UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Câmpus Campo Mourão

Ciência da Computação

Algoritmos e Estrutura de Dados 1

Noções Sobre Complexidade de Algoritmos

Prof. Rafael Liberato

liberato@utfpr.edu.br

Roteiro

- **Objetivos**
- **Fundamentos de Análise de Algoritmos**
- **Cálculo da função de custo**
- **Análise assintótica**
  - Classes de comportamento assintótico
- **Métodos de busca**
  - Busca Linear
  - Busca Binária

Objetivos

- **Todo objeto do mundo real possui características na qual possamos categorizá-los e/ou compará-los**
- **Podemos categorizar sem mencionar os detalhes das reais dimensões**
  - ➔ Carro de médio porte. Passamos a ideia sem mencionar dimensões
- **Com algoritmos não é diferente, precisamos compará-los e categorizá-los**
  - ➔ Quando buscamos soluções computacionais para problemas, a comparação é inevitável

Objetivos

- **Como passar a ideia da complexidade de um algoritmo apenas mencionando sua categoria?**
  - ➔ Existem notações que representam essa categorização baseado no comportamento de execução do algoritmo
- **Por ser uma introdução veremos somente a notação O, mas antes vamos ver como se analisa um algoritmo**

Fundamentos de Análise de Algoritmos

Fundamentos de Análise de Algoritmos

Como analisar um algoritmo

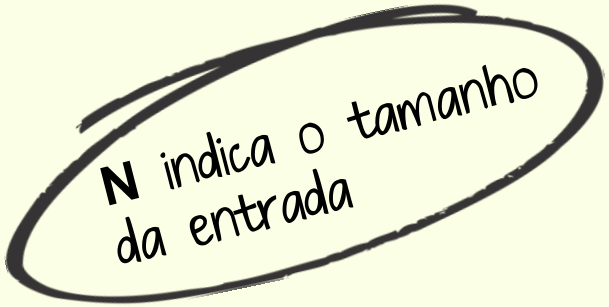
- Tempo gasto na execução não é uma boa opção
- Contar o número de instruções executadas pelo algoritmo para uma determinada entrada
- Expressar este número em função do tamanho da entrada (Função de custo)

Por exemplo

- $f(n) = 5n$
- $f(n) = 2n^2$

Fundamentos de Análise de Algoritmos

- **A análise é realizada em função de N**
  - Número de elementos de um vetor
  - Número de linhas de uma matriz



- **Diferentes entradas podem refletir em diferentes custos**
  - Melhor caso
  - Pior caso
  - Caso médio

Fundamentos de Análise de Algoritmos

- **Exemplo: Busca sequencial no vetor**

Quantas posições precisam ser percorridas:

Melhor caso:

Pior caso:

Caso médio

```
int busca (int* v, int n, int elemento){
    int i;
    for (i=0; i<n; i++){
        if(v[i] == elemento) return 1;
    }
    return 0;
}
```

Fundamentos de Análise de Algoritmos

- **A análise sempre é realizada baseado em um determinado modelo computacional**

- **No nosso exemplo, vamos considerar o seguinte modelo:**
  - O computador tem um único processador
  - Todos os acessos à memória têm o mesmo custo
  - As instruções são executadas sequencialmente
  - Não há execuções paralelas
  - Todas as execuções têm custo igual, uma unidade

Fundamentos de Análise de Algoritmos

- **Vamos analisar as diferenças entre funções de custo. Suponha:**
  1. Um processador de 1GHz
  2. Uma instrução executada a cada ciclo de máquina
     1GHz = 10^9 instruções por segundo
  3. Entrada de tamanho **n = 1.000.000**

Algoritmo com custo	**f(n) = n**	Tempo:	**1 milissegundo**
Algoritmo com custo	**f(n) = 100n**	Tempo:	**1/10 de segundo**
Algoritmo com custo	**f(n) = n^2**	Tempo:	**17 minutos**
Algoritmo com custo	**f(n) = n^3**	Tempo:	**32 anos**

