO 5 E
EXEMPLO 6
(COMPARAÇÃO DE
DUAS FUNÇÕES
QUE FAZEM A
MESMA COISA SÓ
QUE CONSTRUÍDAS
DE FORMA
DIFERENTE):

PARTE 1

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

2º PASSO: VERIFICAR A COMPLEXIDADE DAS FUNÇÕES/MÉTODOS PRÓPRIOS DA LINGUAGEM (SE UTILIZADO).

```
bool exemplo6(vector<int> idades) {  
    //Pego meu vector de "idades" e dou um  
    //"sort()", que ordenará do menor para po maior.  
    sort(idades.begin(), idades.end()); ← ???  
  
    // Se eu quero saber se a menor idade aparece  
    // pelo menos duas vezes, verifico a posição 0  
    // do meu vector, e verifico também a posição 1  
    // do me vector, e comparo e vejo se são iguais !!!  
    return idades[0] == idades[1];  
}  
  
int main() {  
    setlocale(LC_ALL, "portuguese");  
    std::vector<int> id;  
    id.push_back(15); id.push_back(23); id.push_back(5); id.push_back(42);  
    id.push_back(8); id.push_back(5);  
    cout << "\n Função 1: " << exemplo5(id) << " Função 2: " << exemplo6(id); }
```

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:
EXEMPLO 5 E
EXEMPLO 6
(COMPARAÇÃO DE DUAS FUNÇÕES QUE FAZEM A MESMA COISA SÓ QUE CONSTRUÍDAS DE FORMA DIFERENTE):

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 5 e 6:


The screenshot shows the homepage of cplusplus.com. At the top, there is a search bar with the text "Search: sort" and a "Go" button. To the right of the search bar, it says "Not logged in" with "register" and "log in" buttons. Below the search bar, there is a "home" button. On the left side, there is a navigation menu with the following links: "C++", "Information", "Tutorials", "Reference", "Articles", and "Forum". The main content area is divided into six sections: "Information", "Tutorials", "Reference", "Articles", "Forum", and "C++ Search". Each section contains a brief description and a list of links or resources. The "Information" section includes links to "Description of the C++ language", "History of the C++ language", and "F.A.Q., Frequently Asked Questions". The "Tutorials" section includes a link to "C++ Language: Collection of tutorials covering all the features of this versatile and powerful language. Including detailed explanations of pointers, functions, classes and templates, among others..." and a link to "more...". The "Reference" section includes links to "C library: The popular C library, is also part of the of C++ language library.", "IOStream library. The standard C++ library for Input/Output operations.", "String library. Library defining the string class.", "Standard containers. Vectors, lists, maps, sets...", and a link to "more...". The "Articles" section includes links to "Algorithms", "Standard library", "C++11", "Windows API", and "Other...". The "Forum" section includes a link to "Message boards where members can exchange knowledge". The "C++ Search" section includes a link to "Search this website".

Search: sort Not logged in

home

C++

- Information
- Tutorials
- Reference
- Articles
- Forum

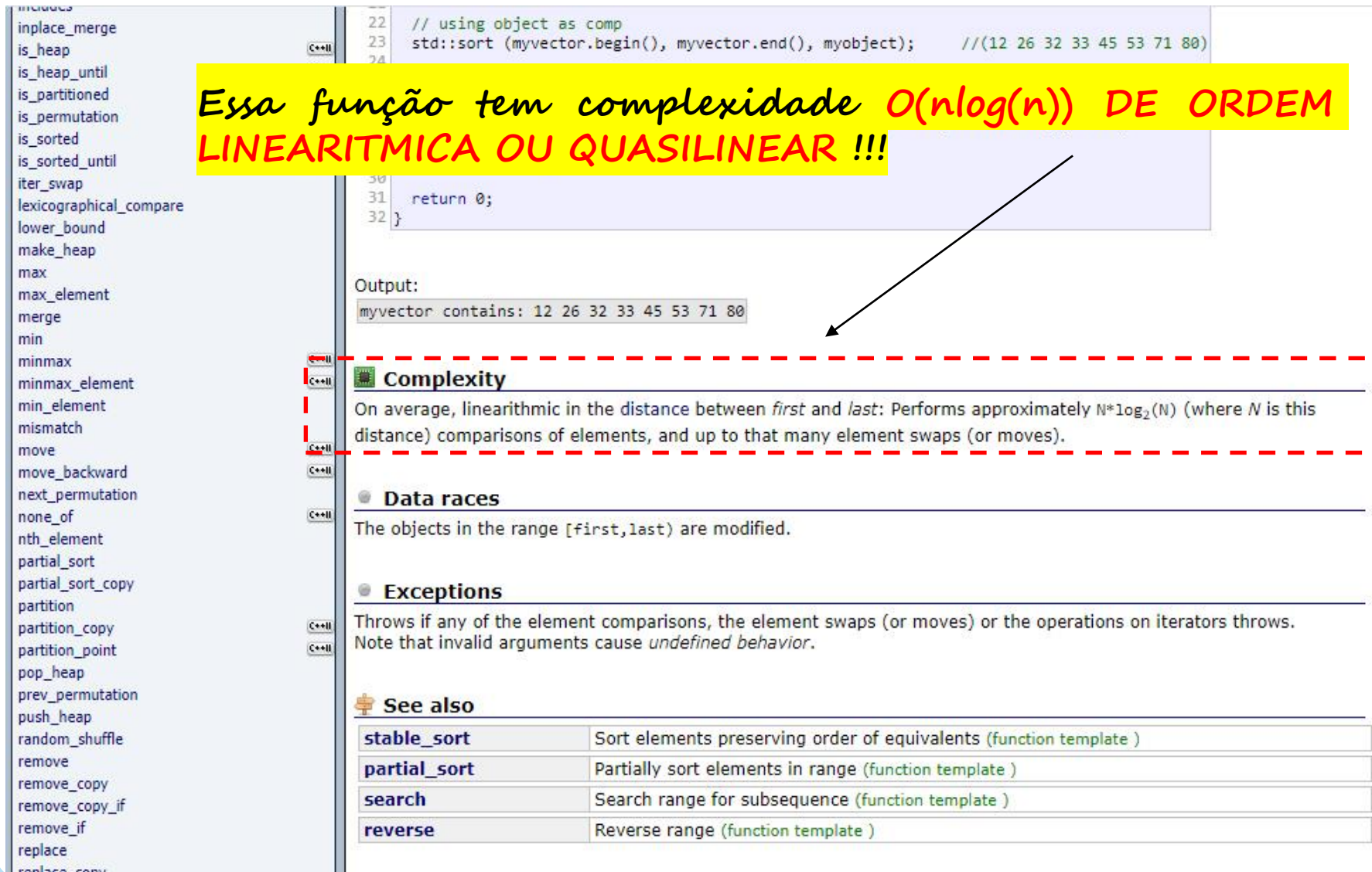
Welcome to **cplusplus.com**  © The C++ Resources Network, 2020

