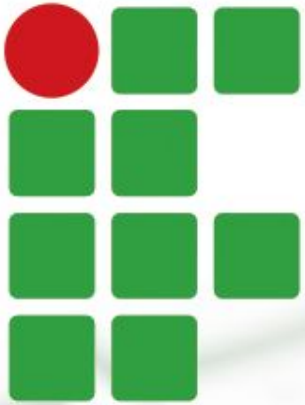


INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Estruturas de Dados 2
- Análise de Algoritmos -

Prof. MSc. Adriano Antunes Prates

Introdução

- Introdução à Análise de Algoritmos
- **Como determinar se um algoritmo é eficiente?**

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Introdução

- Introdução à Análise de Algoritmos
- **Como determinar se um algoritmo é eficiente?**

Tempo de Execução?

Introdução

- Introdução à Análise de Algoritmos
- **Como determinar se um algoritmo é eficiente?**

Tempo de Execução?

- **Problema**: Esta métrica é muito dependente de fatores externos ao código. P.ex.: Quanto ao hardware disponível, carga de trabalho no instante da avaliação, etc…

Notação “BIG O”

- A notação **Big O** é o método mais comum de se descrever o nível de complexidade - ou eficiência/velocidade - de um algoritmo.

- Esta notação leva em consideração o **Número de Operações** que um algoritmo deve executar para atingir um determinado objetivo.

- Vamos observar um exemplo simples e direto…

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Notação "BIG O"

- Considere que: o tempo de consulta de um registro em um Array leva 1 ms. Você precisa encontrar um registro específico…

Número de Elementos no Array	Tempo Máximo	
	Algoritmo: Busca Sequencial	Algoritmo: Busca Binária
10	10 ms	3 ms
100	100 ms	7 ms
10.000	10 seg	14 ms

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Notação "BIG O"

- Considere que: o tempo de consulta de um registro em um Array leva 1 ms. Você precisa encontrar um registro específico…

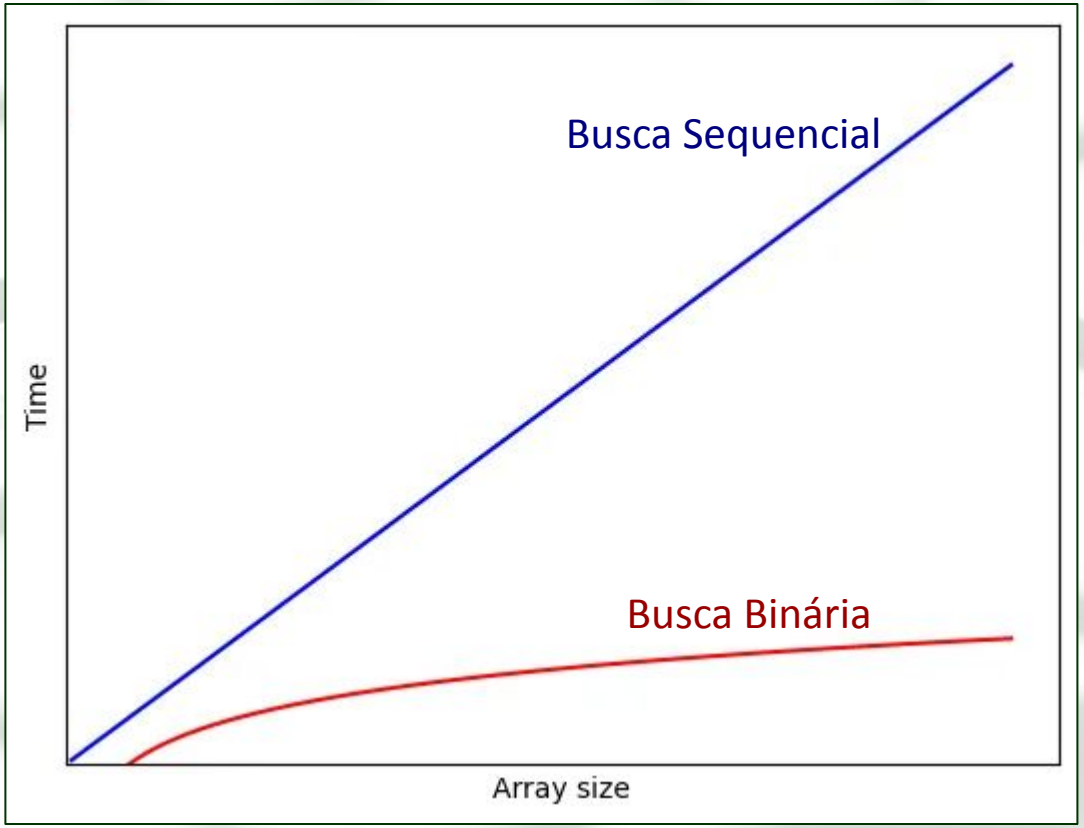
Número de Elementos no Array	Tempo Máximo	
	Algoritmo: Busca Sequencial	Algoritmo: Busca Binária
10	10 ms	3 ms
100	100 ms	7 ms
10.000	10 seg	14 ms
1.000.000.000	**11 DIAS**	**32 MS**