Cálculo da função de custo

Exemplo

```
void troca(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

f(n) = 3

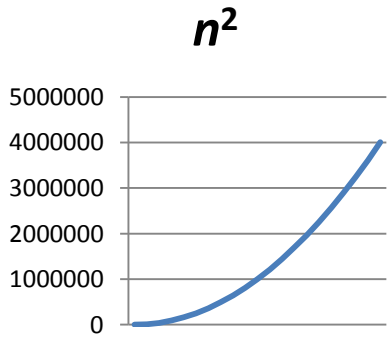
```
for (i=0; i < n; i++)
    v[i] = 0;
```

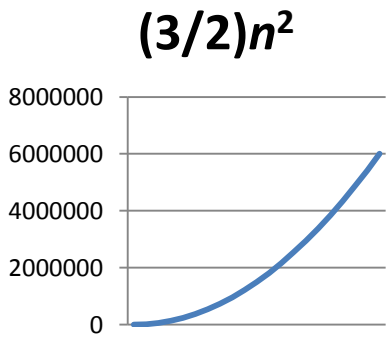
f(n) = 3n + 2

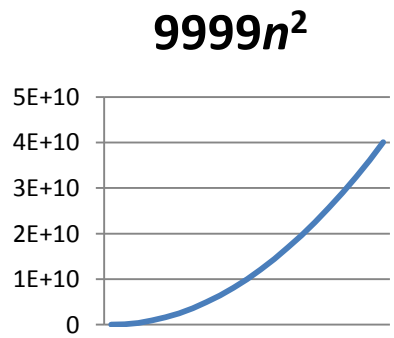
`i=0`	**1**
`i<n`	**n + 1**
`i++`	**n**
`v[i]=0`	**n**

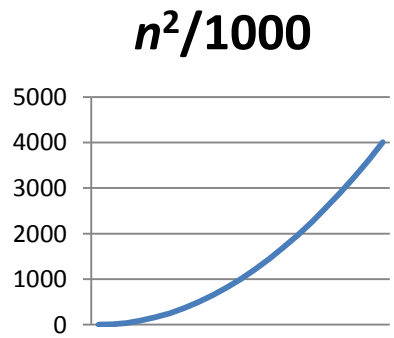
Análise assintótica

- Na análise de algoritmos ignoramos os valores pequenos e nos concentramos nos valores enormes de n
- Para grandes valores de n, as funções abaixo são equivalentes pois crescem com a mesma velocidade









Análise assintótica

- **Notação O**
  - Delimita um limite superior
- **Notação Ω**
  - Delimita um limite inferior
- **Notação Θ**
  - Delimita um limite superior e inferior

Análise assintótica

- **Nessa aula utilizaremos a notação O para analisar os exemplos.**
- **Por exemplo** Considere o seguinte algoritmo que procura um elemento em um vetor