Information	Tutorials
<p>General information about the C++ programming language, including non-technical documents and descriptions:</p> <ul style="list-style-type: none">Description of the C++ languageHistory of the C++ languageF.A.Q., Frequently Asked Questions	<p>Learn the C++ language from its basics up to its most advanced features.</p> <ul style="list-style-type: none">C++ Language: Collection of tutorials covering all the features of this versatile and powerful language. Including detailed explanations of pointers, functions, classes and templates, among others...more...
Reference	Articles
<p>Description of the most important classes, functions and objects of the Standard Language Library, with descriptive fully-functional short programs as examples:</p> <ul style="list-style-type: none">C library: The popular C library, is also part of the of C++ language library.IOStream library. The standard C++ library for Input/Output operations.String library. Library defining the string class.Standard containers. Vectors, lists, maps, sets...more...	<p>User-contributed articles, organized into different categories:</p> <ul style="list-style-type: none">AlgorithmsStandard libraryC++11Windows APIOther... <p>You can contribute your own articles!</p>
Forum	C++ Search
<p>Message boards where members can exchange knowledge</p>	<p>Search this website:</p>

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 5 e 6:



Essa função tem complexidade $O(n \log(n))$ DE ORDEM LINEARITMICA OU QUASILINEAR !!!

```
22 // using object as comp
23 std::sort (myvector.begin(), myvector.end(), myobject);    //(12 26 32 33 45 53 71 80)
24
```

Output:
myvector contains: 12 26 32 33 45 53 71 80

Complexity
On average, linearithmic in the distance between *first* and *last*: Performs approximately $N \cdot \log_2(N)$ (where N is this distance) comparisons of elements, and up to that many element swaps (or moves).

Data races
The objects in the range $[first, last)$ are modified.

Exceptions
Throws if any of the element comparisons, the element swaps (or moves) or the operations on iterators throws. Note that invalid arguments cause *undefined behavior*.

See also

stable_sort	Sort elements preserving order of equivalents (function template)
partial_sort	Partially sort elements in range (function template)
search	Search range for subsequence (function template)
reverse	Reverse range (function template)

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

2º PASSO: VERIFICAR A COMPLEXIDADE DAS FUNÇÕES/MÉTODOS PRÓPRIOS DA LINGUAGEM (SE UTILIZADO).

```
bool exemplo6(vector<int> idades) {  
    //Pego meu vector de "idades" e dou um  
    //"sort()", que ordenará do menor para po maior.  
    sort(idades.begin(), idades.end()); ← O(nlog(n))  
  
    // Se eu quero saber se a menor idade aparece  
    // pelo menos duas vezes, verifico a posição 0  
    // do meu vector, e verifico também a posição 1  
    // do me vector, e comparo e vejo se são iguais !!!  
    return idades[0] == idades[1];  
}  
  
int main() {  
    setlocale(LC_ALL, "portuguese");  
    std::vector<int> id;  
    id.push_back(15); id.push_back(23); id.push_back(5); id.push_back(42);  
    id.push_back(8); id.push_back(5);  
    cout << "\n Função 1: " << exemplo5(id) << " Função 2: " << exemplo6(id); }
```

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO
(COMPARAÇÃO DE DUAS FUNÇÕES QUE FAZEM A MESMA COISA SÓ QUE CONSTRUÍDAS DE FORMA DIFERENTE):

PARTE 2

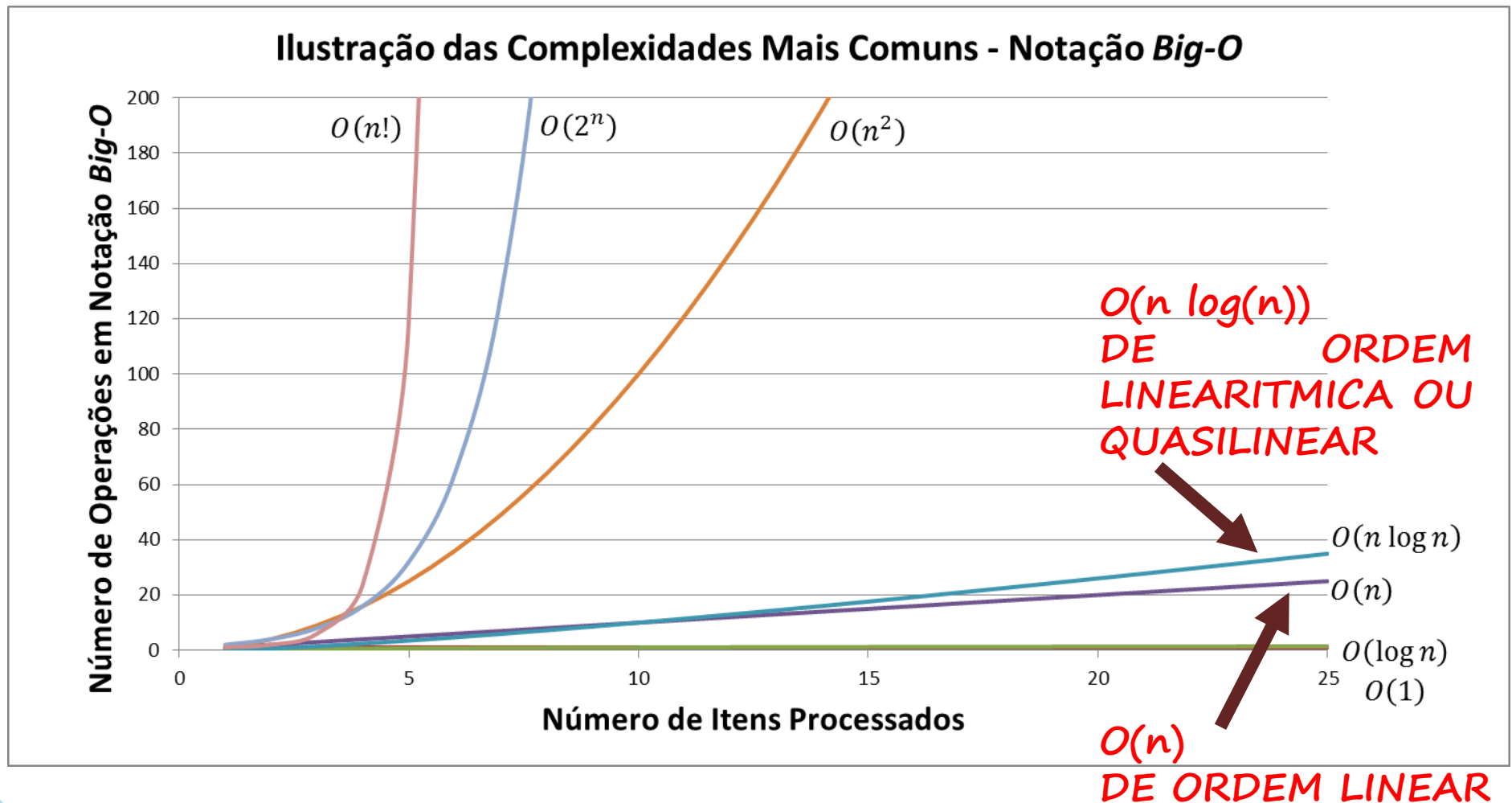
Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 5 e 6:

Gráfico de cada complexidade em função de N

(DEMONSTRA COMO OS TEMPOS DE EXECUÇÃO SÃO AFETADOS PELO NÚMERO DE ITENS



Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 5 e 6:

ESTAMOS VENDO QUE NESTE EXEMPLO QUE NEM SEMPRE UM CÓDIGO "MENOR - COM POUCAS INSTRUÇÕES" É MAIS RÁPIDO !!!!. OU SEJA, O CÓDIGO DO EXEMPLO 6 EXECUTARÁ MAIS LENTAMENTE QUE O CÓDIGO DO EXEMPLO 5.

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 5 e 6:

AGORA TEMOS QUE ESCREVER EM UMA ÚNICA COMPLEXIDADE.

EXEMPLO 5:

$$O(n) + O(n)$$

$$2 * O(n)$$

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

Esse é o termo final: $O(n)$.

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 5 e 6:

AGORA TEMOS QUE ESCREVER EM UMA ÚNICA COMPLEXIDADE.

EXEMPLO 6:

$$O(n \log(n))$$

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

Esse é o termo final: $O(n \log(n))$.

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

```
#include<iostream>
#include<vector>
#include<algorithm> // Utilização do count()
#include <set> // Utilização do set<int>
#include<locale>
using namespace std;

bool exemplo7(set<int> va, vector<int> vb) {
    int tamanho = vb.size();
    for(int i=0; i<tamanho; i++){
        // count() - Retorna o número de elementos.
        if(va.count(vb[i])) {
            return true;
        }
    }
    return false;
}
}
```

```
int main() {
    setlocale(LC_ALL, "portuguese");
    std::vector<int> id;
    std::set<int> num; //Conjunto vazio de inteiros
    num.insert(11);
    id.push_back(11); id.push_back(23); id.push_back(5); id.push_back(42);
    id.push_back(8); id.push_back(5);
    cout << "\n Função exemplo 7: " << exemplo7(num, id);
}
```

**CÁLCULO DA
COMPLEXIDADE
E
ALGORITMO:
EXEMPLO 7:**

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Qual a complexidade deste código?

```
#include<iostream>
#include<vector>
#include<algorithm> // Utilização do count()
#include <set> // Utilização do set<int>
#include<locale>
using namespace std;

bool exemplo7(set<int> va, vector<int> vb) {
    int tamanho = vb.size();
    for(int i=0; i<tamanho; i++){
        // count() - Retorna o número de elementos.
        if(va.count(vb[i])) {
            return true;
        }
    }
    return false;
}
```

**CÁLCULO DA
COMPLEXIDADE
E
DE
ALGORITMO:
EXEMPLO 7:**

```
int main() {
    setlocale(LC_ALL, "portuguese");
    std::vector<int> id;
    std::set<int> num; //Conjunto vazio de inteiros
    num.insert(11);
    id.push_back(11); id.push_back(23); id.push_back(5); id.push_back(42);
    id.push_back(8); id.push_back(5);
    cout << "\n Função exemplo 7: " << exemplo7(num, id);
}
```


Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

1º PASSO: ACHAR AS REPETIÇÕES

```
#include<iostream>
#include<vector>
#include<algorithm> // Utilização do count()
#include <set> // Utilização do set<int>
#include<locale>
using namespace std;
```

```
bool exemplo7(set<int> va, vector<int> vb) {
    int tamanho = vb.size();
    for(int i=0; i<tamanho; i++){
        // count() - Retorna o número de elementos.
        if(va.count(vb[i])) {
            return true;
        }
    }
    return false;
}
```

$O(n)$

CÁLCULO DA
COMPLEXIDADE
E DE
ALGORITMO:
EXEMPLO 7:

```
int main() {
    setlocale(LC_ALL, "portuguese");
    std::vector<int> id;
    std::set<int> num; //Conjunto vazio de inteiros
    num.insert(11);
    id.push_back(11); id.push_back(23); id.push_back(5); id.push_back(42);
    id.push_back(8); id.push_back(5);
    cout << "\n Função exemplo 7: " << exemplo7(num, id);
}
```


Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

```
#include<iostream>
#include<vector>
#include<algorithm> // Utilização do count()
#include <set> // Utilização do set<int>
#include<locale>
using namespace std;

bool exemplo7(set<int> va, vector<int> vb) {
    int tamanho = vb.size();
    for(int i=0; i<tamanho; i++){
        // count() - Retorna o número de elementos.
        if(va.count(vb[i])) {
            return true;
        }
        return false;
    }
}
```

2º PASSO: VERIFICAR A COMPLEXIDADE DAS FUNÇÕES/MÉTODOS PRÓPRIOS DA LINGUAGEM (SE UTILIZADO).

