



ALGORITMOS E COMPLEXIDADE (ARA0174)

AULA 3

DATA: 03/03/2022



Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1

ANÁLISE DE ALGORITMOS - NOTAÇÃO O



Estácio

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Análise de Algoritmos

- A análise de algoritmos estuda a correção e o desempenho de algoritmos.
- Em outras palavras, a análise de algoritmos procura respostas para perguntas do seguinte tipo:
 - *Este algoritmo resolve o meu problema?*
 - *Quanto tempo o algoritmo consome para processar uma 'entrada' de tamanho n ?*

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Porque analisar a complexidade dos algoritmos?

- *A preocupação com a complexidade de algoritmos é fundamental para projetar algoritmos eficientes.*
- *Podemos desenvolver um algoritmo e depois analisar a sua complexidade para verificar a sua eficiência.*
- *Mas o melhor ainda é ter a preocupação de projetar algoritmos eficientes desde a sua concepção.*

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Análise de Algoritmos

- *Além disso, a análise de algoritmos estuda certos paradigmas (como divisão-e-conquista, programação dinâmica, gula, busca local, aproximação, etc.) que se mostraram úteis na criação de algoritmos para vários problemas computacionais.*
- *A análise de algoritmos foi inventada e difundida por Donald Ervin Knuth (veja a série de livros **The Art of Computer Programming / A Arte da Programação de Computador**). Knuth foi o pai da ideia de prever o tempo de execução e o consumo de espaço de um algoritmo.*

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Análise de Algoritmos

- *Um algoritmo serve para resolver um determinado problema, e todos os problemas têm sempre uma entrada de dados.*
- *O tamanho dessa entrada (n) tem geralmente efeito direto no tempo de resposta de um algoritmo.*

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Análise de Algoritmos (Cont.)

- *Dependendo do problema a ser resolvido, já existem algoritmos prontos ou que podem ser adaptados.*
- *O problema é: Qual algoritmo escolher?*

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Vejamos o exemplo a seguir:

- Sejam 5 algoritmos A_1 a A_5 para resolver um mesmo problema de complexidade diferentes. (Supomos que uma operação leva 1 ms para ser efetuada).
- $T_k(n)$ é a complexidade ou seja o número de operações que o algoritmo efetua para n entradas.

n	A_1 $T_1(n) = n$	A_2 $T_2(n) = n \log n$	A_3 $T_3(n) = n^2$	A_4 $T_4(n) = n^3$	A_5 $T_5(n) = 2^n$
16	0.016s	0.064s	0.256s	4s	1m4s
32	0.032s	0.16s	1s	33s	46 Dias
512	0.512s	9s	4m22s	1 Dia 13h	10^{137} Séculos

tempo necessário para o algoritmo em função de n entradas

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Análise assintótica

- Para valores suficientemente pequenos de n , qualquer algoritmo custa pouco para ser executado, mesmo os algoritmos ineficientes;
- A Análise Assintótica é a análise de algoritmos quando realizada para valores grandes de n considerando-se o devido custo computacional das funções do programa;

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Tipos de Complexidade

Espacial:

- Este tipo de complexidade representa, por exemplo, o **espaço de memória** usado para executar o algoritmo.

Temporal:

- Este tipo de complexidade é o mais usado podendo dividir-se em dois grupos:
 - Tempo (real) necessário à execução do algoritmo **(como podemos medir?)**
 - Número de instruções necessárias à execução.

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Tipos de Complexidade

- Em ambos os casos, a complexidade é medida de acordo com **o tamanho dos dados de entrada (n)**.

Estamos mais interessados em calcular a **Complexidade Temporal** de um algoritmo!.

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Complexidade de Algoritmos

- Existem três perspectivas para análise de complexidade:
 - Melhor Caso;
 - Caso Médio;
 - Pior Caso.
- Nas três perspectivas, a função $f(n)$ retorna a complexidade de um algoritmo com entrada de tamanho n .

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Análise do Melhor Caso

- Definido pela letra grega Ω (Ômega).
- Busca o *menor tempo de execução* de um algoritmo para uma entrada de tamanho n .
- É pouco usado, por ter aplicação em poucos casos.

- EXEMPLO:

O algoritmo de pesquisa sequencial em um vetor tem complexidade:

$$f(n) = \Omega(1)$$

- A análise assume que o número procurado seria o primeiro selecionado na lista.

- Abordagem otimista!.

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Análise do Caso Médio

- Definido pela **letra grega Θ (Theta)**.
- Deve-se obter a *média dos tempos de execução de todas as entradas de tamanho n* , ou baseado em probabilidade de determinada condição ocorrer.

EXEMPLO:

- O algoritmo de pesquisa sequencial em um vetor tem complexidade:

$$f(n) = \Theta(n/2)$$

- Em média será necessário visitar $n/2$ elementos do vetor até encontrar o elemento procurado.
 - Melhor aproximação;
 - Muito difícil de determinar na maioria dos casos.

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Análise do Pior Caso

- Representado pela letra grega O.
- O maiúsculo. Trata-se da letra grega ômicron maiúscula.
- Baseia-se no maior tempo de execução sobre todas as entradas de tamanho n.
- É o método mais fácil de se obter.

EXEMPLO:

- O algoritmo de pesquisa sequencial em um vetor tem complexidade:

$$f(n) = O(n)$$

- No pior caso será necessário visitar todos os n elementos do vetor até encontrar o elemento procurado.
 - Abordagem pessimista!

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Análise do Pior Caso

Ao analisarmos algoritmos devemos analisar o **PIOR CASO** do algoritmo, ou seja:

com o maior número de instruções a ser executado.

Justamente porque isso garante, se for trabalhado uma grande quantidade de dados, sabermos qual o PIOR CASO possível para a execução desse algoritmo.

É necessário estar preparado para este tipo de situação, por não saber a “quantidade de dados” recebida.

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Análise do Pior Caso

Ao analisarmos algoritmos devemos analisar o **PIOR CASO** do algoritmo, ou seja:

com o maior número de instruções a ser executado.

Justamente porque isso garante, se for trabalhado uma grande quantidade de dados, sabermos qual o PIOR CASO possível para a execução desse algoritmo.

É necessário estar preparado para este tipo de situação, por não saber a “quantidade de dados” recebida.

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Análise do Pior Caso

Ao analisarmos algoritmos devemos analisar o **PIOR CASO** do algoritmo, ou seja:

com o maior número de instruções a ser executado.

Se estamos preparados para o **PIOR CASO** saberemos como o nosso algoritmo vai funcionar.

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Análise do Pior Caso

Ao analisarmos algoritmos devemos analisar o **PIOR CASO** do algoritmo, ou seja:

com o maior número de instruções a ser executado.

POR EXEMPLO:

Quantas instruções ele executa para um tamanho n (dados)?

Quando perguntamos isso estamos falando do **CUSTO DO ALGORITMO**.

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Análise do Pior Caso

Aquele **PIOR CASO** possível do CUSTO DO ALGORITMO para todas as entradas de dados do tamanho n .

POR EXEMPLO:

Quantas instruções ele executa para um tamanho n (entrada de dados)?

Quando perguntamos isso estamos falando do CUSTO DO ALGORITMO.

E...

o CUSTO DO ALGORITMO, nesse caso, é o **LIMITE SUPERIOR!**

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Análise do Pior Caso

Aquele **PIOR CASO** possível do CUSTO DO ALGORITMO para todas as entradas de dados do tamanho n .

LIMITE SUPERIOR???

SIM, pois o custo do algoritmo que nós iremos encontrar **NUNCA** vai ultrapassar esse **LIMITE!**.

A NOTAÇÃO O (que trabalharemos na aula), diz que nunca será ultrapassado, o custo do PIOR CASO do algoritmo, desse **LIMITE SUPERIOR !!!**.

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Análise do Pior Caso

Aquele **PIOR CASO** possível do CUSTO DO ALGORITMO para todas as entradas de dados do tamanho n .

LOGO,

CUSTO DO ALGORITMO \leq LIMITE ASSINTÓTICO SUPERIOR

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Análise do Pior Caso

Aquele **PIOR CASO** possível do CUSTO DO ALGORITMO para todas as entradas de dados do tamanho n .

ENTÃO,

- assintoticamente, o custo do algoritmo não é pior do que seu pior caso.
- o custo do algoritmo, no máximo é tão ruim quanto ao limite.
- **não pode ser pior, mas pode ser melhor.**
- (Dependendo como esse conjunto de entradas (dados) chega no algoritmo).

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

A notação O

- Tempo (ou espaço) é contabilizado em número de passos do algoritmo (unidade de armazenamento).
- Análise do algoritmo determina uma função que depende do tamanho da entrada n .

EXEMPLO:

$$10n^3 + 4n - 10$$

- à medida que n aumenta, o termo cúbico começa a dominar.
- A constante do termo cúbico tem relativamente a mesma importância que a velocidade da CPU.

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

A notação O (Análise)

- Desprezar constantes aditivas ou multiplicativas.
Número de passos $3n$ será aproximado para n .
- Interesse assintótico.
 - termos de menor grau podem ser desprezados.

EXEMPLOS:

- 1 $n^2 + n$ (será aproximado para n^2).
- 2 $6n^3 + 4n - 9$ será aproximado para n^3 .

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

A notação O (Análise)

- Sabendo como reduzir as funções, vamos entender como trabalhar funções em termos de **NOTAÇÃO O**.

POR EXEMPLO:

FUNÇÕES REDUZIDAS	PARA ====>	NOTAÇÃO O
$f(n) = 1$		$O(1)$
$f(n) = n$		$O(n)$
$f(n) = n^2$		$O(n^2)$
$f(n) = n^3$		$O(n^3)$

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

A notação O (Análise)

- Sabendo como reduzir as funções, vamos entender como trabalhar funções em termos de **NOTAÇÃO O**.