```
int busca (int* v, int n, int elemento){
    int i;
    for (i=0; i<n; i++){
        if(v[i] == elemento) return 1;
    }
    return 0;
}
```

Melhor caso:

O elemento é encontrado na primeira posição

Pior caso:

O elemento é encontrado na última posição ou não é encontrado

Caso médio:

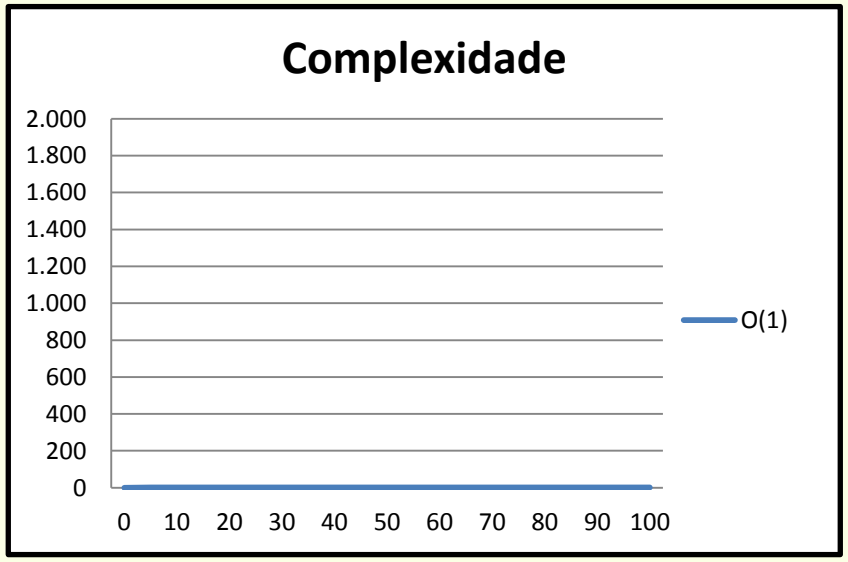
O elemento é encontrado nas posições intermediárias

Classes de comportamento assintótico

- **Podemos comparar algoritmos usando suas complexidades assintóticas**
  - Um algoritmo $O(n)$ é melhor do que um $O(n^2)$
  - Algoritmos com a mesma complexidade assintótica são equivalentes
- **Quando dois algoritmos têm a mesma complexidade**
  - Podemos desempatar usando as constantes da função

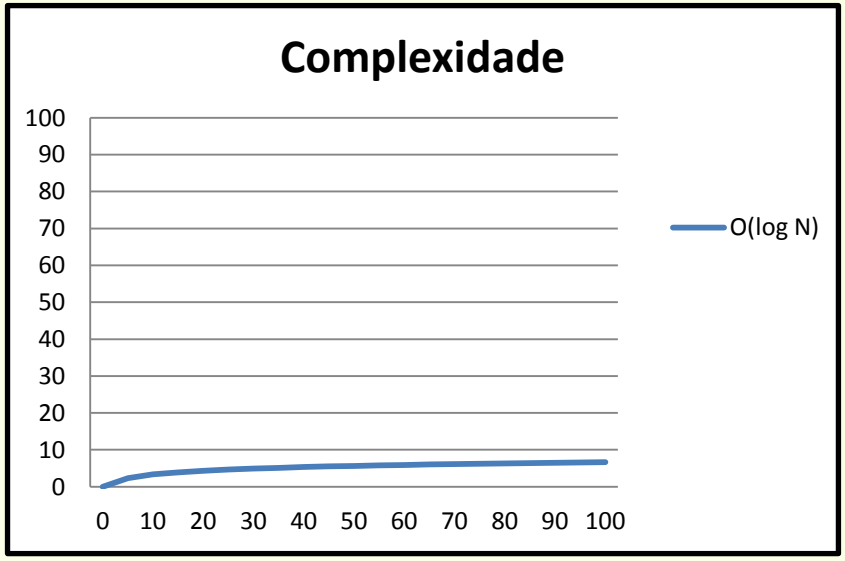
f(n) = O(1)

- Complexidade constante
- Tempo de execução independe do tamanho da entrada
- Os passos do algoritmo são executados um número fixo de vezes



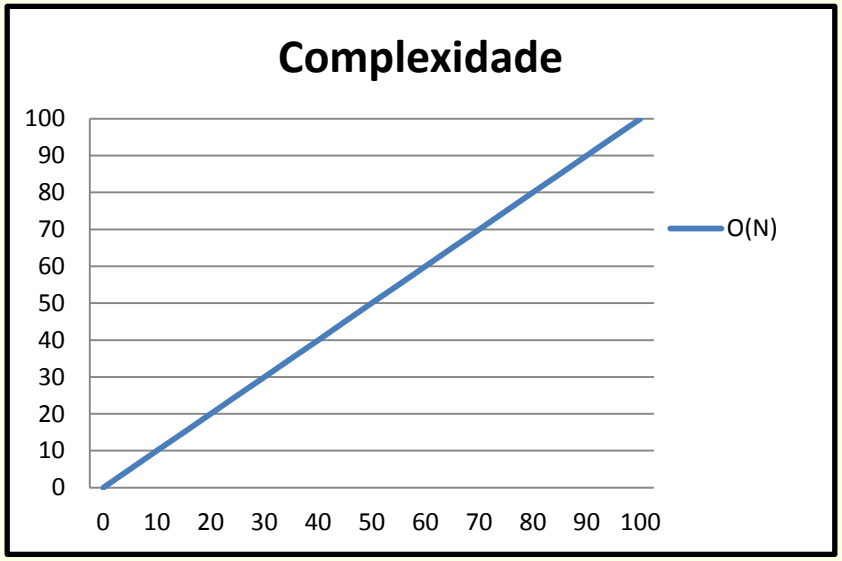
f(n) = O(log n)

- Complexidade logarítmica