$O(n)$

```
int main() {
    setlocale(LC_ALL, "portuguese");
    std::vector<int> id;
    std::set<int> num; //Conjunto vazio de inteiros
    num.insert(11);
    id.push_back(11); id.push_back(23); id.push_back(5); id.push_back(42);
    id.push_back(8); id.push_back(5);
    cout << "\n Função exemplo 7: " << exemplo7(num, id);
}
```

CÁLCULO DA COMPLEXIDADE E DE ALGORITMO:
EXEMPLO 7:

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

```
#include<iostream>
#include<vector>
#include<algorithm> // Utilização do count()
#include <set> // Utilização do set<int>
#include<locale>
using namespace std;
```

```
bool exemplo7(set<int> va, vector<int> vb) {
    int tamanho = vb.size();
    for(int i=0; i<tamanho; i++){
        // count() - Retorna o número de elementos.
        if(va.count(vb[i])) {
            return true;
        }
        return false;
    }
}
```

```
int main() {
    setlocale(LC_ALL, "portuguese");
    std::vector<int> id;
    std::set<int> num; //Conjunto vazio de inteiros
    num.insert(11);
    id.push_back(11); id.push_back(23); id.push_back(5); id.push_back(42);
    id.push_back(8); id.push_back(5);
    cout << "\n Função exemplo 7: " << exemplo7(num, id);
}
```

2º PASSO: VERIFICAR A COMPLEXIDADE DAS FUNÇÕES/MÉTODOS PRÓPRIOS DA LINGUAGEM (SE UTILIZADO).

$O(1)$

$O(n)$

???

CÁLCULO DA COMPLEXIDADE E DE ALGORITMO:
EXEMPLO 7:

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 7:

inplace_merge
is_heap
is_heap_until
is_partitioned
is_permutation
is_sorted
is_sorted_until
iter_swap
lexicographical_compare
lower_bound
make_heap
max
max_element
merge
min
minmax
minmax_element
min_element
mismatch
move
move_backward
next_permutation
none_of
nth_element
partial_sort
partial_sort_copy
partition
partition_copy
partition_point
pop_heap
prev_permutation

```
17 return 0;  
18 }
```

Essa função tem complexidade $O(n)$ DE ORDEM LINEAR !!!

Output:

```
10 appears 3 times.  
20 appears 3 times.
```

Complexity

Linear in the distance between *first* and *last*: Compares once each element.

Data races

The objects in the range `[first,last)` are accessed (each object is accessed exactly once).

Exceptions

Throws if either an element comparison or an operation on an iterator throws.
Note that invalid arguments cause *undefined behavior*.

See also

for_each	Apply function to range (function template)
count_if	Return number of elements in range satisfying condition (function template)
find	Find value in range (function template)
replace	Replace value in range (function template)

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

```
#include<iostream>
#include<vector>
#include<algorithm> // Utilização do count()
#include <set> // Utilização do set<int>
#include<locale>
using namespace std;
```

```
bool exemplo7(set<int> va, vector<int> vb) {
    int tamanho = vb.size();
    for(int i=0; i<tamanho; i++){
        // count() - Retorna o número de elementos.
        if(va.count(vb[i])) {
            return true;
        }
    }
    return false;
}
```

```
int main() {
    setlocale(LC_ALL, "portuguese");
    std::vector<int> id;
    std::set<int> num; //Conjunto vazio de inteiros
    num.insert(11);
    id.push_back(11); id.push_back(23); id.push_back(5); id.push_back(42);
    id.push_back(8); id.push_back(5);
    cout << "\n Função exemplo 7: " << exemplo7(num, id);
}
```

2º PASSO: VERIFICAR A COMPLEXIDADE DAS FUNÇÕES/MÉTODOS PRÓPRIOS DA LINGUAGEM (SE UTILIZADO).

$O(1)$

$O(n)$

$O(n)$

CÁLCULO DA COMPLEXIDADE E DE ALGORITMO:
EXEMPLO 7:

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

```
#include<iostream>
#include<vector>
#include<algorithm> // Utilização do count()
#include <set> // Utilização do set<int>
#include<locale>
using namespace std;

bool exemplo7(set<int> va, vector<int> vb) {
    int tamanho = vb.size();
    for(int i=0; i<tamanho; i++){
        // count() - Retorna o número de elementos.
        if(va.count(vb[i])) {
            return true;
        }
        return false;
    }
}
```

2º PASSO: VERIFICAR A COMPLEXIDADE DAS FUNÇÕES/MÉTODOS PRÓPRIOS DA LINGUAGEM (SE UTILIZADO).

$O(1)$

$O(n)$

$O(1)$

$O(n)$

$O(1)$

```
int main() {
    setlocale(LC_ALL, "portuguese");
    std::vector<int> id;
    std::set<int> num; //Conjunto vazio de inteiros
    num.insert(11);
    id.push_back(11); id.push_back(23); id.push_back(5); id.push_back(42);
    id.push_back(8); id.push_back(5);
    cout << "\n Função exemplo 7: " << exemplo7(num, id);
}
```

CÁLCULO DA COMPLEXIDADE E DE ALGORITMO:
EXEMPLO 7:

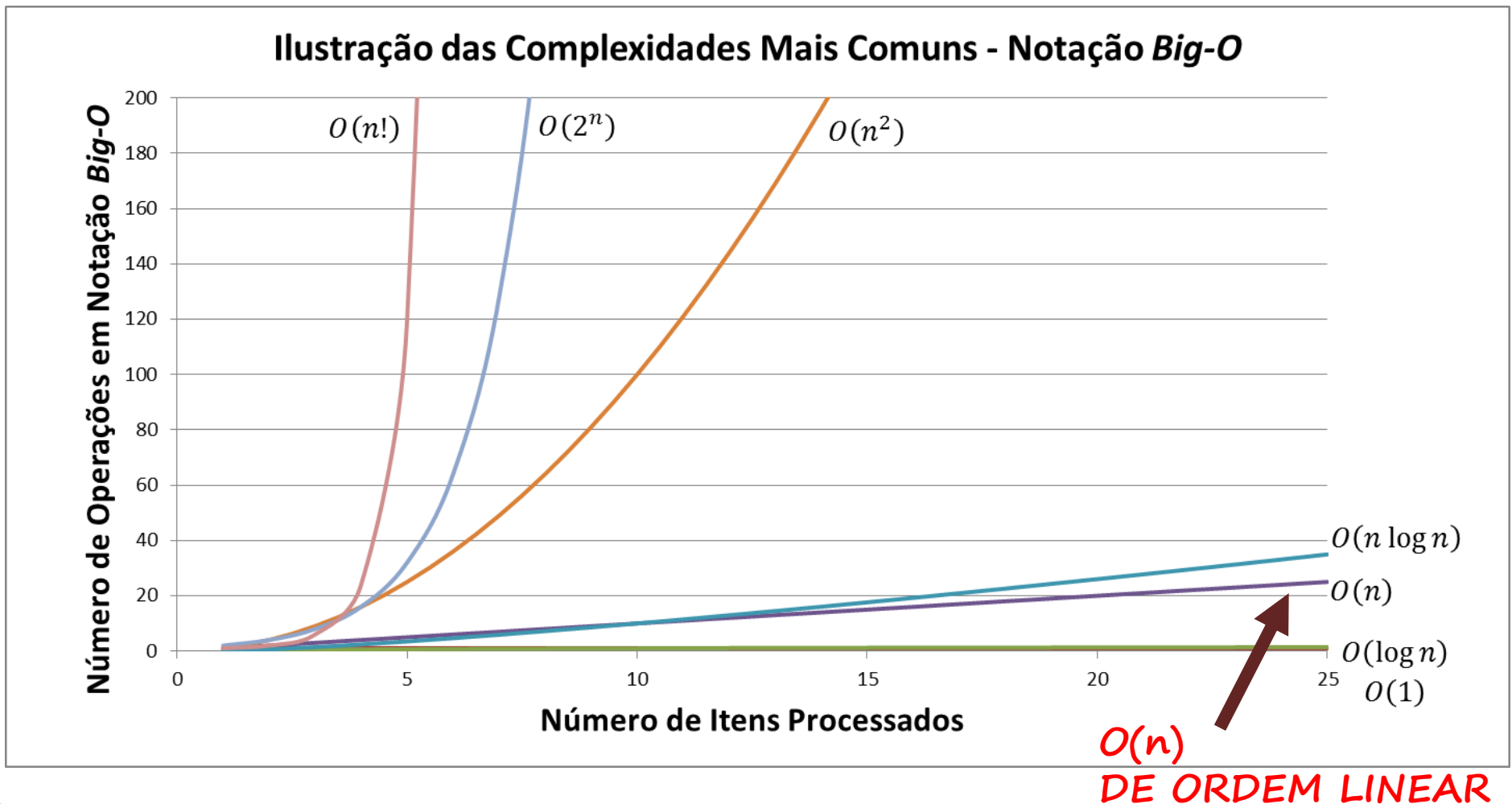
Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 7:

Gráfico de cada complexidade em função de N

(DEMONSTRA COMO OS TEMPOS DE EXECUÇÃO SÃO AFETADOS PELO NÚMERO DE ITENS)



Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 7:

DEPOIS DE TER ENCONTRADO A COMPLEXIDADE DE TUDO DO MEU CÓDIGO. O QUE FOR CONSTANTE EU "IGNORO".

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

```
#include<iostream>
#include<vector>
#include<algorithm> // Utilização do count()
#include <set> // Utilização do set<int>
#include<locale>
using namespace std;

bool exemplo7(set<int> va, vector<int> vb) {
    int tamanho = vb.size();
    for(int i=0; i<tamanho; i++){
        // count() - Retorna o número de elementos.
        if(va.count(vb[i])) {
            return true;
        }
    }
    return false;
}
```

2º PASSO: VERIFICAR A COMPLEXIDADE DAS FUNÇÕES/MÉTODOS PRÓPRIOS DA LINGUAGEM (SE UTILIZADO).

$O(n)$

$O(n)$

```
int main() {
    setlocale(LC_ALL, "portuguese");
    std::vector<int> id;
    std::set<int> num; //Conjunto vazio de inteiros
    num.insert(11);
    id.push_back(11); id.push_back(23); id.push_back(5); id.push_back(42);
    id.push_back(8); id.push_back(5);
    cout << "\n Função exemplo 7: " << exemplo7(num, id);
}
```

CÁLCULO DA COMPLEXIDADE E DE ALGORITMO:
EXEMPLO 7:

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 7:

FINALIZANDO:

EXEMPLO 7:

$$O(n) * O(n)$$

$$O(n^2)$$

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

Esse é o termo final: $O(n^2)$.

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

TEMA 2 - RECURSIVIDADE

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

RECURSIVIDADE

Em programação, a recursividade é um mecanismo útil e poderoso que permite a uma função chamar a si mesma *direta ou indiretamente*, ou seja, uma função é dita recursiva se ela contém pelo menos uma chamada *explícita ou implícita* a si própria.

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

RECURSIVIDADE

Funções Recursivas

Uma função que chama a si mesma (ela própria) é chamada de *função recursiva* durante a execução.

Para a função realizar a resolução de problemas ela faz chamadas de si própria, para resolver geralmente partes mais simples do problema, até que seja totalmente resolvido.

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

RECURSIVIDADE

Funções Recursivas

Essa função pode ser trabalhadas de duas formas:

- **Recursão direta – Chama a si própria diretamente.**
- **Recursão indireta – Por meio de uma outra função para resolver um problema.**

Ao manipularmos funções recursivas, temos que ter em mente que a função trabalha dividindo o PROBLEMA em dois casos (duas partes): Trivial (base) e geral.