POR EXEMPLO:

FUNÇÕES REDUZIDAS	PARA ====>	NOTAÇÃO O
$f(n) = 1$	CADA UM DESSES ALGORITMOS NÃO PODE ULTRAPASSAR O LIMITE ESTABELECIDO !	$O(1)$
$f(n) = n$		$O(n)$
$f(n) = n^2$		$O(n^2)$
$f(n) = n^3$		$O(n^3)$

Aula 3- ALGORITMOS E COMPLEXIDADE

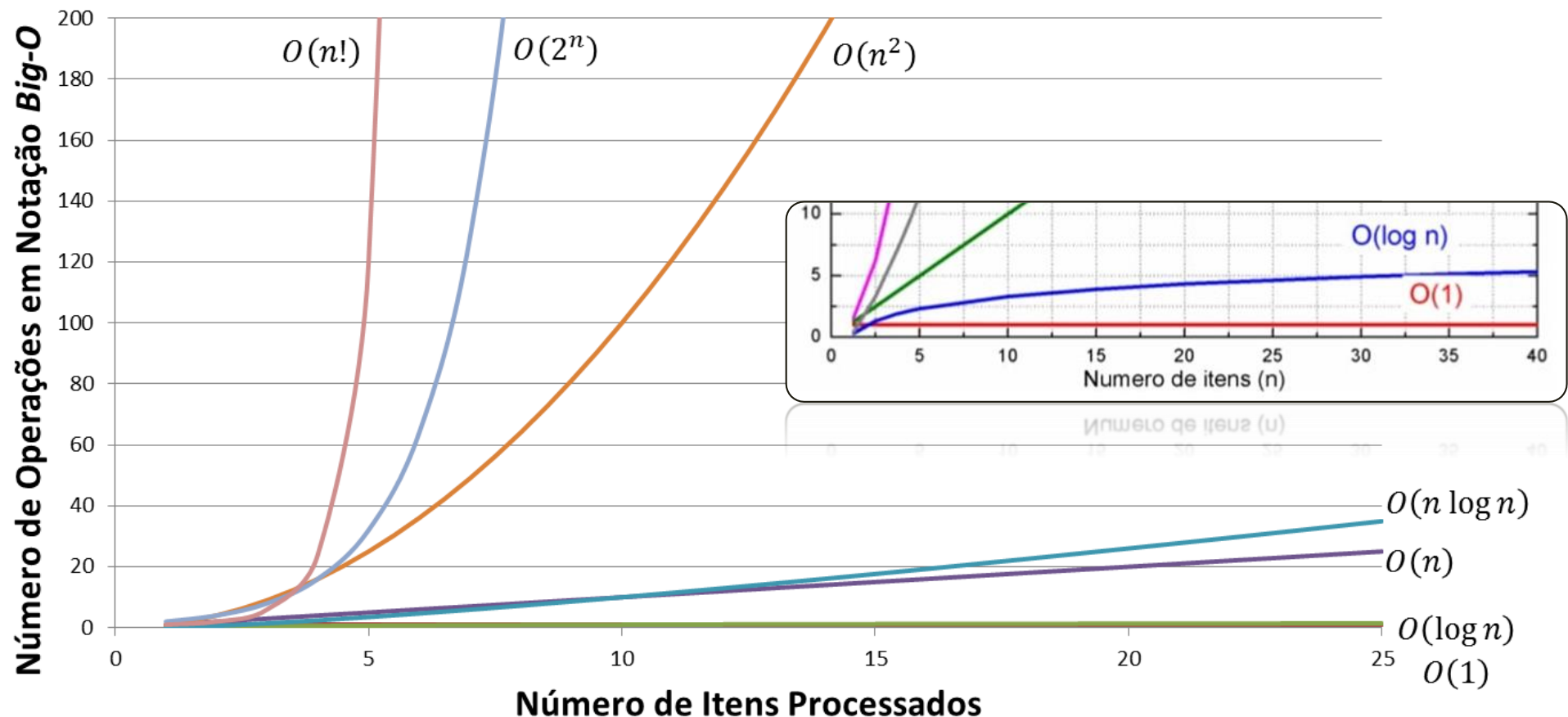
TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Classes de Algoritmos (Análise)

Gráfico de cada complexidade em função de N

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

Ilustração das Complexidades Mais Comuns - Notação *Big-O*



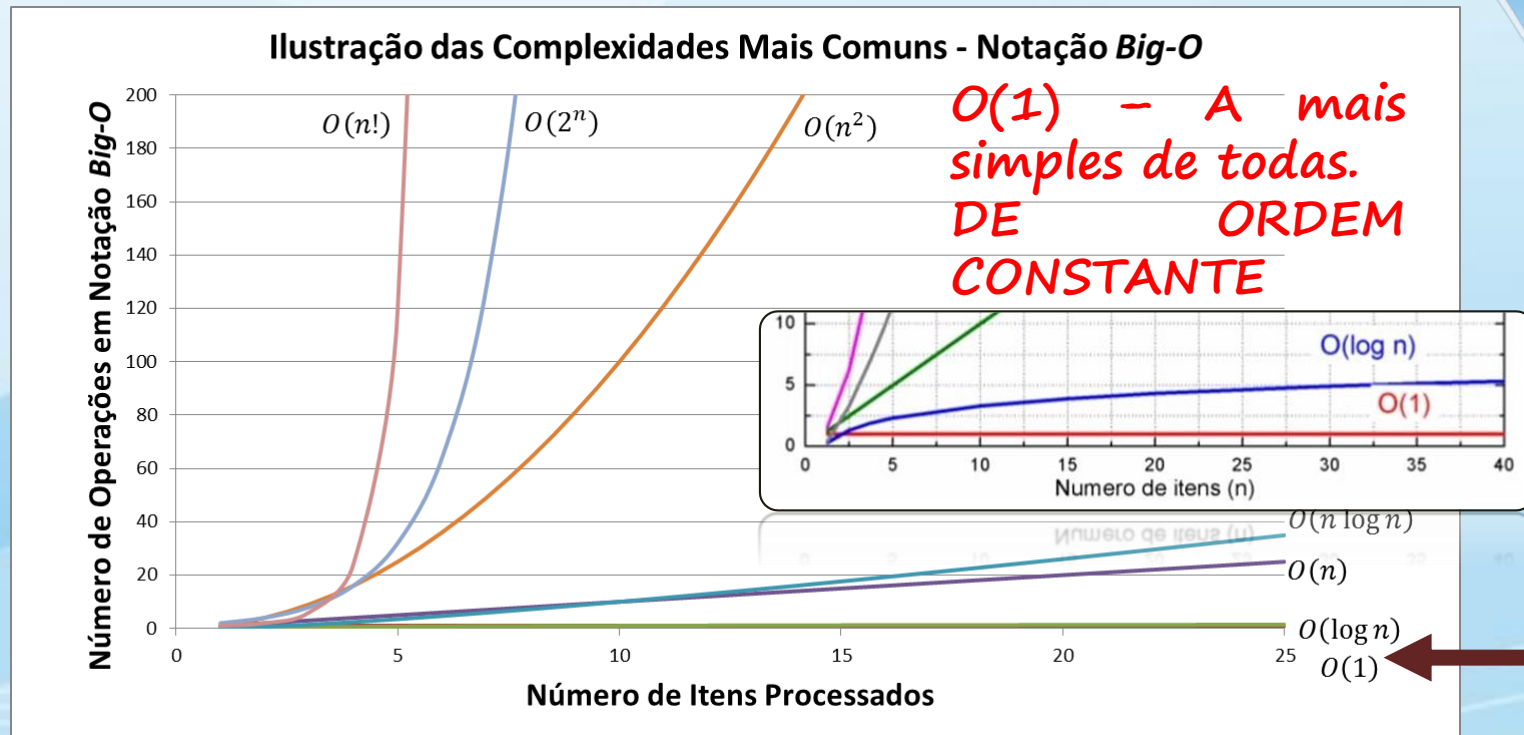
Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Classes de Algoritmos (Análise)

Gráfico de cada complexidade em função de N

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



- Independentemente do tamanho dos dados de entrada, a quantidade de instruções executadas é sempre a mesma (CONSTANTE).

Ou seja, independentemente se o tamanho de entrada n for grande ou pequeno, os algoritmos que fazem parte dessa classe sempre executaram a mesma quantidade de instruções.

Aula 3- ALGORITMOS E COMPLEXIDADE

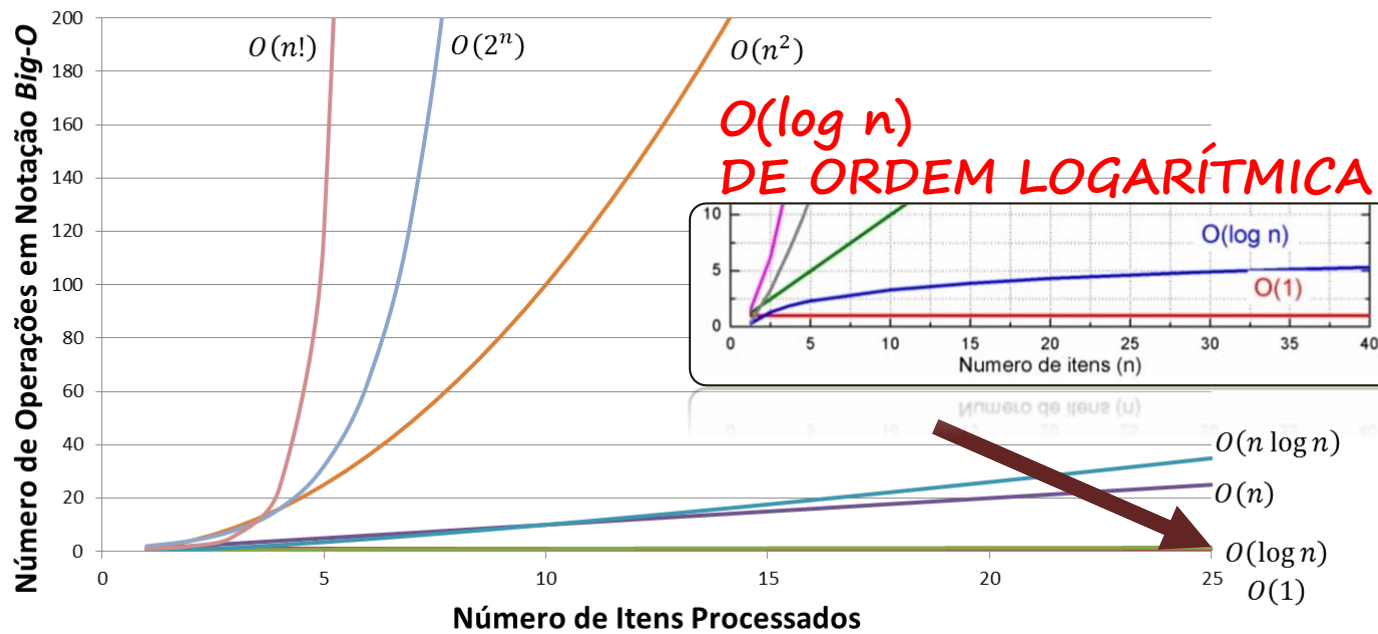
TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Classes de Algoritmos (Análise)

Gráfico de cada complexidade em função de N

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

Ilustração das Complexidades Mais Comuns - Notação Big-O



- Soluções que **dividem** o problema em **problemas menores**, processando a cada iteração, em geral, a metade dos dados da vez anterior.