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Notação "BIG O"

- Fica nítido que os tempos de execução **não tem a mesma taxa de crescimento**.

Conforme o número de elementos cresce, o tempo da busca binária **aumenta minimamente**, enquanto o tempo de execução da busca sequencial **aumenta proporcionalmente**.

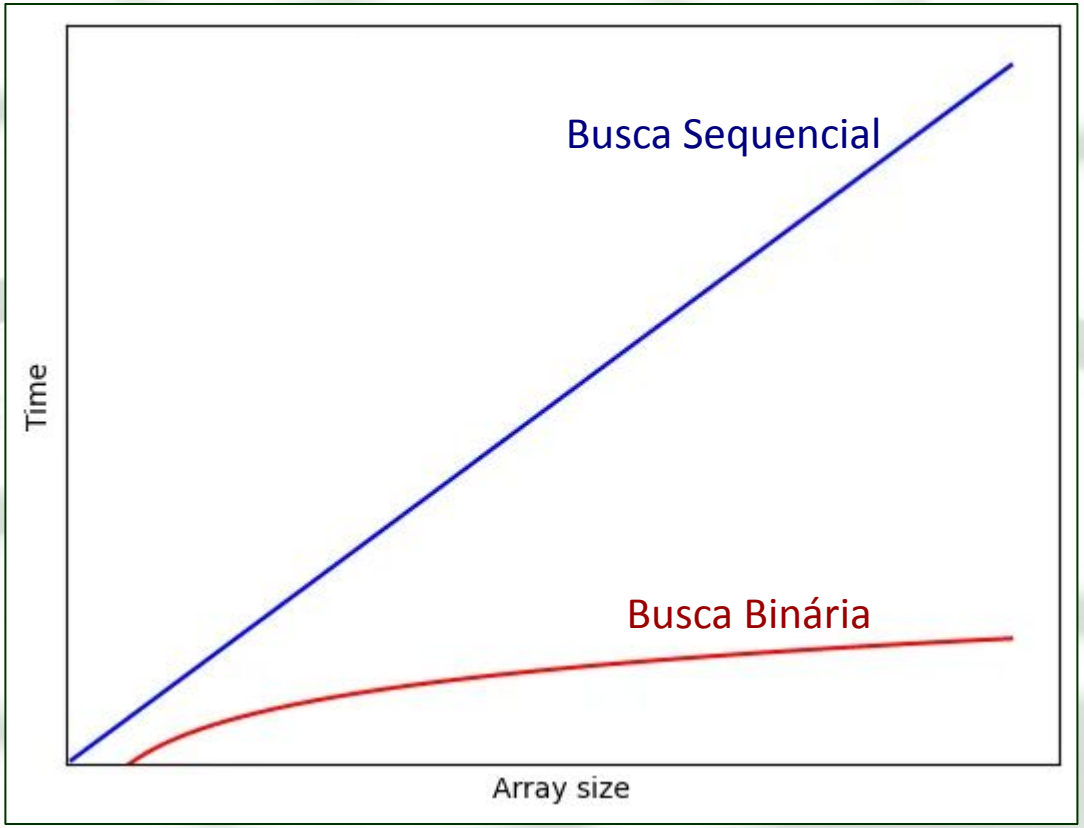


INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Notação "BIG O"

- Fica nítido que os tempos de execução **não tem a mesma taxa de crescimento**.