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

ALGORTIMOS RECURSIVOS

(Características)

- i) algoritmos recursivos quase sempre consomem mais recursos (especialmente memória, devido uso intensivo da pilha) do computador.
- ii) algoritmos recursivos são mais difíceis de serem depurados, especialmente quando for alta a profundidade de recursão (número máximo de chamadas simultâneas).

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

RECURSIVIDADE

- Conforme dito anteriormente, a *recursividade* é uma forma de resolver problemas por meio da divisão dos problemas em problemas menores de *mesma natureza*.
- Se a natureza dos subproblemas é a mesma do problema, o mesmo método usado para reduzir o problema pode ser usado para reduzir os subproblemas e assim por diante.
- Quando devemos parar? Quando alcançarmos um **caso trivial** que conhecemos a solução.

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

RECURSIVIDADE

- Exemplos de **casos triviais (BASE)**:
 - Qual o fatorial de 0 ?
 - Quanto é um dado X multiplicado por 1 ?
 - Quanto é um dado X multiplicado por 0 ?
 - Quanto é um dado X elevado a 1 ?
 - Quantos elementos tem em uma lista vazia?

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

RECURSIVIDADE

- Assim, um *processo recursivo* para a solução de um problema consiste em duas partes:
 - O caso trivial, cuja solução é conhecida;
 - Um método geral que reduz o problema a um ou mais problemas menores (subproblemas) de mesma natureza.
- Muitas funções podem ser definidas recursivamente. Para isso, é preciso identificar as duas partes acima.
- Exemplo: fatorial de um número e o n -ésimo termo da seqüência de Fibonacci.

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- A função *fatorial* de um inteiro não negativo pode ser definida como:

$$fatorial(n) = \begin{cases} 1 & \text{se } n = 0 \\ n \times fatorial(n - 1) & \text{se } n > 0 \end{cases}$$

- Esta definição estabelece um *processo recursivo* para calcular o fatorial de um inteiro n .
- Caso trivial: $n=0$.
- Método geral: $n \times (n-1)!$. **(RECURSIVO)**

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- Assim, usando-se este *processo recursivo*, o cálculo de **4!**, por exemplo, é feito como a seguir:

4! =

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- Assim, usando-se este *processo recursivo*, o cálculo de **4!**, por exemplo, é feito como a seguir:

$$4! = 4 * 3!$$

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- Assim, usando-se este *processo recursivo*, o cálculo de **4!**, por exemplo, é feito como a seguir:

$$\begin{aligned} 4! &= 4 * 3! \\ &= 4 * (3 * 2!) \end{aligned}$$

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- Assim, usando-se este *processo recursivo*, o cálculo de **4!**, por exemplo, é feito como a seguir:

$$\begin{aligned}4! &= 4 * 3! \\ &= 4 * (3 * 2!) \\ &= 4 * (3 * (2 * 1!))\end{aligned}$$

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- Assim, usando-se este *processo recursivo*, o cálculo de **4!**, por exemplo, é feito como a seguir:

$$\begin{aligned}4! &= 4 * 3! \\ &= 4 * (3 * 2!) \\ &= 4 * (3 * (2 * 1!)) \\ &= 4 * (3 * (2 * (1 * 0!)))\end{aligned}$$

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- Assim, usando-se este *processo recursivo*, o cálculo de **4!**, por exemplo, é feito como a seguir:

$$\begin{aligned}4! &= 4 * 3! \\ &= 4 * (3 * 2!) \\ &= 4 * (3 * (2 * 1!)) \\ &= 4 * (3 * (2 * (1 * 0!))) \\ &= 4 * (3 * (2 * (1 * 1)))\end{aligned}$$

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- Assim, usando-se este *processo recursivo*, o cálculo de **4!**, por exemplo, é feito como a seguir:

$$\begin{aligned}4! &= 4 * 3! \\&= 4 * (3 * 2!) \\&= 4 * (3 * (2 * 1!)) \\&= 4 * (3 * (2 * (1 * 0!))) \\&= 4 * (3 * (2 * (1 * 1))) \\&= 4 * (3 * (2 * 1))\end{aligned}$$

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- Assim, usando-se este *processo recursivo*, o cálculo de **4!**, por exemplo, é feito como a seguir:

$$\begin{aligned}4! &= 4 * 3! \\&= 4 * (3 * 2!) \\&= 4 * (3 * (2 * 1!)) \\&= 4 * (3 * (2 * (1 * 0!))) \\&= 4 * (3 * (2 * (1 * 1))) \\&= 4 * (3 * (2 * 1)) \\&= 4 * (3 * 2)\end{aligned}$$

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- Assim, usando-se este *processo recursivo*, o cálculo de **4!**, por exemplo, é feito como a seguir:

$$\begin{aligned}4! &= 4 * 3! \\&= 4 * (3 * 2!) \\&= 4 * (3 * (2 * 1!)) \\&= 4 * (3 * (2 * (1 * 0!))) \\&= 4 * (3 * (2 * (1 * 1))) \\&= 4 * (3 * (2 * 1)) \\&= 4 * (3 * 2) \\&= 4 * 6\end{aligned}$$

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- Assim, usando-se este *processo recursivo*, o cálculo de **4!**, por exemplo, é feito como a seguir:

$$\begin{aligned}4! &= 4 * 3! \\ &= 4 * (3 * 2!) \\ &= 4 * (3 * (2 * 1!)) \\ &= 4 * (3 * (2 * (1 * 0!))) \\ &= 4 * (3 * (2 * (1 * 1))) \\ &= 4 * (3 * (2 * 1)) \\ &= 4 * (3 * 2) \\ &= 4 * 6 \\ &= 24\end{aligned}$$

```
int fatorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * fatorial(n-1);
}
```



Mas como uma *função recursiva* é de fato implementada no computador?