Ou seja, são algoritmos que vão dividindo o problema (dividindo para conquistar) para resolução desse problemas menores.

SOBRE: Dividir para conquistar

1. **Dividir** o problema em um número de subproblemas que sejam partes menores do mesmo problemas.
2. **Conquistar** os subproblemas resolvendo-os recursivamente. Se eles forem pequenos o suficiente, resolva os subproblemas como problemas base.
3. **Combinar** as soluções dos subproblemas em uma solução para o problema original.

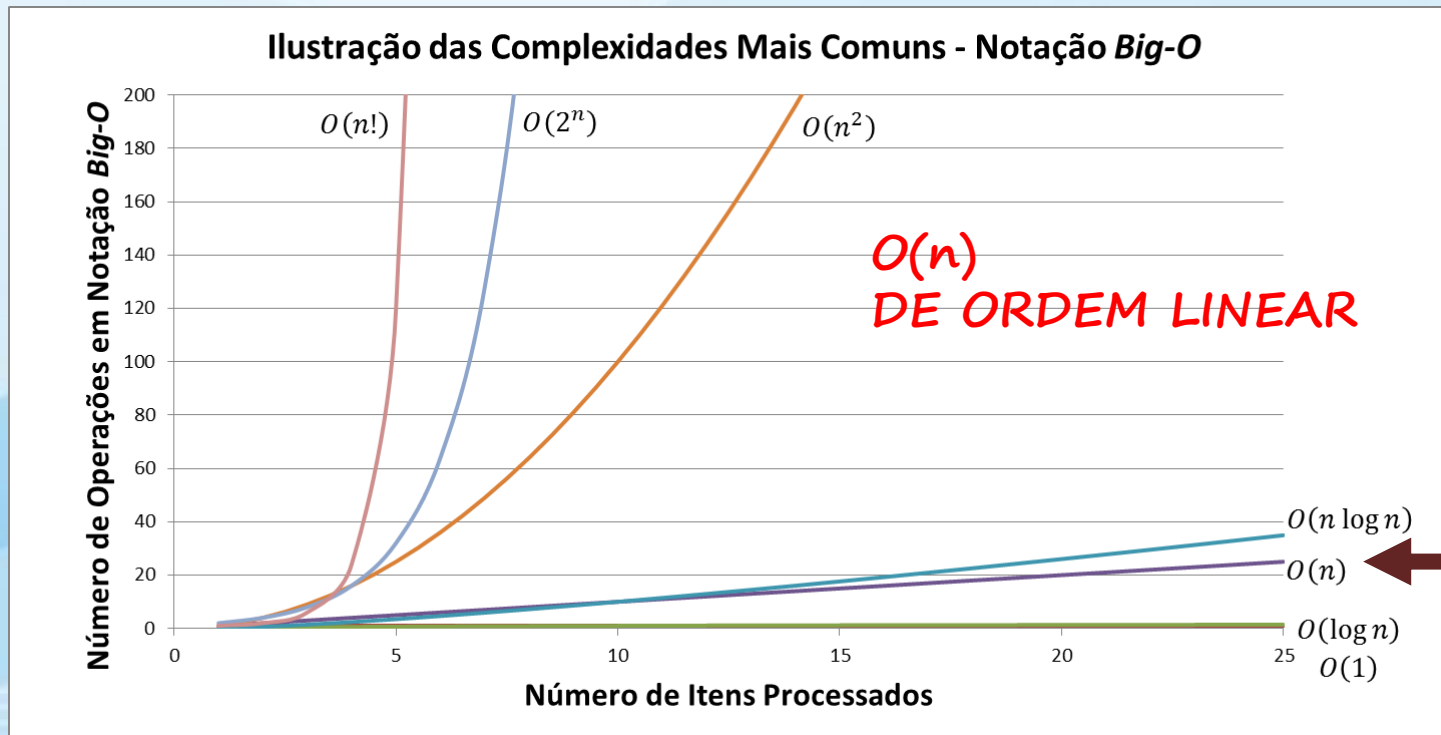
Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Classes de Algoritmos (Análise)

Gráfico de cada complexidade em função de N

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



- O mesmo número de instruções é executado para cada um dos elementos de entrada.

Cresce mais rapidamente que o $O(\log n)$.

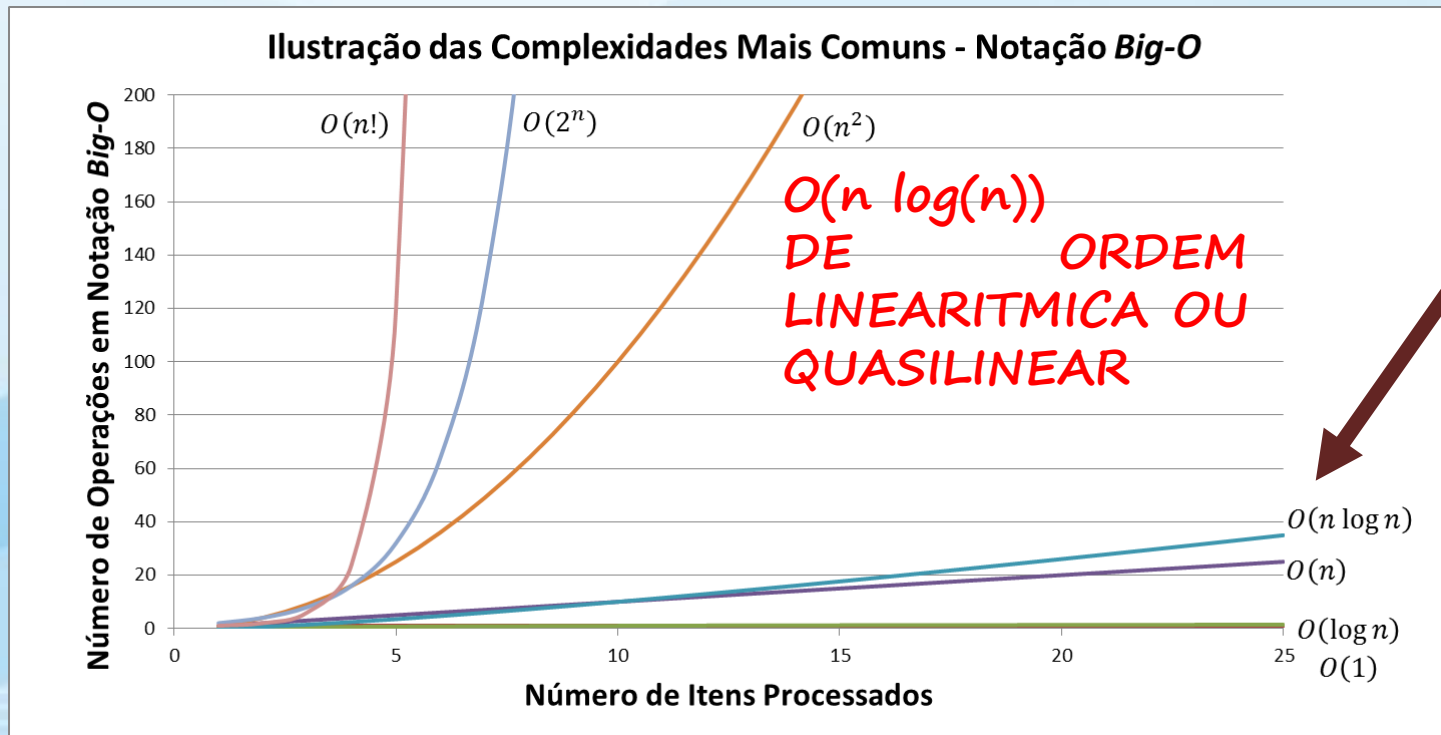
Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Classes de Algoritmos (Análise)

Gráfico de cada complexidade em função de N

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



- é aquela em que o resultado das operações ($\log n$) são executadas n vezes, que por sua vez possuem **soluções independentes**.

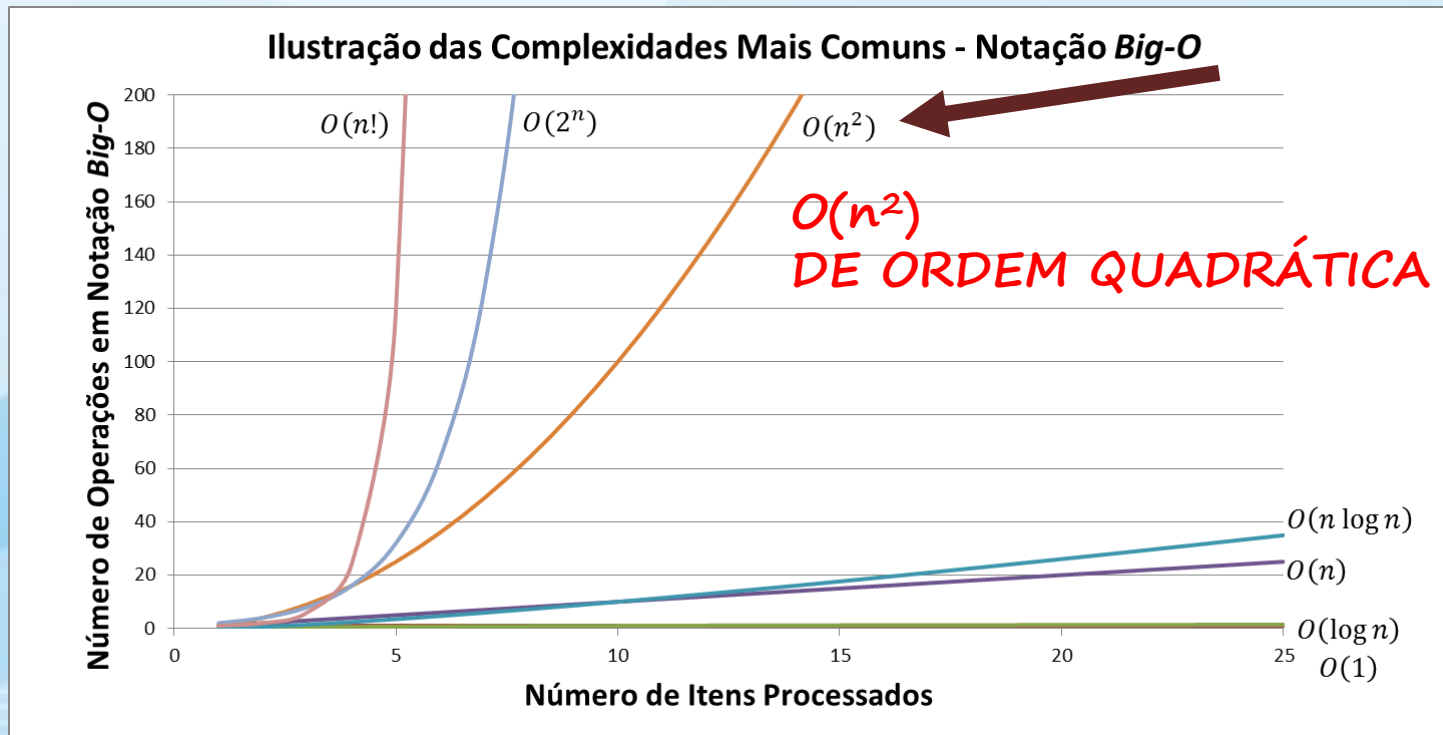
Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Classes de Algoritmos (Análise)

Gráfico de cada complexidade em função de N

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



- São algoritmos que possuem **aninhamentos de laços** para trabalhar com os dados de entrada em pares.