A notação Big O nos faz compreender como funciona a **escalabilidade de um algoritmo**, ou seja, como **o tamanho da entrada de dados pode impactar o tempo de resposta de um código**.

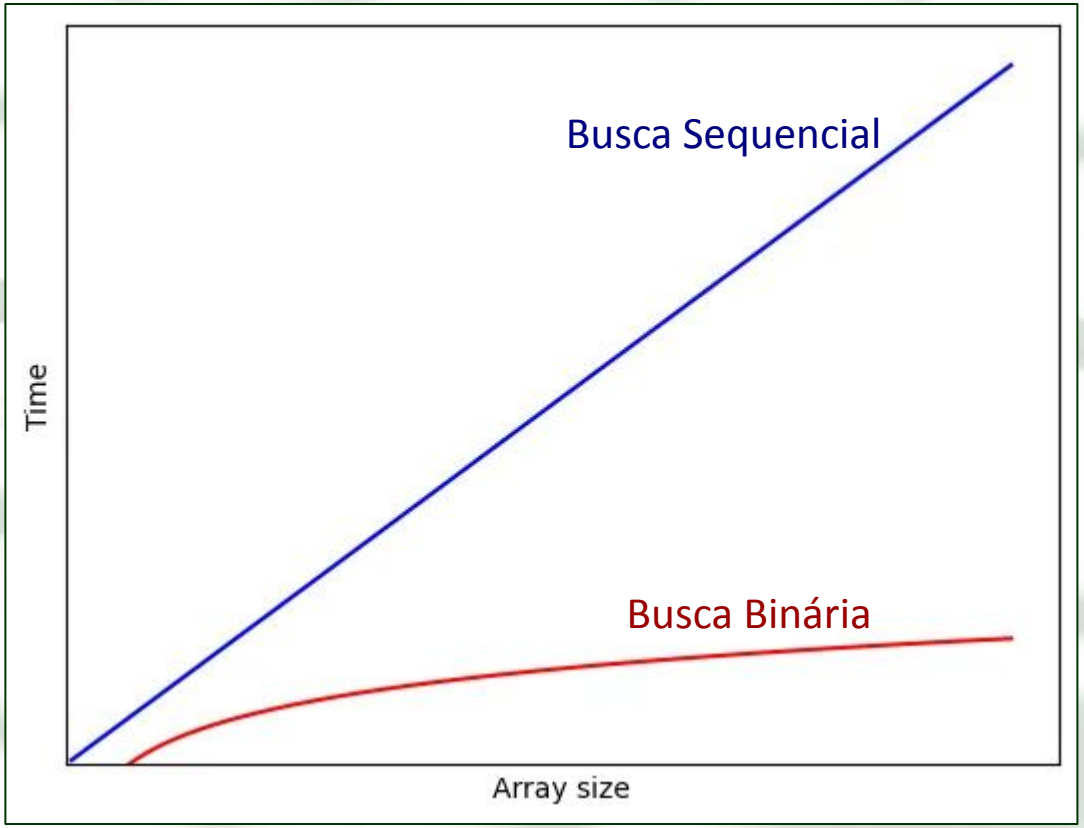


INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Notação "BIG O"

- Fica nítido que os tempos de execução **não tem a mesma taxa de crescimento**.