Usando-se o mecanismo conhecido como **Pilha de Execução!**

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- Considere, novamente, o exemplo para **4!**:

```
int fatorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * fatorial(n-1);
}
```

Pilha de Execução



fatorial(4) -> return 4*fatorial(3)

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- Considere, novamente, o exemplo para **4!**:

```
int fatorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * fatorial(n-1);
}
```

Pilha de Execução



fatorial(3)	-> return 3*fatorial(2)
fatorial(4)	-> return 4*fatorial(3)

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- Considere, novamente, o exemplo para **4!**:

```
int fatorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * fatorial(n-1);
}
```

Pilha de Execução



fatorial(2)	-> return 2*fatorial(1)
fatorial(3)	-> return 3*fatorial(2)
fatorial(4)	-> return 4*fatorial(3)

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- Considere, novamente, o exemplo para **4!**:

```
int fatorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * fatorial(n-1);
}
```

Pilha de Execução



fatorial(1)	-> return 1*fatorial(0)
fatorial(2)	-> return 2*fatorial(1)
fatorial(3)	-> return 3*fatorial(2)
fatorial(4)	-> return 4*fatorial(3)

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- Considere, novamente, o exemplo para **4!**:

```
int fatorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * fatorial(n-1);
}
```

Pilha de Execução



fatorial(0)	-> return 1 (caso trivial/BASE)
fatorial(1)	-> return 1*fatorial(0)
fatorial(2)	-> return 2*fatorial(1)
fatorial(3)	-> return 3*fatorial(2)
fatorial(4)	-> return 4*fatorial(3)

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- Considere, novamente, o exemplo para **4!**:

```
int fatorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * fatorial(n-1);
}
```

Pilha de Execução



fatorial(0)
fatorial(1)
fatorial(2)
fatorial(3)
fatorial(4)

Desempilha fatorial(0)

-> return 1 (**caso trivial/BASE**)

-> return 1*fatorial(0)

-> return 2*fatorial(1)

-> return 3*fatorial(2)

-> return 4*fatorial(3)

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- Considere, novamente, o exemplo para **4!**:

```
int fatorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * fatorial(n-1);
}
```

Desempilha fatorial(1)

fatorial(1)	-> return 1*1
fatorial(2)	-> return 2*fatorial(1)
fatorial(3)	-> return 3*fatorial(2)
fatorial(4)	-> return 4*fatorial(3)

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- Considere, novamente, o exemplo para **4!**:

```
int fatorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * fatorial(n-1);
}
```

Desempilha fatorial(2)

fatorial(2)	-> return 2*1*1
fatorial(3)	-> return 3*fatorial(2)
fatorial(4)	-> return 4*fatorial(3)

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- Considere, novamente, o exemplo para **4!**:

```
int fatorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * fatorial(n-1);
}
```

Desempilha fatorial(3)

fatorial(3)
fatorial(4)

-> return 3*2*1*1

-> return 4*fatorial(3)

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- Considere, novamente, o exemplo para **4!**:

```
int fatorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * fatorial(n-1);
}
```

Desempilha fatorial(4)

fatorial(4)

-> **return 4*3*2*1*1**

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Função fatorial

- Considere, novamente, o exemplo para **4!**:

```
int fatorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * fatorial(n-1);
}
```

Resultado = 24

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Programa fatorial

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

// protótipo da função
// num -> Valor no qual será realizado o fatorial
int fatorial(int num);
int main() { // Corpo principal do programa
    setlocale(LC_ALL, "portuguese");

    // Para trabalhar com o problema FATORIAL,
    // vou precisar de uma variável inteira.
    int num, resp;
    cout << "\n\n Digite um número para ser calculado o seu fatorial: ";
    cin >> num;
    resp = fatorial(num);
    cout << " Resposta: " << resp;
    cout << "\n\n PRESSIONE QUALQUER T E C L A PARA FINALIZAR !!!";
    getch();
    return 0;
}

// função com recursividade
// (Função chama a si própria)
int fatorial(int num){
    // Condição de parada e chamada da função novamente
    if (num <= 1) { // Se num for menor e igual a 1 então...
        return 1; // retorno o valor 1 (Caso base (Trivial).
    } else { // senão...
        // Retorna o num multiplicado
        // pelo fatorial (num-1). E quem é num-1?
        // O antecessor...
        return num * fatorial(num-1); // Caso geral (RECURSIVO).
    }
}
```


Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Programa fatorial

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

// protótipo da função
// num -> Valor no qual será realizado o fatorial
int fatorial(int num);
int main() { // Corpo principal do programa
    setlocale(LC_
```

AO COMPILAR E EXECUTAR O CÓDIGO APARECERÁ:

```
Digite um número para ser calculado o seu fatorial: 4
Resposta: 24
```

```
PRESSIONE QUALQUER T E C L A PARA FINALIZAR !!!
```

```

// função com recursividade
// (Função chama a si própria)
int fatorial(int num){
    // Condição de parada e chamada da função novamente
    if (num <= 1) { // Se num for menor e igual a 1 então...
        return 1; // retorno o valor 1 (Caso base (Trivial).
    } else { // senão...
        // Retorna o num multiplicado
        // pelo fatorial (num-1). E quem é num-1?
        // O antecessor...
        return num * fatorial(num-1); // Caso geral (RECURSIVO).
    }
}
```

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Funções recursivas

- Muitos algoritmos complexos são resolvidos através de soluções recursivas
 - Quick-sort (ordenação rápida);
 - Ordenação por intercalação (merge sort);
 - Busca binária.
- Muitas estruturas de dados são recursivas
 - Listas encadeadas;
 - Árvores binárias, n-árias ...
- Tendem a gerar códigos menores

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Recursão Binária

- Recursão binária ocorre sempre que houver *2 chamadas recursivas* para cada caso não básico.
- Estas chamadas podem, por exemplo, ser usadas para resolver duas metades do mesmo problema.