Observação:

Se tivermos por exemplo uma complexidade $O(n^3)$, então estaríamos trabalhando com 3 laços aninhados.

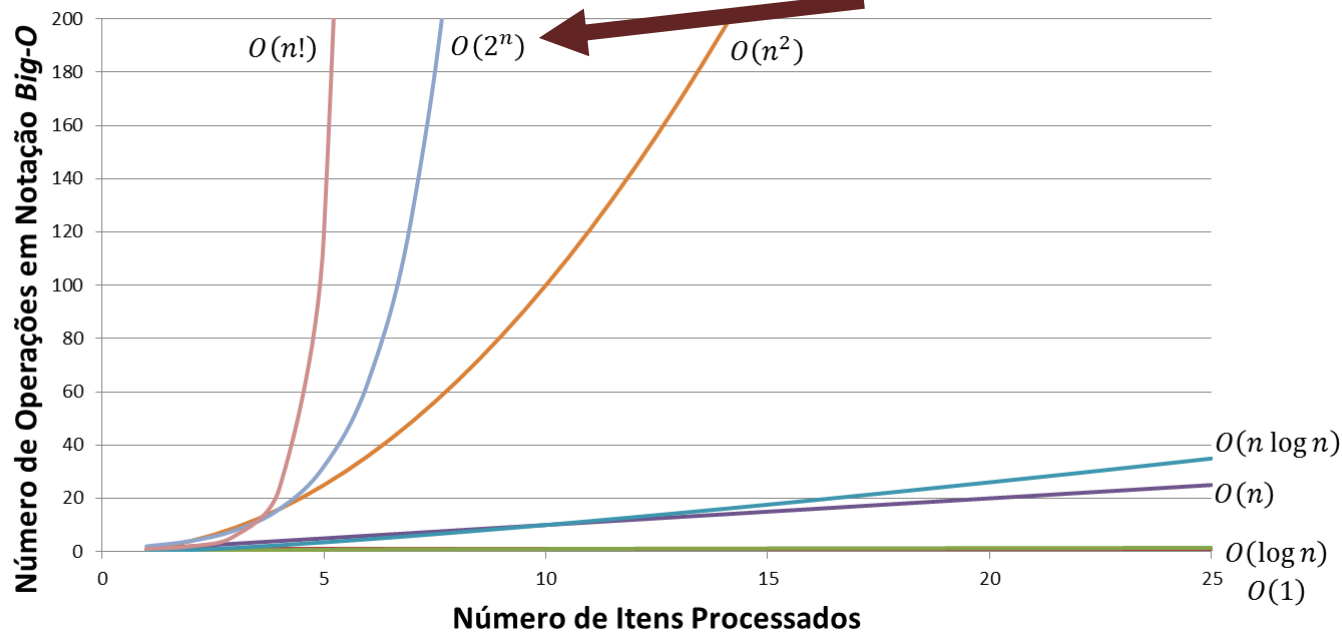
Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Classes de Algoritmos (Análise)

Gráfico de cada complexidade em função de N

Ilustração das Complexidades Mais Comuns - Notação Big-O



$O(2^n)$

DE ORDEM EXPONENCIAL

- São algoritmos de força bruta.

Algoritmo de Força Bruta

Um algoritmo que implementa a solução de um problema através da força bruta é aquele que compara todas as possibilidades possíveis de resposta para o problema e devolve a melhor solução, ou a solução mais correta.

CRESCER MUITO RAPIDAMENTE !!!!

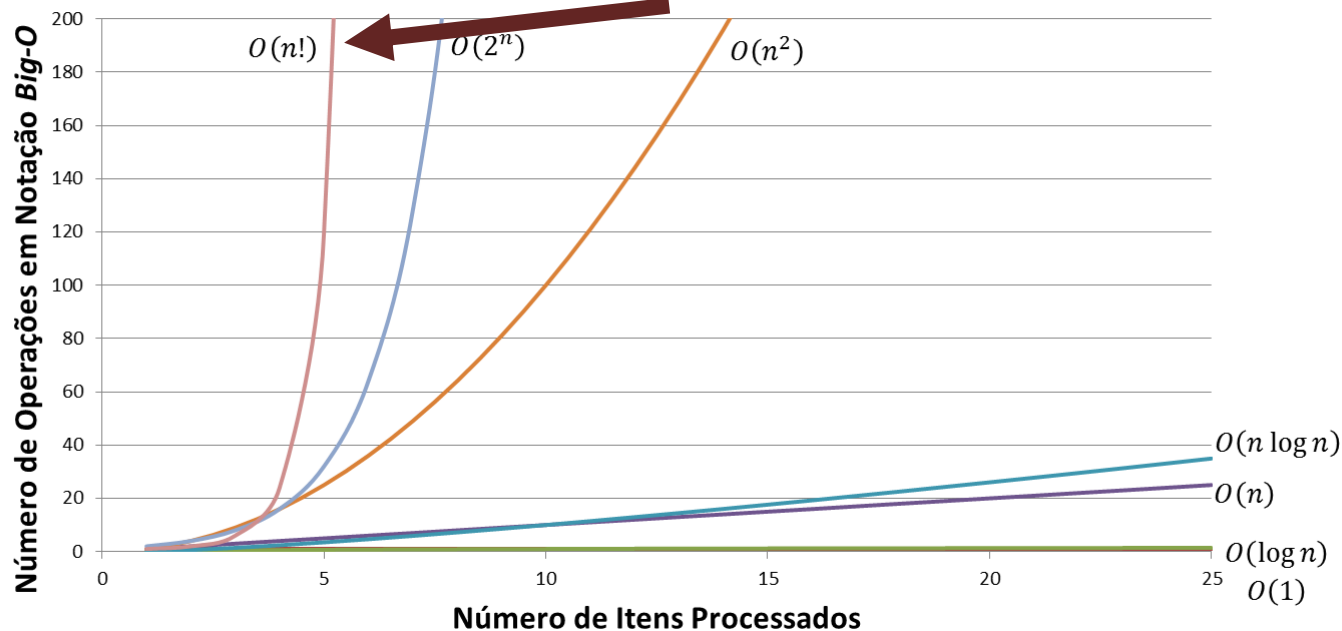
Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Classes de Algoritmos (Análise)

Gráfico de cada complexidade em função de N

Ilustração das Complexidades Mais Comuns - Notação Big-O



$O(n!)$
DE ORDEM FATORIAL

- São algoritmos de força bruta, **pior** que o de EXPONENCIAL !.

Algoritmo de Força Bruta

Um algoritmo que implementa a solução de um problema através da força bruta é aquele que compara todas as possibilidades possíveis de resposta para o problema e devolve a melhor solução, ou a solução mais correta.

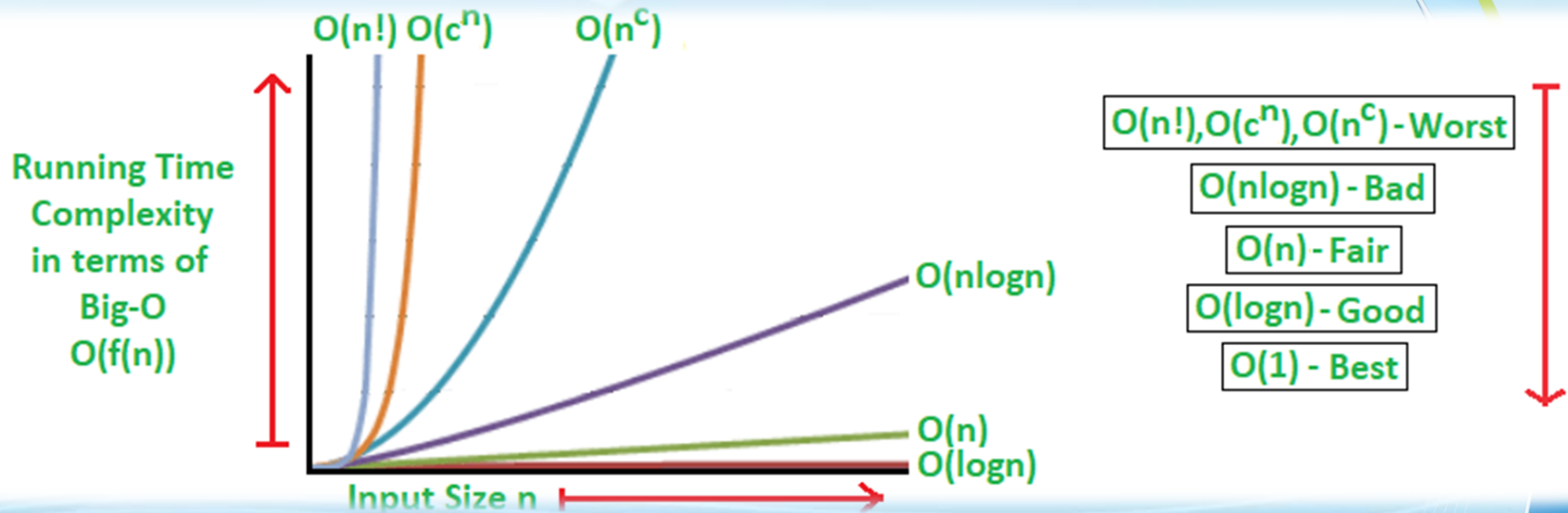
CRESCE EXTREMAMENTE RÁPIDO!!!!

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Classes de Algoritmos (Análise)

Gráfico de cada complexidade em função de N



Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Quando trabalhamos com ALGORITMOS não temos somente uma função ou um comportamento, mas sim vários comportamentos associados !!!.



Então como analisar um ALGORITMO com uma composição de FUNÇÕES ???

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Então como analisar um ALGORITMO com uma composição de FUNÇÕES ???

Como nós analisamos isso? Como analisamos a complexidade assintótica dele?