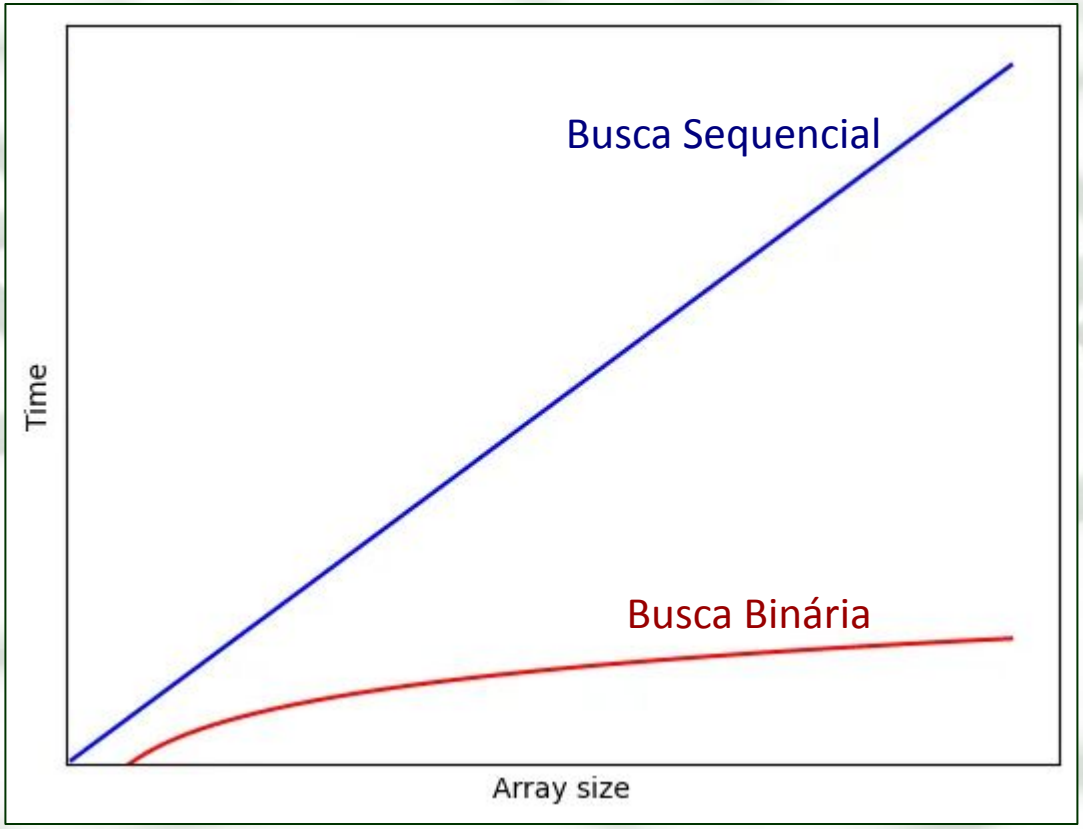
No "**mundo ideal**" a performance de um algoritmo deve (parecer) ser a mesma, **tanto para uma entrada de dados pequena, quanto para uma entrada de dados gigantesca.**



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Notação "BIG O"

- E qual é o segredo da "mágica" aqui???
- Qual a diferença entre os dois algoritmos???



Ordem Constante

- Algoritmos de **Ordem Constante**

O(1)

- São algoritmos em que o número de operações (tempo de execução) **independe** da quantidade de elementos.
- P.Ex.:

```
void multi(int array[], int indice, int valor){
    array[indice] *= valor;
}
```

Ordem Constante

- Algoritmos de **Ordem Constante**

O(1)

- São algoritmos em que o número de operações (tempo de e
elementos.
- P.Ex.:

Essa função irá executar uma única operação, independentemente se o array for de tamanho 10 ou 1.000.000.000

```
void multi(int array[], int indice, int valor){
    array[indice] *= valor;
}
```

Ordem Constante

- E se fosse uma função assim...

```
void troca(int array[], int indice){
    int temp = array[indice];
    array[indice] = array[indice+1];
    array[indice+1] = temp;
}
```

Ordem Constante

- E se fosse uma função assim...

```
void troca(int array[], int indice){
    int temp = array[indice];
    array[indice] = array[indice+1];
    array[indice+1] = temp;
}
```

O(3) ou O(1) ???

Ordem Constante

- E se fosse uma função assim...

```
void troca(int array[], int indice){
    int temp = array[indice];
    array[indice] = array[indice+1];
    array[indice+1] = temp;
}
```

O(3) ou O(1) ???