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Seqüência de Fibonacci

- A seqüência de Fibonacci é a seqüência de inteiros:
 - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
- Cada elemento nessa seqüência é a soma dos dois elementos anteriores. Por exemplo:
 - $0+1=1$; $1+1=2$; $1+2=3$; $2+3=5$...
 - Assume-se que os 2 primeiros elementos são 0 e 1.
- Se partirmos que $\text{fib}(0)=0$ e $\text{fib}(1)=1$ e assim por diante, podemos definir um número da seqüência fibonnaci da seguinte maneira:

$$\begin{cases} \text{fib}(n)=n & \text{se } n=0 \text{ OU } n=1 \text{ (Trivial / Caso base)} \\ \text{fib}(n)=\text{fib}(n-2)+\text{fib}(n-1) & \text{se } n \geq 2 \text{ (Recursivo / Caso geral)} \end{cases}$$

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

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

// protótipo da função
// num -> Valor no qual será realizado o fibonacci
int fibonacci(int num);

int main() { // Corpo principal do programa
    setlocale(LC_ALL, "portuguese");

    int num, resp; // Utilizo variáveis inteiras
    cout << "\n Sequência Fibionacci (Alguns números)";
    cout << "\n\n 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, " <<
        "144, 233, 377, 610, 987, 1597, 2584, 4181,...";
    cout << "\n\n Digite um número de uma determinada posição do fibonacci: ";

    // -----
    cin >> num; // O usuário irá digitar a posição que deseja.
    num = num - 1; // Na programação temos que considerar a posição escolhida diminuindo 1!!!
    // Então ele irá retornar a posição correta !!!
    // -----

    resp = fibonacci(num);
    cout << " Resposta: " << resp;
    cout << "\n\n PRESSIONE QUALQUER <T E C L A> PARA FINALIZAR !!!";
    getch(); return 0; }

// função com recursividade
// (Função chama a si própria)
int fibonacci(int num){
    // Condição de parada e chamada da função novamente
    // Se fib for igual a 0 ou igual 1 então...
    if ((num == 0) || (num == 1)) {
        return num; // Trivial / Caso base.
    } else { // senão...
        // Retorna a soma dos dois antecessores
        // Recursivo (Caso geral)
        return fibonacci(num-1) + fibonacci(num-2); }
}
```

Seqüência de Fibonacci (Implementação)

$$\begin{cases} \text{fib}(n)=n & \text{se } n=0 \text{ OU } n=1 \\ \text{fib}(n)=\text{fib}(n-2)+\text{fib}(n-1) & \text{se } n \geq 2 \end{cases}$$

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

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

// protótipo da função
// num -> Valor no qual será realizado o fibonacci
int fibonacci(int num);
```

```
int main() { // Corpo principal do programa
    setlocale(LC_ALL, "portuguese");
```

```
    int num, resp; // Utilizo variáveis inteiras
```

```
    cout << "\n Sequência de Fibonacci (Alguns números): ";
    cout << "\n\n 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, ... ;
```

Seqüência de Fibonacci (Implementação)

$\text{fib}(n) = n$ se $n = 0$ OU $n = 1$

$\text{fib}(n) = \text{fib}(n-2) + \text{fib}(n-1)$ se $n \geq 2$

AO COMPILAR E EXECUTAR O CÓDIGO APARECERÁ:

Sequência Fibionacci (Alguns números)

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181,...

Digite um número de uma determinada posição do fibonacci: 5

Resposta: 3

PRESSIONE QUALQUER <T E C L A> PARA FINALIZAR !!!

```
// função com recursividade
// (Função chama a si própria)
int fibonacci(int num){
    // Condição de parada e chamada da função novamente
    // Se fib for igual a 0 ou igual 1 então...
    if ((num == 0) || (num == 1)) {
        return num; // Trivial / Caso base.
    } else { // senão...
        // Retorna a soma dos dois antecessores
        // Recursivo (Caso geral)
        return fibonacci(num-1) + fibonacci(num-2); }
}
```


Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Outra manipulação do algoritmo (RECURSIVIDADE)

PARTE 1

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

// protótipo de cada função
float funcao_A(float n);
float funcao_B(float i);
float funcao_C(float c);

int main()
{
    setlocale(LC_ALL, "portuguese");
    float num;
    system("color 0E");
    cout << "\n RECURSIVIDADE (VEJA O CÁLCULO DESSE ALGORITMO !!!):";
    cout << "\n 1. Chama a função A levando o valor de num.";
    cout << "\n 2. Na função A é chamado a função B sendo (n-1).";
    cout << "\n 3. Na função B é chamado a função C sendo (i-3).";
    cout << "\n 4. Na função C faz c/2. (RECURSIVAMENTE)";
    cout << "\n\n Digite um número: ";
    cin >> num;
    cout << "\n\n Resultado: " << fixed << setprecision(2) << funcao_A(num);
}
```

Essa parte é somente
um texto explicando
o funcionamento.

PARTE 2 (CONTINUAÇÃO)

```
float funcao_C(float c)
{
    float cont;
    cout << " " << c << " - ";
    if (c < 1) {
        cout << "FIM !";
        return c;
    } else {
        return funcao_C(c/2);
    }
}

float funcao_B(float i)
{
    float k;
    k=funcao_C(i-3);
    return(k);
}

float funcao_A(float n)
{
    float x;
    x=funcao_B(n-1);
    return(x);
}
```

Aula 4- ALGORITMOS E COMPLEXIDADE

TEMA 2 - RECURSIVIDADE

Outra manipulação do algoritmo (RECURSIVIDADE)

PARTE 1

PARTE 2 (CONTINUAÇÃO)

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

AO COMPILAR E EXECUTAR O CÓDIGO APARECERÁ:

```
float funcao_C(float c)
{
    float cont;
    << " - ";
```

RECURSIVIDADE (VEJA O CÁLCULO DESSE ALGORITMO !!!):

1. Chama a função A levando o valor de num.
2. Na função A é chamado a função B sendo (n-1).
3. Na função B é chamado a função C sendo (i-3).
4. Na função C faz $c/2$. (RECURSIVAMENTE)

Digite um número: 500

496 - 248 - 124 - 62 - 31 - 15.5 - 7.75 - 3.875 - 1.9375 - 0.96875 - FIM !

Resultado: 0.97

Process exited after 2.319 seconds with return value 0

Pressione qualquer tecla para continuar. . .

```
cout << "\n 3. Na função B é chamado a função C sendo (i-3).";
cout << "\n 4. Na função C faz c/2. (RECURSIVAMENTE)";
cout << "\n\n Digite um número: ";
cin >> num;
cout << "\n\n Resultado: " << fixed << setprecision(2) << funcao_A(num);
```

```
float funcao_A(float n)
{
    float x;
    x=funcao_B(n-1);
    return(x);
}
```