VAMOS ANALISAR:

$$O(f(n)) + O(g(n))$$

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Então como analisar um ALGORITMO com uma composição de FUNÇÕES ???

Como nós analisamos isso? Como analisamos a complexidade assintótica dele?

VAMOS ANALISAR:

ALGORITMO ①

ALGORITMO ②


$$O(f(n)) + O(g(n))$$

(Estamos somando DUAS complexidades !)

- Se dois algoritmos são executados em sequência a complexidade será estabelecida pela complexidade do MAIOR ALGORITMO.

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Então como analisar um ALGORITMO com uma composição de FUNÇÕES ???

Como nós analisamos isso? Como analisamos a complexidade assintótica dele?

VAMOS ANALISAR:

Se temos $O(f(n)) + O(g(n))$,

O CUSTO TOTAL DA COMPLEXIDADE SERÁ

$= O(\max(f(n), g(n)))$

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Então como analisar um ALGORITMO com uma composição de FUNÇÕES ???

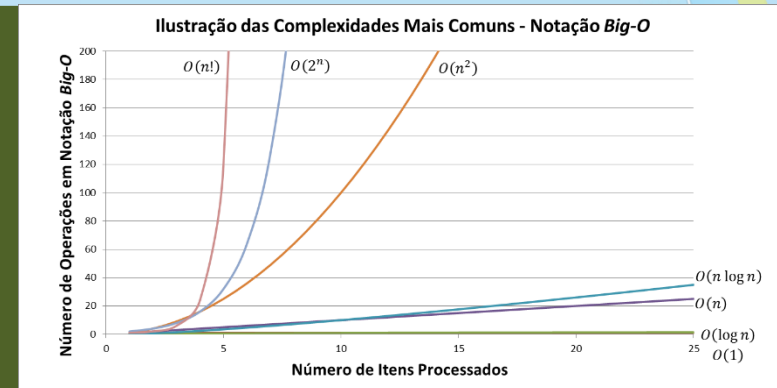
Como nós analisamos isso? Como analisamos a complexidade assintótica dele?

POR EXEMPLO:

Se temos $O(n) + O(n^2)$,

$O(n^2)$ tem a COMPLEXIDADE MAIOR !.

Portanto a soma de $O(n) + O(n^2)$ é $O(n^2)$.



Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Então como analisar um ALGORITMO com uma composição de FUNÇÕES ???

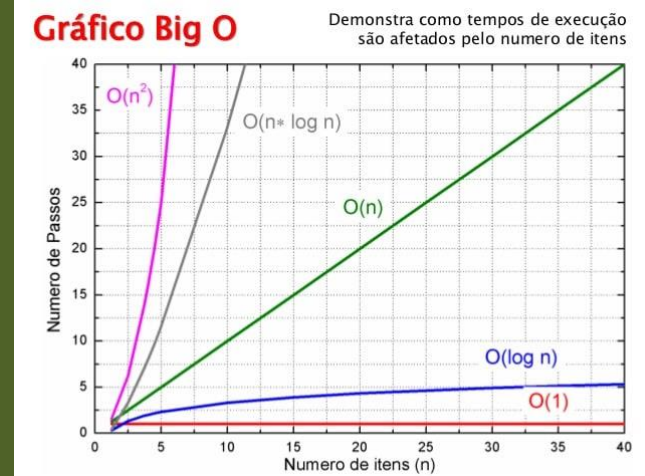
Como nós analisamos isso? Como analisamos a complexidade assintótica dele?

OUTRO EXEMPLO:

Se temos $O(n) + O(n \log n)$,

$O(n \log n)$ tem a COMPLEXIDADE MAIOR !.

Portanto a soma de $O(n) + O(n \log n)$ é $O(n \log n)$.



Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Então como analisar um ALGORITMO com uma composição de FUNÇÕES ???

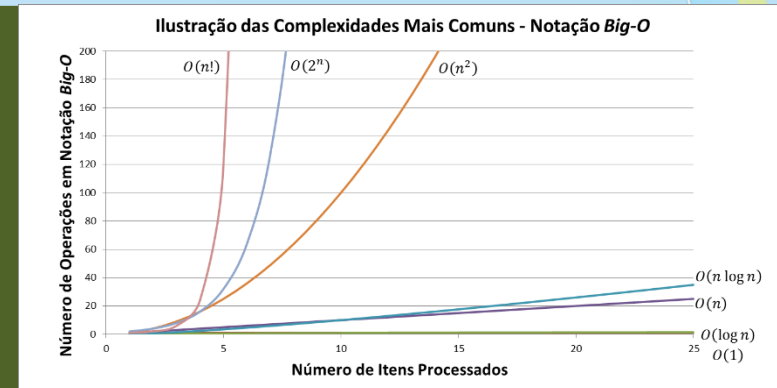
Como nós analisamos isso? Como analisamos a complexidade assintótica dele?

MAIS UM EXEMPLO:

Se temos $O(n^2) + O(n \log n)$,

$O(n^2)$ tem a COMPLEXIDADE MAIOR !.

Portanto a soma de $O(n^2) + O(n \log n)$ é $O(n^2)$.



Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

ANALISANDO O ALGORITMO E TRABALHANDO COM A NOTAÇÃO O:
CONSTRUA O ALGORITMO ABAIXO E ALTERE O VALOR DE n
(INICIANDO EM $1E+0$ ATÉ $1E+9$) E MONTE O GRÁFICO.

LEMBRANDO:

$1e+0 == 1.00 == 1 \times 10^0$

$1e+1 == 10.00 == 1 \times 10^1$

$1e+2 == 100.00 == 1 \times 10^2$

```
#include<iostream>
#include<time.h>
using namespace std;
int main() {
    int n=1E+0; //Vamos iniciar com o valor 1.00.
    // time_t -> O tipo de dado time_t é definido pela biblioteca ISO
    //           da linguagem C para armazenar valores de tempo.
    //           Estes valores são obtidos através da função time().
    time_t inicio, fim;
    time(&inicio); //obtendo o tempo em segundos.
    for (int i=0; i<n;i++) {
        for (int j=0; j<1E+3; j++) {
            time(&fim); //obtendo o tempo em segundos.
        }
    }
    // difftime -> Calcula a diferença em segundos entre o início e o fim.
    cout << "\n\n\nTempo: " << difftime(fim,inicio) << " segundos !";
}
```

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Executando-o por meio de um programa na linguagem C++, em um computador com CPU Intel® Core™ i7, 2.2GHz com 8GB de memória RAM, obtém-se os seguintes tempos de execução:

$n = 1E+0 : \text{tempo} = 0s$

$n = 1E+1 : \text{tempo} = 0s$

$n = 1E+2 : \text{tempo} = 0s$

$n = 1E+3 : \text{tempo} = 0s$

$n = 1E+4 : \text{tempo} = 0s$

$n = 1E+5 : \text{tempo} = 1s$

$n = 1E+6 : \text{tempo} = 2s$

$n = 1E+7 : \text{tempo} = 20s$

$n = 1E+8 : \text{tempo} = 198s$

$n = 1E+9 : \text{tempo} = 2.328s$

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Executando-o por meio de um programa na linguagem C++, em um computador com CPU Intel® Core™ i7, 2.2GHz com 8GB de memória RAM, obtém-se os seguintes tempos de execução:

$n = 1E+0 : \text{tempo} = 0s$

$n = 1E+1 : \text{tempo} = 0s$

$n = 1E+2 : \text{tempo} = 0s$

$n = 1E+3 : \text{tempo} = 0s$

$n = 1E+4 : \text{tempo} = 0s$

$n = 1E+5 : \text{tempo} = 1s$

$n = 1E+6 : \text{tempo} = 2s$

$n = 1E+7 : \text{tempo} = 20s$

$n = 1E+8 : \text{tempo} = 198s$

$n = 1E+9 : \text{tempo} = 2.328s$

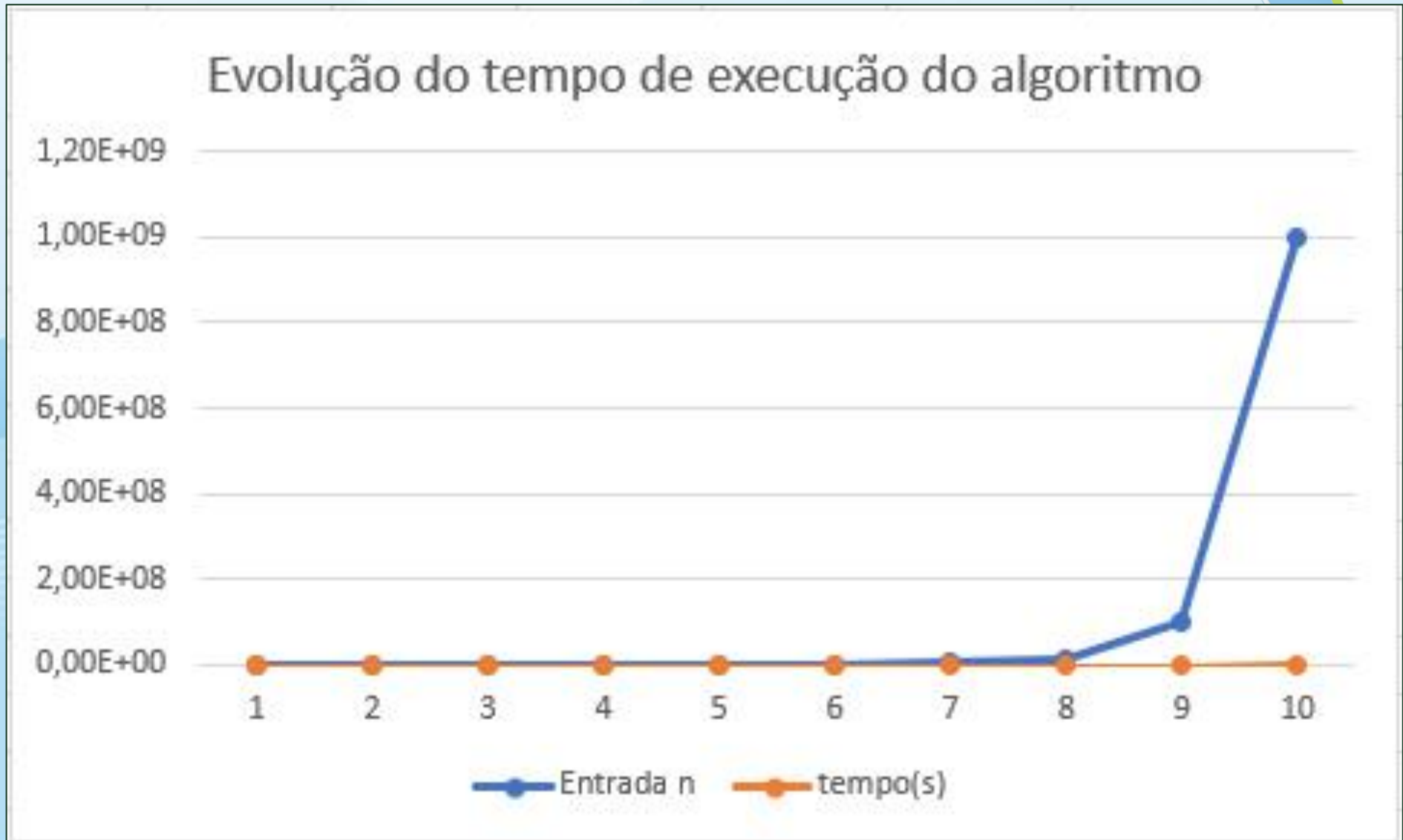
Colocando esses dados no EXCEL e construindo o gráfico teremos...