- Para notação Big O ainda seria O(1), pois embora faça mais operações, o tempo de execução é **CONSTANTE**, independente do tamanho do Array.

Ordem Constante

- E se fosse uma função assim…

```
void troca(int array[], int indice){
    int temp = array[indice];
    array[indice] = array[indice+1];
    array[indice+1] = temp;
}
```

Para Big O, não nos preocupamos se o código é O(1), O(2), O(23), O(647653), etc…

Arredonda-se para O(1), pois **a operação é uma linha plana em termos de escalabilidade**, e levaria a mesma quantidade de tempo independente do tamanho da entrada de dados.

Ordem Linear

- Algoritmos de **Ordem Linear**

O(n)

- São algoritmos em que o número de operações **aumenta proporcionalmente** à quantidade de elementos...

```
void multi(int array[], int cont, int valor){
    for(int i=0; i<cont; i++)
        array[i] *= valor;
}
```

Ordem Linear

- Algoritmo de busca sequencial…
  - **Constante ou Linear?**

```
int busca(int array[], int cont, int alvo){
    for(int i=0; i<cont; i++)
        if(array[i] == alvo)
            return i;
    return -1;
}
```

Ordem Linear

- Algoritmo de busca sequencial…
  - **Constante ou Linear?**

```
int busca(int array[], int cont, int alvo){
    for(int i=0; i<cont; i++)
        if(array[i] == alvo)
            return i;
    return -1;
}
```

A notação Big O **sempre** considera o **PIOR** caso de execução.

Entretanto, existem outras notações (Omega Ω, Theta Θ) que consideram o melhor caso e caso médio, respectivamente.

Ordem Quadrática

- Algoritmos de **Ordem Quadrática**

$$O(n^2)$$

- São algoritmos em que para N entrada de dados, precisaremos realizar N * N operações.

- Algum exemplo em mente???

Ordem Quadrática

- Bubble Sort | Selection Sort | Insertion Sort

$O(n^2)$

```
void ordena(int array[], int cont){
   for(int i=0; i<cont; i++)
      for(int j=0; j<cont-i-1; j++)
         if(array[j] > array[j+1])
            troca(array,j);
}
```

Ordem Quadrática

- Bubble Sort | Selection Sort | Insertion Sort

$O(n^2)$

```
void ordena(int array[], int cont){
  for(int i=0; i<cont; i++)
    for(int j=0; j<cont-i-1; j++)
      if(array[j] > array[j+1])
        troca(array,j);
}
```

repete n vezes

repete (n-i) vezes

Mudando de Assunto...

- Quantas vezes você seria capaz de dobrar uma folha de papel ao meio?

Mudando de Assunto…

- Havendo papel (e energia) suficiente…
- Dobrando 13 vezes você quebraria o **recorde mundial** (atualmente é 12) e entraria guiness book.

Mudando de Assunto…

- Havendo papel (e energia) suficiente…

- Dobrando 13 vezes você quebraria o recorde mundial (atualmente é 12) e entraria guiness book.
- Com 30 dobras você já poderia subir nele e **chegar ao espaço** (haveria 100 KM de altura).

Mudando de Assunto…

- Havendo papel (e energia) suficiente…

- Dobrando 13 vezes você quebraria o recorde mundial (atualmente é 12) e entraria guiness book.
- Com 30 dobras você já poderia subir nele e chegar ao espaço (haveria 100 KM de altura).
- Com 42 dobras você **chegaria à Lua**, e 51 ao **Sol**.

Mudando de Assunto…

- Havendo papel (e energia) suficiente…

- Dobrando 13 vezes você quebraria o recorde mundial (atualmente é 12) e entraria guiness book.
- Com 30 dobras você já poderia subir nele e chegar ao espaço (haveria 100 KM de altura).
- Com 42 dobras você chegaria à Lua, e 51 ao Sol.
- Com 103 dobras teria uma espessura do **tamanho do universo observável** (93 bilhões de anos-luz)

Fonte

Ordem Exponencial

- O que isso tem haver com a matéria?
- TUDO! Dobrar papel trata-se de um exemplo de problema de **Ordem Exponencial**.
- Algoritmos de **Ordem Exponencial**

$$O(2^n)$$

Ordem Exponencial

- Algoritmos de **Ordem Exponencial** só perdem para os de **Ordem Fatorial** em relação à complexidade e custo (número de operações e tempo de execução envolvidos).