- Típico de algoritmos que dividem um problema transformando-o em problemas menores (dividir para conquistar)



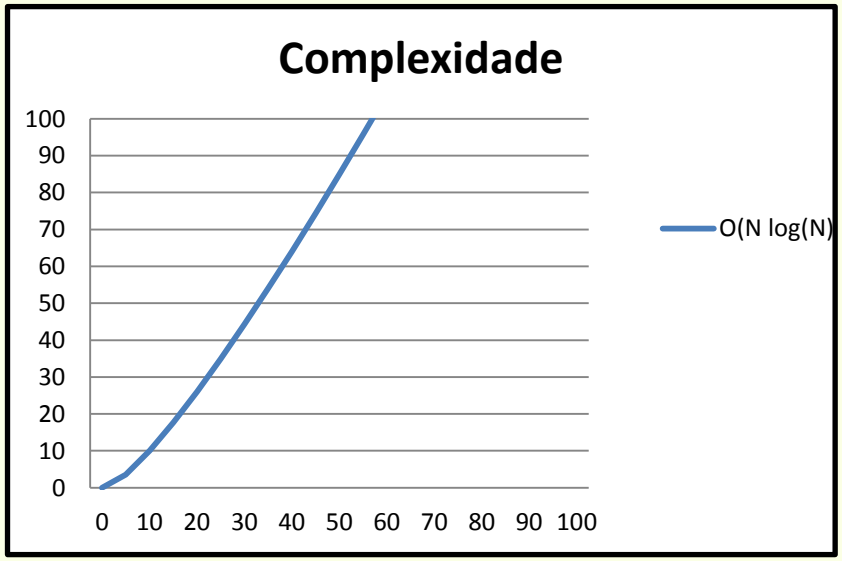
f(n) = O(n)

- Complexidade linear
- O algoritmo realiza um número fixo de operações sobre CADA elemento da entrada



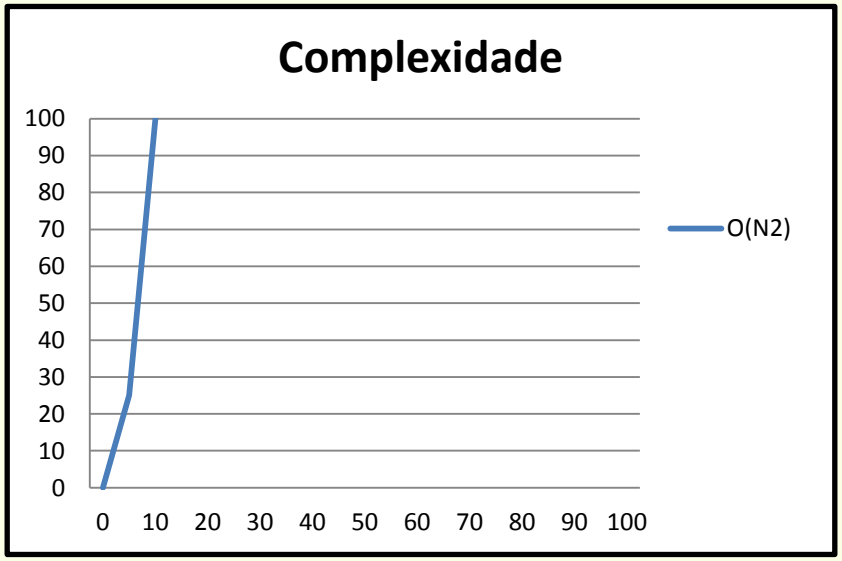
f(n) = O(n log n)

- Típico de algoritmos que dividem um problema em subproblemas, resolve cada subproblema de forma independente, e depois combina os resultados
- Exemplo: ordenações eficientes



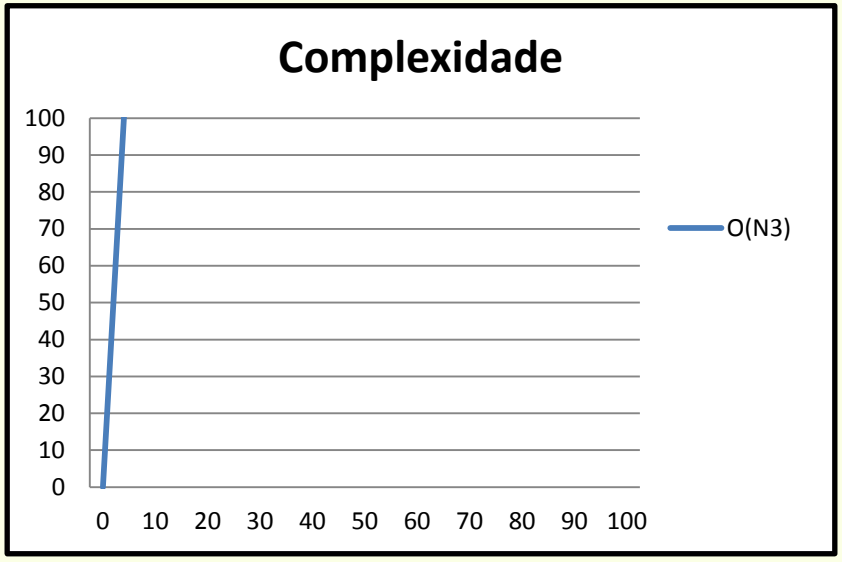
$f(n) = O(n^2)$

- Complexidade quadrática
- Normalmente em um laço dentro de outro
- Exemplo: Imprimir uma matriz



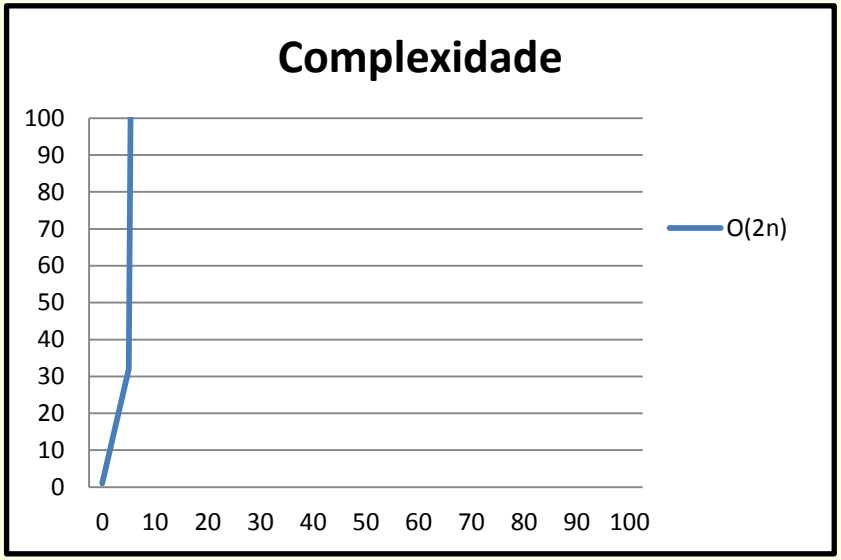
$f(n) = O(n^3)$

- Complexidade cúbica
- Exemplo: multiplicação de matrizes



$f(n) = O(c^n)$

- Complexidade exponencial
- Típicos de algoritmos que fazem busca exaustiva (força bruta) para resolver um problema
- Não são úteis do ponto de vista prático
  - Se n=20, $O(2^n)$=1.000.000



f(n) = O(n!)

- **Complexidade exponencial**
  - Pior do que $O(c^n)$