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

GRÁFICO DO EXERCÍCIO (CONSTRUÍDO NO EXCEL) PROPOSTO PARA O EQUIPAMENTE CITADO:

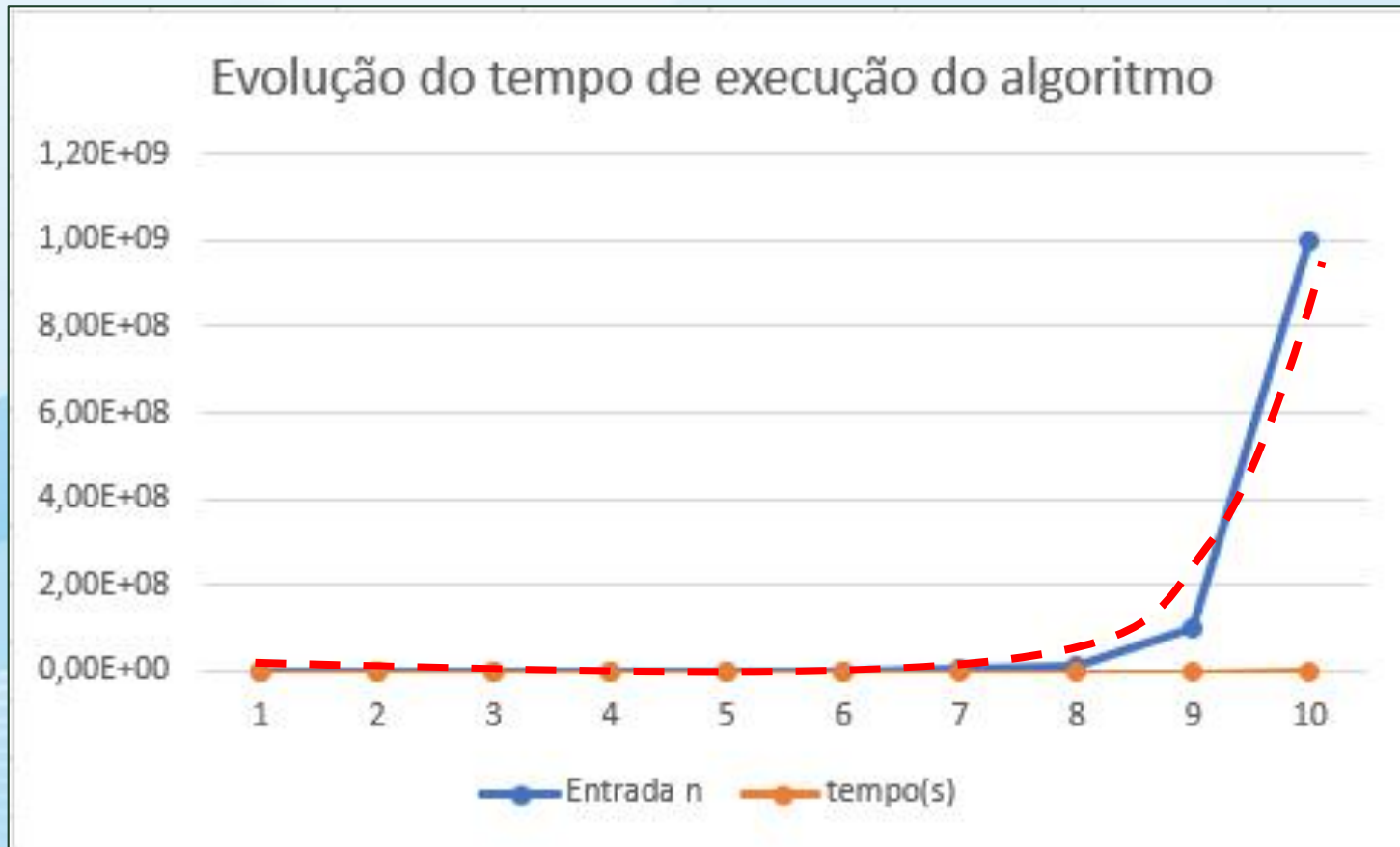
A representação gráfica dos tempos de processamento é como segue:



Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

Apesar dos poucos dados observados de tempo de execução, nota-se, pela taxa de crescimento apresentado, tratar-se de uma função quadrática, ou seja, $f(n) = n^2$:



Isto significa que, qualquer algoritmo que tenha característica de uma função quadrática, terá seus tempos de execução variando conforme o gráfico apresentado acima.

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 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

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 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

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.

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

REFINANDO A ANÁLISE (ALGUMAS PARTICULARIDADES):

Seria interessante analisar o algoritmo levando-se em consideração uma entrada cujo tamanho fosse a média de todas as possíveis entradas. Porém, obter esta quantidade requer cálculos probabilísticos complexos, o que inviabiliza essa abordagem.

Então, leva-se em consideração, na análise do algoritmo, um tamanho de entrada n correspondente ao PIOR CASO, partindo do princípio que, analisando o pior caso, o algoritmo tenha *tempos de execução menor para as demais entradas*.

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

BASICAMENTE, NÓS REALIZAMOS UMA ANÁLISE, DA QUANTIDADE DE "PASSOS" OU INTERAÇÕES QUE O NOSSO CÓDIGO TEM PARA EXECUTAR DO SEU INÍCIO AO SEU FIM.

ISSO CONSIDERANDO O PIOR CASO POSSÍVEL, OU SEJA, O CASO QUE TEORICAMENTE LEVARIA UM MAIOR TEMPO PARA SER EXECUTADO !.

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

PASSOS PARA O CÁLCULO:

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

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 1:

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

bool exemplo1(vector<int> vt, int X) {

    // size() = Função que retorna o número de elementos do vetor.
    int tamanho = vt.size();
    for(int i=0; i<tamanho; i++){
        if(vt[i] == X) {
            return true;
        }
    }
    return false;
}

int main() {
    std::vector<int> vt;
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
    cout << exemplo1(vt,9);
}
```

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 1:

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

Qual a complexidade deste código?



```
bool exemplo1(vector<int> vt, int X) {

    // size() = Função que retorna o número de elementos do vetor.
    int tamanho = vt.size();
    for(int i=0; i<tamanho; i++){
        if(vt[i] == X) {
            return true;
        }
    }
    return false;
}
```

```
int main() {
    std::vector<int> vt;
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
    cout << exemplo1(vt,9);
}
```

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 1:

1º PASSO: ACHAR AS REPETIÇÕES

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

```
bool exemplo1(vector<int> vt, int X) {
```

```
    // size() = Função que retorna o número de elementos do vetor.
```

```
    int tamanho = vt.size();
```

```
    for(int i=0; i<tamanho; i++){
```

```
        if(vt[i] == X) {
```

```
            return true;
```

```
        }
```

```
    }
```

```
    return false;
```

```
}
```

```
int main() {
```

```
    std::vector<int> vt;
```

```
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
```

```
    cout << exemplo1(vt,9);
```

```
}
```

← TEMOS UMA AQUI !!!

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 1:

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

```
bool exemplo1(vector<int> vt, int X) {
```

```
    // size() = Função que retorna o número de elementos do vetor.
```

```
    int tamanho = vt.size();
```

```
    for(int i=0; i<tamanho; i++){
```

```
        if(vt[i] == X) {
```

```
            return true;
```

```
        }
```

```
    }
```

```
    return false;
```

```
}
```

```
int main() {
```

```
    std::vector<int> vt;
```

```
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
```

```
    cout << exemplo1(vt,9);
```

```
}
```

1º PASSO: ACHAR AS REPETIÇÕES

← TEMOS UMA AQUI !!!

Esse "FOR" repete "TAMANHO" vezes. Onde "TAMANHO" é o tamanho do meu vetor de entrada.

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 1:

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

1º PASSO: ACHAR AS REPETIÇÕES

```
bool exemplo1(vector<int> vt, int X) {
```

```
    // size() = Função que retorna o número de elementos do vetor.
```

```
    int tamanho = vt.size();
```

```
    for(int i=0; i<tamanho; i++){
```

```
        if(vt[i] == X) {
```

```
            return true;
```

```
        }
```

```
    }
```

```
    return false;
```

```
}
```

```
int main() {
```

```
    std::vector<int> vt;
```

```
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
```

```
    cout << exemplo1(vt,9);
```

```
}
```

← TEMOS UMA AQUI !!!
TAMANHO é o meu (n de entrada).

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 1:

1º PASSO: ACHAR AS REPETIÇÕES

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

```
bool exemplo1(vector<int> vt, int X) {
```

```
    // size() = Função que retorna o número de elementos do vetor.
```

```
    int tamanho = vt.size();
```

```
    for(int i=0; i<tamanho; i++){
```

```
        if(vt[i] == X) {
```

```
            return true;
```

```
        }
```

```
    }
```

```
    return false;
```

```
}
```

```
int main() {
```

```
    std::vector<int> vt;
```

```
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
```

```
    cout << exemplo1(vt,9);
```

```
}
```

← Logo, esse “FOR” representa:

$O(n)$

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 1:

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

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

```
bool exemplo1(vector<int> vt, int X) {
```

```
    // size() = Função que retorna o número de elementos do vetor.
```

```
    int tamanho = vt.size(); ← Qual a complexidade dessa função?
```

```
    for(int i=0; i<tamanho; i++){ ← O(n)
```

```
        if(vt[i] == X) {
            return true;
        }
    }
```

```
    return false;
```

```
}
```

```
int main() {
```

```
    std::vector<int> vt;
```

```
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
```

```
    cout << exemplo1(vt,9);
```

```
}
```

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 1:

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

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