- Um exemplo de Algoritmo de Complexidade Exponencial são os de **Brute Force Attack**.

- *P.Ex.:* Uma senha forte de 12 caracteres levaria aproximadamente 8 milhões de anos para ser quebrada, mesmo nos supercomputadores.

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Ordem Exponencial

Password Length	Numerical 0-9	Upper & Lower case a-Z	Numerical Upper & Lower case 0-9 a-Z	Numerical Upper & Lower case Special characters 0-9 a-Z %$
1	instantly	instantly	instantly	instantly
2	instantly	instantly	instantly	instantly
3	instantly	instantly	instantly	instantly
4	instantly	instantly	instantly	instantly
5	instantly	instantly	instantly	instantly
6	instantly	instantly	instantly	20 sec
7	instantly	2 sec	6 sec	49 min
8	instantly	1 min	6 min	5 days
9	instantly	1 hr	6 hr	2 years
10	instantly	3 days	15 days	330 years
11	instantly	138 days	3 years	50k years
12	2 sec	20 years	162 years	8m years
13	16 sec	1k years	10k years	1bn years
14	3 min	53k years	622k years	176bn years
15	26 min	3m years	39m years	27tn years
16	4 hr	143m years	2bn years	4qdn years
17	2 days	7bn years	148bn years	619qdn years
18	18 days	388bn years	9tn years	94qtn years
19	183 days	20tn years	570tn years	14sxn years
20	5 years	1qdn years	35qdn years	2sptn years

Ordem Logarítmica

- Saindo de um extremo (Algoritmos de maior ordem de complexidade) e indo para o extremo oposto, temos os **Algoritmos de Ordem Logarítmica**.

- Algoritmos de **Ordem Logarítmica**

$$O(\log_2 n)$$

ou simplesmente . . .

$$O(\log n)$$

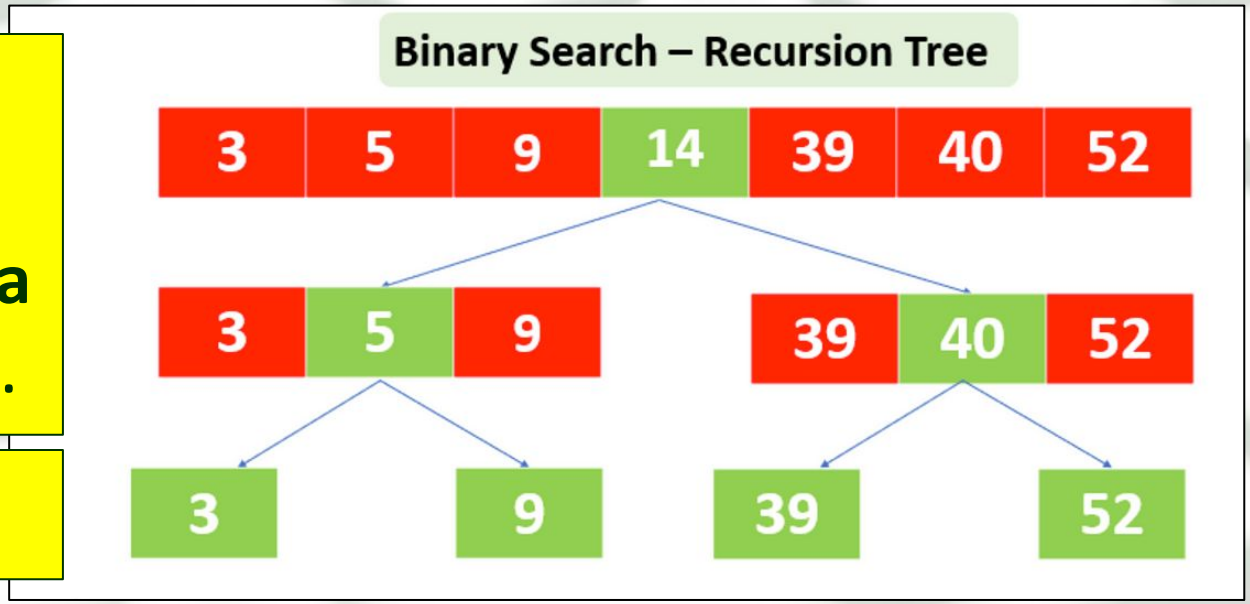
INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Ordem Logarítmica