- **Típicos de algoritmos que fazem busca exaustiva (força bruta) para resolver um problema**
- **Não são úteis do ponto de vista prático**
  - Se n=20, O(n!) é maior do que 2 quintilhões

Classes de comportamento assintótico

Comparação

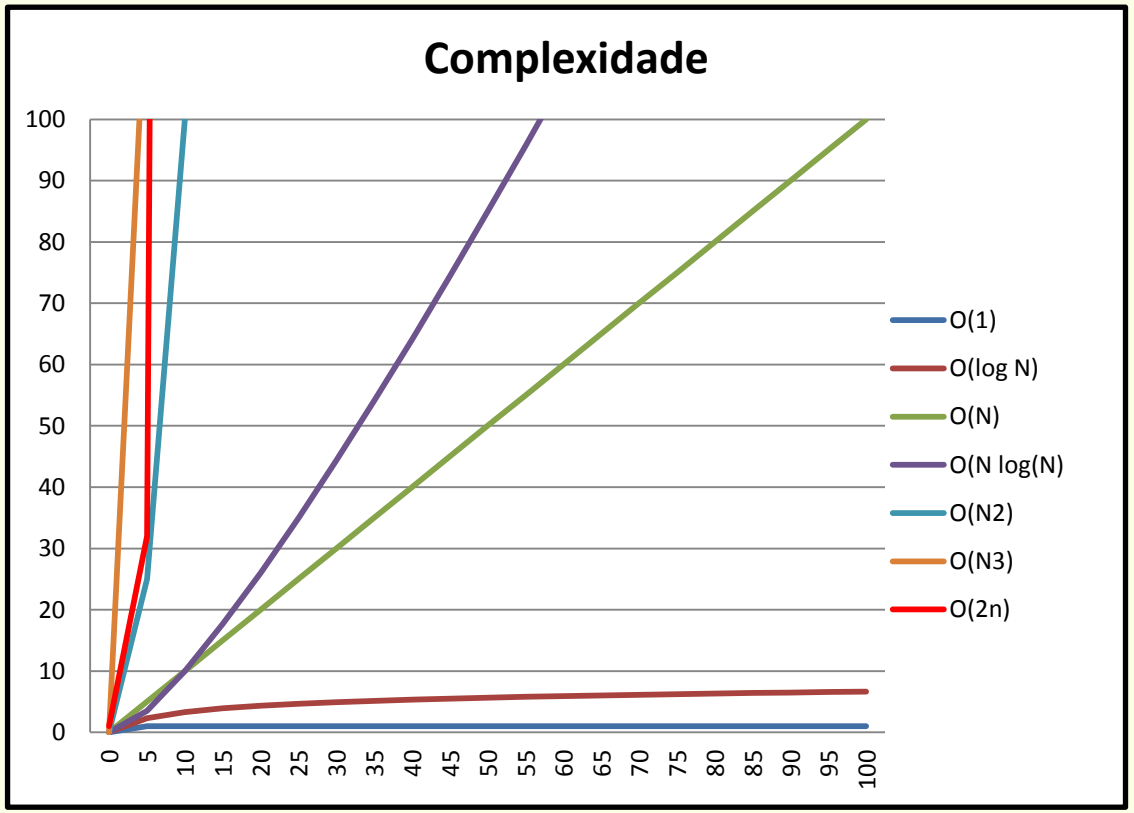
N	O(1)	O(log N)	O(N)	O(N log(N)	$O(N^2)$	$O(N^3)$	$O(2^n)$
0	0	0,00	0	0	0	0	1
5	1	2,32	5	3	25	125	32
10	1	3,32	10	10	100	1.000	1.024
15	1	3,91	15	18	225	3.375	32.768
20	1	4,32	20	26	400	8.000	1.048.576
25	1	4,64	25	35	625	15.625	33.554.432
30	1	4,91	30	44	900	27.000	1.073.741.824
35	1	5,13	35	54	1.225	42.875	34.359.738.368
40	1	5,32	40	64	1.600	64.000	1.099.511.627.776
45	1	5,49	45	74	2.025	91.125	35.184.372.088.832
50	1	5,64	50	85	2.500	125.000	1.125.899.906.842.620
55	1	5,78	55	96	3.025	166.375	36.028.797.018.964.000
60	1	5,91	60	107	3.600	216.000	1.152.921.504.606.850.000
65	1	6,02	65	118	4.225	274.625	36.893.488.147.419.100.000
70	1	6,13	70	129	4.900	343.000	1.180.591.620.717.410.000.000
75	1	6,23	75	141	5.625	421.875	37.778.931.862.957.200.000.000
80	1	6,32	80	152	6.400	512.000	1.208.925.819.614.630.000.000.000