```
bool exemplo1(vector<int> vt, int X) {
```

```
    // size() = Função que retorna o número de elementos do vetor.
```

```
    int tamanho = vt.size(); ← Qual a complexidade dessa função?
```

```
    for(int i=0; i<tamanho; i++){ ← O(n)
```

```
        if(vt[i] == X) {
            return true;
        }
    }
```

```
    return false;
```

```
}
```

```
int main() {
```

```
    std::vector<int> vt;
```

```
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
```

```
    cout << exemplo1(vt,9);
```

```
}
```

Essa função size() é do C++, vamos pesquisar no site "www.cplusplus.com" e procurar a complexidade dessa função.

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 1:

The screenshot shows the homepage of cplusplus.com. At the top, there is a search bar with the text "size()" and a "Go" button. To the right of the search bar, it says "Not logged in" with "register" and "log in" buttons. The main header features the "Welcome to cplusplus.com" text and a copyright notice "© The C++ Resources Network, 2020". The page is divided into several sections: "Information" (General information about the C++ programming language, including non-technical documents and descriptions), "Tutorials" (Learn the C++ language from its basics up to its most advanced features), "Reference" (Description of the most important classes, functions and objects of the Standard Language Library, with descriptive fully-functional short programs as examples), "Articles" (User-contributed articles, organized into different categories), "Forum" (Message boards where members can exchange knowledge), and "C++ Search" (Search this website:). A left sidebar contains links to "C++", "Information", "Tutorials", "Reference", "Articles", and "Forum".

Information

General information about the C++ programming language, including non-technical documents and descriptions:

- Description of the C++ language
- History of the C++ language
- F.A.Q., Frequently Asked Questions

Tutorials

Learn the C++ language from its basics up to its most advanced features.

- 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

Description of the most important classes, functions and objects of the Standard Language Library, with descriptive fully-functional short programs as examples:

- 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...

Articles

User-contributed articles, organized into different categories:

- Algorithms
- Standard library
- C++11
- Windows API
- Other...

You can contribute your own articles!

Forum

Message boards where members can exchange knowledge

C++ Search

Search this website:

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 1:

The screenshot shows the cplusplus.com website interface. At the top, there is a search bar with the text "Search:" and a "Go" button. To the right of the search bar, it says "Not logged in" with links for "register" and "log in". Below the search bar, there is a navigation menu with links for "home", "C++", "Information", "Tutorials", "Reference", "Articles", and "Forum". The main content area is titled "C++ Search - 'size()'" and shows search results for the term "size()". The results are categorized by "All", "Reference", "Articles", and "Forum". The first result is "string::size - C++ Reference" and is highlighted with a red dashed border. The second result is "vector::size - C++ Reference". The third result is "deque::size - C++ Reference". The fourth result is "stack::size - C++ Reference".

C++ Search - "size()"

About 31,600 results (0.30 seconds)

string::size - C++ Reference
www.cplusplus.com › reference › string › string › size
size_t size() const; ... #include <string> int main () { std::string str ("Test string"); std::cout << "The size of str is " << str.size() << " bytes."
Labeled [Reference](#)

vector::size - C++ Reference
www.cplusplus.com › reference › vector › vector › size
vector::size #include <iostream> #include <vector> int main () { std::vector< int > myints; std::cout << "0. size: " << myints.size() << " "; for (int ...
Labeled [Reference](#)

deque::size - C++ Reference
www.cplusplus.com › reference › deque › deque › size
deque::size #include <iostream> #include <deque> int main () { std::deque< int > myints; std::cout << "0. size: " << myints.size() << " "; for (int i=0; ...
Labeled [Reference](#)

stack::size - C++ Reference
www.cplusplus.com › reference › stack › stack › size
std::stack::size. size_type size() const; Return size. Returns the number of elements in the stack. This member ...
Labeled [Reference](#)

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 1:

The screenshot shows a C++ IDE with the following components:

- Left Panel (Class Hierarchy):** Displays the hierarchy for `<string>`. Under **class templates:**, it lists `basic_string` and `char_traits`. Under **classes:**, it lists `string`, `u16string`, `u32string`, and `wstring`. Under **functions:**, it lists various string manipulation functions like `stod`, `stof`, `stoi`, etc.
- Top Right Panel (Code Editor):** Contains a C++ program:

```
1 // Example: string\n2 #include <string>\n3\n4\n5 int main ()\n6 {\n7     std::string str ("Test string");\n8     std::cout << "The size of str is " << str.size() << " bytes.\\n";\n9     return 0;\n10 }
```
- Output Panel:** Shows the result of running the program: "The size of str is 11 bytes".
- Complexity Panel:** A red dashed box highlights this section. It shows the complexity for `string` in C++11 as **Constant.** A yellow highlight and an arrow point to this text with the note: "Essa função tem complexidade constante !!!".
- Iterator validity Panel:** Shows "No changes."
- Data races Panel:** Shows "The object is accessed."
- Exception safety Panel:** Shows "No-throw guarantee: this member function never throws exceptions."
- See also Panel:** Lists related functions like `string::length`, `string::resize`, `string::max_size`, and `string::capacity`.

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 1:

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

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

```
bool exemplo1(vector<int> vt, int X) {
```

```
    // size() = Função que retorna o número de elementos do vetor.
```

```
    int tamanho = vt.size(); ← O(1)
    for(int i=0; i<tamanho; i++){ ← O(n)
```

```
        if(vt[i] == X) {
            return true;
        }
```

```
    return false;
```

```
}
```

```
int main() {
```

```
    std::vector<int> vt;
```

```
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
```

```
    cout << exemplo1(vt,9);
```

```
}
```

→ ESSA COMPARAÇÃO TAMBÉM É CONSIDERADA COMO UMA CONSTANTE, LOGO TERÁ COMPLEXIDADE O(1).

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 1:

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

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

bool exemplo1(vector<int> vt, int X) {

    // size() = Função que retorna o número de elementos do vetor.
    int tamanho = vt.size(); ← O(1)
    for(int i=0; i<tamanho; i++){ ← O(n)
        if(vt[i] == X) { ← O(1)
            return true;
        }
    }
    return false;
}

int main() {
    std::vector<int> vt;
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
    cout << exemplo1(vt,9);
}
```


Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 1:

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

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

bool exemplo1(vector<int> vt, int X) {

    // size() = Função que retorna o número de elementos do vetor.
    int tamanho = vt.size(); ← O(1)
    for(int i=0; i<tamanho; i++){ ← O(n)
        if(vt[i] == X) { ← O(1)
            return true; ← O(1)
        }
    }
    return false;
}

int main() {
    std::vector<int> vt;
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
    cout << exemplo1(vt,9);
}
```

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 1:

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

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

bool exemplo1(vector<int> vt, int X) {

    // size() = Função que retorna o número de elementos do vetor.
    int tamanho = vt.size(); ← O(1)
    for(int i=0; i<tamanho; i++){ ← O(n)
        if(vt[i] == X) { ← O(1)
            return true; ← O(1)
        }
    }
    return false; ← O(1)
}

int main() {
    std::vector<int> vt;
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
    cout << exemplo1(vt,9);
}
```

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 1:

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

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 1:

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

```
bool exemplo1(vector<int> vt, int X) {
```

```
    // size() = Função que retorna o número de elementos do vetor.
```

```
    int tamanho = vt.size();
```

```
    for(int i=0; i<tamanho; i++){ ← O(n)
```

```
        if(vt[i] == X) {
```

```
            return true;
```

```
        }
```

```
    }
    return false;
```

```
}
```

```
int main() {
```

```
    std::vector<int> vt;
```

```
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
```

```
    cout << exemplo1(vt,9);
```

```
}
```

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

RESPOSTA: A COMPLEXIDADE DE MAIOR GRAU É O(n), OU SEJA DE ORDEM LINEAR.

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 2:

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

bool exemplo2(vector<int> vt) {

    // size() = Função que retorna o número de elementos do vetor.
    int tamanho = vt.size();
    for(int i=0; i<tamanho; i++){
        for(int j=0; j<tamanho; j++) {
            if( i!=j && vt[i]==vt[j]) {
                return true;
            }
        }
    }
    return false;
}

int main() {
    std::vector<int> vt;
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
    cout << exemplo2(vt);
}
```

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 2:

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

Qual a complexidade deste código?



```
bool exemplo2(vector<int> vt) {
    // size() = Função que retorna o número de elementos do vetor.
    int tamanho = vt.size();
    for(int i=0; i<tamanho; i++){
        for(int j=0; j<tamanho; j++) {
            if( i!=j && vt[i]==vt[j]) {
                return true;
            }
        }
    }
    return false;
}
```

```
int main() {
    std::vector<int> vt;
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
    cout << exemplo2(vt);
}
```

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 2:

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

1º PASSO: ACHAR AS REPETIÇÕES

```
bool exemplo2(vector<int> vt) {
    // size() = Função que retorna o número de elementos do vetor.
    int tamanho = vt.size();
    for(int i=0; i<tamanho; i++){ ← O(n)
        for(int j=0; j<tamanho; j++) {
            if( i!=j && vt[i]==vt[j]) {
                return true;
            }
        }
    }
    return false;
}

int main() {
    std::vector<int> vt;
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
    cout << exemplo2(vt);
}
```


Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 2:

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

1º PASSO: ACHAR AS REPETIÇÕES

```
bool exemplo2(vector<int> vt) {
    // size() = Função que retorna o número de elementos do vetor.
    int tamanho = vt.size();
    for(int i=0; i<tamanho; i++){
        for(int j=0; j<tamanho; j++) {
            if( i!=j && vt[i]==vt[j]) {
                return true;
            }
        }
    }
    return false;
}

int main() {
    std::vector<int> vt;
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
    cout << exemplo2(vt);
}
```

$O(n)$

$O(n)$

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 2:

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

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

```
bool exemplo2(vector<int> vt) {
```

```
    // size() = Função que retorna o número de elementos do vetor.
```

```
    int tamanho = vt.size();
```

```
    for(int i=0; i<tamanho; i++){
```

```
        for(int j=0; j<tamanho; j++) {
```

```
            if( i!=j && vt[i]==vt[j]) {
```

```
                return true;
```

```
            }
```

```
        }
```

```
    }
```

```
    return false;
```

```
}
```

```
int main() {
```

```
    std::vector<int> vt;
```

```
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
```

```
    cout << exemplo2(vt);
```

```
}
```

← $O(1)$

← $O(n)$

← $O(n)$

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 2:

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

bool exemplo2(vector<int> vt) {

    // size() = Função que retorna o número de elementos do vetor.
    int tamanho = vt.size(); ← O(1)
    for(int i=0; i<tamanho; i++){ ← O(n)
        for(int j=0; j<tamanho; j++){ ← O(n)
            if( i!=j && vt[i]==vt[j]) { ← O(1)
                return true;
            }
        }
    }
    return false;
}

int main() {
    std::vector<int> vt;
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
    cout << exemplo2(vt);
}
```