- Algoritmos de Ordem Logarítmica caracterizam-se pela delimitação da entrada de dados (geralmente 50%) a cada iteração realizada…

Exemplo clássico de algoritmo de **ordem logarítmica** é a **Busca Binária**.

Log_2 7 == 3

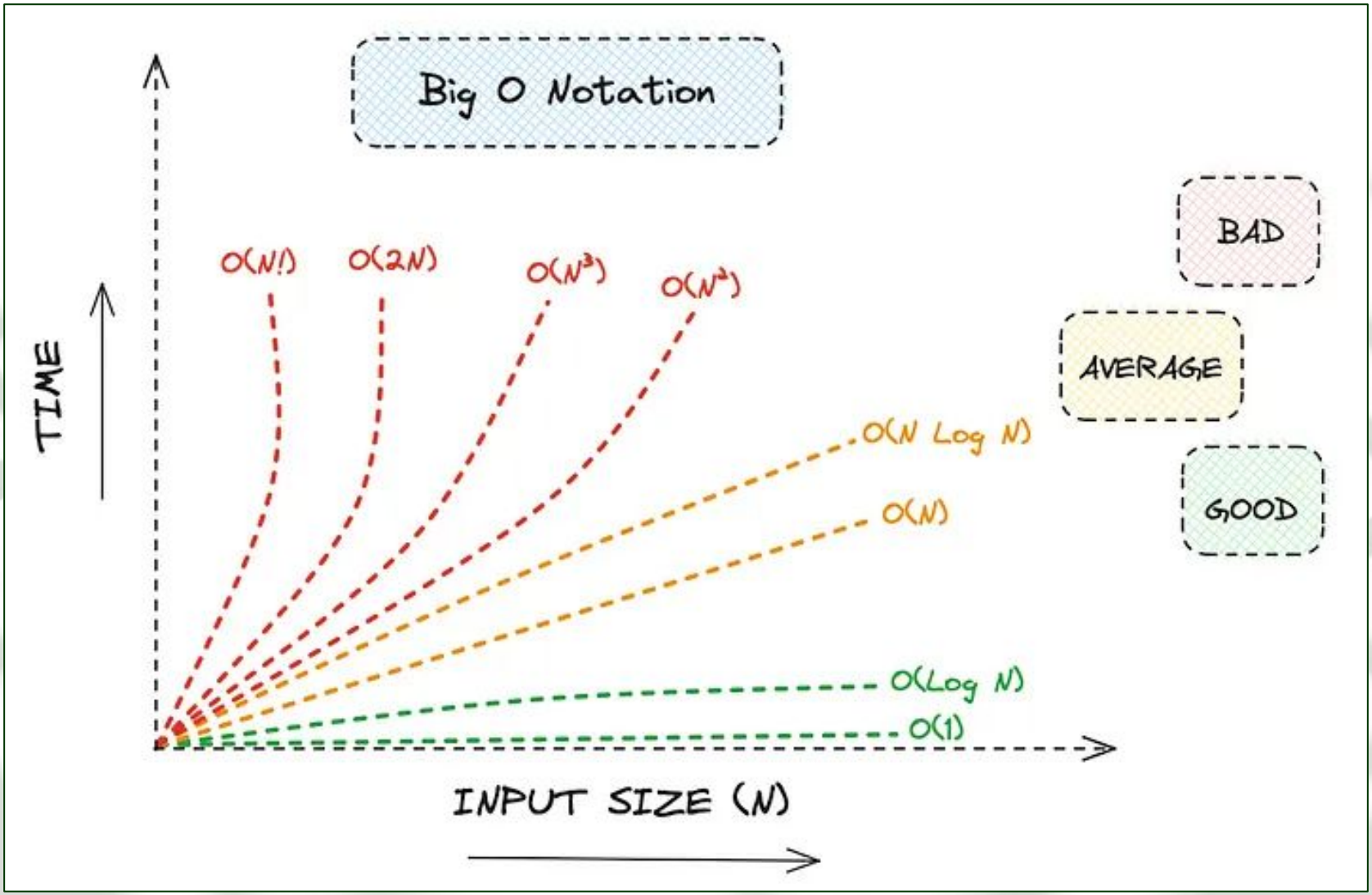


Principais Classificações

Ordem	Classificação	Exemplo de Algoritmo
O(1)	Constante	Acesso direto a elementos
O(n)	Linear	Busca Sequencial
$O(n^2)$	Quadrático	Selection Sort
O(log n)	Logarítmico	Busca Binária
O(n log n)	Linearítmica	Merge Sort
$O(2^n)$	Exponencial	Brute Force
O(n!)	Fatorial	Caixeiro Viajante

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Principais Classificações



Complexidades

- Qual é a complexidade dos algoritmos clássicos de ordenação?
  - Bubble, Insertion e Selection?

- Seria possível criar um algoritmo de ordenação **tão simples quanto os clássicos**, mas com uma eficiência melhor?

Complexidades

- Analise o Funcionamento do Método de Ordenação **Counting Sort**

- LINK

- Qual é a **ordem de complexidade** deste método?

Complexidades

- Qual é a **ordem de complexidade** deste método?
  - 1 Laço de repetição para contagem dos elementos (N => Linear)
  - 1 Laço de repetição para espalhar as K chaves contadas (K => Linear)

- Perceba… 02 laços de repetição simples, não-aninhados, diferentemente dos outros algoritmos clássicos.

O(n+k), ou simplesmente O(n)

Complexidades

- Qual é a **ordem de complexidade** deste método?
  - 1 Laço de repetição para contagem dos elementos (N => Linear)
  - 1 Laço de repetição para espalhar as K chaves contadas (K => Linear)

- Perceba… 02 laços de repetição simples, não-aninhados, diferentemente dos outros algoritmos clássicos.

Qual é o porém entretanto?