85	1	6,41	85	164	7.225	614.125	38.685.626.227.668.100.000.000.000
90	1	6,49	90	176	8.100	729.000	1.237.940.039.285.380.000.000.000.000
95	1	6,57	95	188	9.025	857.375	39.614.081.257.132.200.000.000.000.000
100	1	6,64	100	200	10.000	1.000.000	1.267.650.600.228.230.000.000.000.000.000

Classes de comportamento assintótico

- O(1)
- O(n)
- O(log n)
- O(n log n)
- O(n^2)
- O(n^3)
- O(2^n)

+ eficiente

- eficiente



Exemplos

Função de Custo	Categoria
5	O(1)
n	O(n)
7n + 9	O(n)
n^2	O(n^2)
$10n^2 + 20n + 8$	O(n^2)

Classes de comportamento assintótico

Para compreender os algoritmos cujo comportamento acompanha a função **log n**, vamos conhecer alguns métodos de busca.

Métodos de Busca

Métodos de busca

- **O fato dos elementos de um vetor estarem ordenados facilita a busca?**
  - Sim
- **Como?**
  - Busca Linear
  - Busca Binária

Busca Linear

- **Busca convencional com que estamos acostumados**
  - A diferença é que no vetor ordenado a pesquisa terminará se um item com uma chave maior for encontrado
  - Entretanto não uma melhora muito significativa