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 2:

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

bool exemplo2(vector<int> vt) {

    // size() = Função que retorna o número de elementos do vetor.
    int tamanho = vt.size(); ← O(1)
    for(int i=0; i<tamanho; i++){ ← O(n)
        for(int j=0; j<tamanho; j++){ ← O(n)
            if( i!=j && vt[i]==vt[j]) { ← O(1)
                return true; ← O(1)
            }
        }
    }
    return false;
}

int main() {
    std::vector<int> vt;
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
    cout << exemplo2(vt);
}
```

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 2:

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

bool exemplo2(vector<int> vt) {

    // size() = Função que retorna o número de elementos do vetor.
    int tamanho = vt.size(); ← O(1)
    for(int i=0; i<tamanho; i++){ ← O(n)
        for(int j=0; j<tamanho; j++){ ← O(n)
            if( i!=j && vt[i]==vt[j]) { ← O(1)
                return true; ← O(1)
            }
        }
    }
    return false; ← O(1)
}

int main() {
    std::vector<int> vt;
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
    cout << exemplo2(vt);
}
```

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 2:

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

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 2:

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

bool exemplo2(vector<int> vt) {

    // size() = Função que retorna o número de elementos do vetor.
    int tamanho = vt.size();
    for(int i=0; i<tamanho; i++){
        for(int j=0; j<tamanho; j++) {
            if( i!=j && vt[i]==vt[j]) {
                return true;
            }
        }
    }
    return false;
}
```

$O(n)$

$O(n)$

SE EU TENHO UM "FOR"
DENTRO DE UM OUTRO "FOR"
ENTÃO EU MULTIPLICO.

```
int main() {
    std::vector<int> vt;
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
    cout << exemplo2(vt);
}
```



Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 2:

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

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

```
bool exemplo2(vector<int> vt) {
```

```
    // size() = Função que retorna o número de elementos do vetor.
```

```
    int tamanho = vt.size();
```

```
    for(int i=0; i<tamanho; i++){
```

```
        for(int j=0; j<tamanho; j++) {
```

```
            if( i!=j && vt[i]==vt[j]) {
```

```
                return true;
```

```
            }
```

```
        }
```

```
    }
```

```
    return false;
```

```
}
```

```
int main() {
```

```
    std::vector<int> vt;
```

```
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
```

```
    cout << exemplo2(vt);
```

```
}
```

← $O(n)$ ← $O(n)$

RESPOSTA: LOGO A COMPLEXIDADE DO CÓDIGO SERÁ $O(n) * O(n)$ QUE É IGUAL A $O(n^2)$, OU SEJA DE ORDEM QUADRÁTICA.

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 3:

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

PARTE 1

```
int exemplo3(vector<int> vt) {
    // size() = Função que retorna o número de elementos do vetor.
    int tamanho = vt.size();
    int mnp_1 = 0;
    for(int i=0; i<tamanho; i++){
        for(int j=0; j<tamanho; j++) {
            if( vt[i] == vt[j]) {
                mnp_1++;
            }
        }
    }

    int mnp_2 = 0;
    for(int i=0; i<tamanho; i++){
        if( vt[i] == 10) {
            mnp_1 = 5*mnp_1;
        }
    }
}
```

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 3:

PARTE 2

```
int mnp_3 = 0;
for(int i=0; i<tamanho; i++){
    if( vt[i] == 5) {
        mnp_3 += 50;
    }
}

return mnp_1+mnp_2+mnp_3;
}

int main() {
    std::vector<int> vt;
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
    cout << exemplo3(vt); }
```

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 3:

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

Qual a complexidade deste código?



PARTE 1

```
int exemplo3(vector<int> vt) {
```

```
    // size() = Função que retorna o número de elementos do vetor.
```

```
    int tamanho = vt.size();
```

```
    int mnp_1 = 0;
```

```
    for(int i=0; i<tamanho; i++){
```

```
        for(int j=0; j<tamanho; j++) {
```

```
            if( vt[i] == vt[j]) {
```

```
                mnp_1++;
```

```
            }
```

```
        }
```

```
    }
```

```
    int mnp_2 = 0;
```

```
    for(int i=0; i<tamanho; i++){
```

```
        if( vt[i] == 10) {
```

```
            mnp_1 = 5*mnp_1;
```

```
        }
```

```
    }
```

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 3:

Qual a complexidade deste código?

PARTE 2

```
int mnp_3 = 0;
for(int i=0; i<tamanho; i++){
    if( vt[i] == 5) {
        mnp_3 += 50;
    }
}

return mnp_1+mnp_2+mnp_3;
}
```

```
int main() {
    std::vector<int> vt;
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
    cout << exemplo3(vt); }
```


Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 3:

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

1º PASSO: ACHAR AS REPETIÇÕES

PARTE 1

```
int exemplo3(vector<int> vt) {
    // size() = Função que retorna o número de elementos do vetor.
    int tamanho = vt.size();
    int mnp_1 = 0;
    for(int i=0; i<tamanho; i++){ ← O(n)
        for(int j=0; j<tamanho; j++) {
            if( vt[i] == vt[j]) {
                mnp_1++;
            }
        }
    }

    int mnp_2 = 0;
    for(int i=0; i<tamanho; i++){
        if( vt[i] == 10) {
            mnp_1 = 5*mnp_1;
        }
    }
}
```


Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 3:

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

1º PASSO: ACHAR AS REPETIÇÕES

PARTE 1

```
int exemplo3(vector<int> vt) {
    // size() = Função que retorna o número de elementos do vetor.
    int tamanho = vt.size();
    int mnp_1 = 0;
    for(int i=0; i<tamanho; i++){
        for(int j=0; j<tamanho; j++){
            if( vt[i] == vt[j]) {
                mnp_1++;
            }
        }
    }

    int mnp_2 = 0;
    for(int i=0; i<tamanho; i++){
        if( vt[i] == 10) {
            mnp_1 = 5*mnp_1;
        }
    }
}
```

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

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 3:

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

1º PASSO: ACHAR AS REPETIÇÕES

PARTE 1

```
int exemplo3(vector<int> vt) {
    // size() = Função que retorna o número de elementos do vetor.
    int tamanho = vt.size();
    int mnp_1 = 0;
    for(int i=0; i<tamanho; i++){
        for(int j=0; j<tamanho; j++){
            if( vt[i] == vt[j]) {
                mnp_1++;
            }
        }
    }
}
```

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

```
int mnp_2 = 0;
for(int i=0; i<tamanho; i++){
    if( vt[i] == 10) {
        mnp_1 = 5*mnp_1;
    }
}
```

$O(n)$

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 3:

1º PASSO: ACHAR AS REPETIÇÕES

PARTE 2

```
int mnp_3 = 0;
for(int i=0; i<tamanho; i++){ ← O(n)
    if( vt[i] == 5) {
        mnp_3 += 50;
    }
}

return mnp_1+mnp_2+mnp_3;
}

int main() {
    std::vector<int> vt;
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
    cout << exemplo3(vt); }
```

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 3:

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

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

PARTE 1

```
int exemplo3(vector<int> vt) {
    // size() = Função que retorna o número de elementos do vetor.
    int tamanho = vt.size(); ← O(1)
    int mnp_1 = 0;
    for(int i=0; i<tamanho; i++){ ← O(n)
        for(int j=0; j<tamanho; j++){ ← O(n)
            if( vt[i] == vt[j]) {
                mnp_1++;
            }
        }
    }

    int mnp_2 = 0;
    for(int i=0; i<tamanho; i++){
        if( vt[i] == 10) {
            mnp_1 = 5*mnp_1;
        }
    }
}
```


Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 3:

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

PARTE 1

```
int exemplo3(vector<int> vt) {
    // size() = Função que retorna o número de elementos do vetor.
    int tamanho = vt.size(); ← O(1)
    int mnp_1 = 0; ← O(1)
    for(int i=0; i<tamanho; i++){ ← O(n)
        for(int j=0; j<tamanho; j++){ ← O(n)
            if( vt[i] == vt[j]) { ← O(1)
                mnp_1++; ← O(1)
            }
        }
    }
}
```

```
int mnp_2 = 0; ← O(1)
for(int i=0; i<tamanho; i++){ ← O(n)
    if( vt[i] == 10) { ← O(1)
        mnp_1 = 5*mnp_1; ← O(1)
    }
}
```


Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 3:

PARTE 2

```
int mnp_3 = 0; ← O(1)
for(int i=0; i<tamanho; i++){ ← O(n)
    if( vt[i] == 5) { ← O(1)
        mnp_3 += 50; ← O(1)
    }
}

return mnp_1+mnp_2+mnp_3; ← O(1)
}

int main() {
    std::vector<int> vt;
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
    cout << exemplo3(vt); }
```

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO:

EXEMPLO 3:

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

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 3:

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

PARTE 1

```
int exemplo3(vector<int> vt) {
    // size() = Função que retorna o número de elementos do vetor.
    int tamanho = vt.size();
    int mnp_1 = 0;
    for(int i=0; i<tamanho; i++){
        for(int j=0; j<tamanho; j++){
            if( vt[i] == vt[j]) {
                mnp_1++;
            }
        }
    }
```

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

```
    int mnp_2 = 0;
    for(int i=0; i<tamanho; i++){
        if( vt[i] == 10) {
            mnp_1 = 5*mnp_1;
        }
    }
```

$O(n)$

NESSA PARTE DO CÓDIGO
TEMOS: $O(n)*O(n) + O(n)$.

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 3:

PARTE 2

```
int mnp_3 = 0;
for(int i=0; i<tamanho; i++){ ← O(n)
    if( vt[i] == 5) {
        mnp_3 += 50;
    }
}

return mnp_1+mnp_2+mnp_3;
}

int main() {
    std::vector<int> vt;
    vt.push_back(10); vt.push_back(20); vt.push_back(30);
    cout << exemplo3(vt); }
```

NESSA PARTE DO CÓDIGO
TEMOS: $O(n)$.

Aula 3- ALGORITMOS E COMPLEXIDADE

TEMA 1 - ANÁLISE DE ALGORITMOS - NOTAÇÃO O

CÁLCULO DA COMPLEXIDADE DE ALGORITMO: EXEMPLO 3:

AGORA TEMOS QUE ESCREVER EM UMA ÚNICA COMPLEXIDADE.

LOGO:

$$O(n) * O(n) + O(n) + O(n).$$

$$O(n^2) + 2 * O(n).$$

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

$$O(n^2) + O(n).$$

$$\rightarrow O(n^2).$$