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Complexidade de Espaço

- Embora a complexidade de operações (tempo) seja mais eficiente (Linear), a complexidade de espaço no **Counting Sort** é maior.
- A ordenação não é *in-place*, porque exige espaço de memória extra, proporcional ao tamanho de k.

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)	O(1)
Counting Sort	Ω(n+k)	θ(n+k)	O(n+k)	O(k)

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Complexidade de Espaço

- Embora a complexidade de operações (tempo) seja mais eficiente (Linear), a complexidade de espaço no **Counting Sort** é maior.
- A ordenação não é *in-place*, porque exige espaço de memória extra, proporcional ao tamanho de k.

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)	O(1)
Counting Sort	Ω(n+k)	θ(n+k)	O(n+k)	O(k)

?

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Complexidade de Espaço

- Embora a complexidade de operações (tempo) seja mais eficiente (Linear), a complexidade de espaço no **Counting Sort** é maior.
- A ordenação não é *in-place*, porque exige espaço de memória extra, proporcional ao tamanho de k.

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)	O(1)
Counting Sort	Ω(n+k)	θ(n+k)	O(n+k)	O(k)

Memória adicional requerida INDEPENDE do tamanho da entrada de dados.

Complexidades

- **Complexidade de Tempo (Operações)**
  - **Número de operações** que um algoritmo necessita para completar seu objetivo à medida que a entrada de dados aumenta.

- **Complexidade de Espaço**
  - **Quantidade de memória** utilizada por um algoritmo durante sua execução à medida que a entrada de dados aumenta.

Referências

- Big-O Cheat Sheet
- Iniciando a Notação Big O
- O que é Notação Big O