```
int buscaLinear(int *v, int n, int elemento){
    int i;
    for(i=0; i<n; i++){
        count++;
        if(v[i] == elemento) return 1;
        if(v[i] > elemento) break;
    }
    return 0;
}
```

Busca Binária

- **É muito mais rápida do que a Pesquisa Linear**
  - Para encontrar um número entre 1 e 100, precisamos, no máximo, de 7 passos

- **A pesquisa binária usa a mesma dinâmica de um jogo de adivinhação em que fazemos uma tentativa e obtemos as respostas:**
  - É MAIOR
  - É MENOR
  - Acertou

Passos	Número adivinhado	Resultado	Faixas de possíveis valores
0			1-100
1	50	Menor	1-49
2	25	Maior	26-49
3	37	Menor	26-36
4	31	Maior	32-36
5	34	Menor	32-33
6	32	Maior	33-33
7	33	Correto	

Busca Binária

- **A busca binária é um algoritmo que executa em O(log n)**
- **Vamos entender o porquê**
  - A cada passo reduzimos nossa entrada pela metade.
  - Assim, se n=100 temos

```
100 | 2
 50 | 2
 25 | 2
12,5| 2
6,25| 2
3,13| 2
1,57| 2
0,79
```

Número de Divisões

1 2 3 4 5 6 7

Quantas vezes precisamos multiplicar o **2** por si mesmo para obter **100**

$\log_2(100) = 6{,}644$

$\log_2(100) = 7$ {arredondado}

Ou seja, com **7** tentativas conseguimos encontrar um valos em um conjunto de **128** elementos

$2^7 = 128$

Busca Binária

Comparando

Faixa		Comparações Pesquisa Linear		Comparações Pesquisa Binária	
10 = (10/**2**/**2**/**2**/**2** = **0,625**)		5		4	
100	**x 10**	50	**x 10**	7	**+ 3,322**
1.000	**x 10**	500	**x 10**	10	**+ 3,322**
10.000	**x 10**	5.000	**x 10**	14	**+ 3,322**
100.000	**x 10**	50.000	**x 10**	17	**+ 3,322**
1.000.000	**x 10**	500.000	**x 10**	20	**+ 3,322**

Exercício

- Escreva uma versão recursiva da busca binária

Referências

- LAFORE Robert. Estrutura de Dados e Algoritmos em Java. 2ª Edição. Rio de Janeiro: Ciência Moderna, 2005.
- Análise assintótica: ordens O, Omega e Theta. Análise de Algoritmos. Prof. Paulo Feofiloff. http://www.ime.usp.br/~pf/analise_de_algoritmos/
- Ítalo Cunha. Material da disciplina de Algoritmos e Estrutura de Dados 1. Departamento de Ciência da Computação. UFMG. http://homepages.dcc.ufmg.br/~cunha/