Algoritmos e Estrutura de Dados II

Prof. Fellipe Guilherme Rey de Souza

Aula 01 – Complexidade de Algoritmos

Agenda

- O que é complexidade?
- Anotação O (Big-Oh)
- Comparação entre funções
- Exemplos

Mas afinal, o que significa analisar a complexidade de um Software?

Complexidade de leitura? De escrita?

O que é complexidade?

```c
/*
 * Tokenise an ASN.1 grammar
 */
static void tokenise(char *buffer, char *end)
{
	struct token *tokens;
	char *line, *nl, *start, *p, *q;
	unsigned tix, lineno;

	/* Assume we're going to have half as many tokens as we have
	 * characters
	 */
	token_list = tokens = calloc((end - buffer) / 2, sizeof(struct token));
	if (!tokens) {
		perror(NULL);
		exit(1);
	}
	tix = 0;

	lineno = 0;
	while (buffer < end) {
		/* First of all, break out a line */
		lineno++;
		line = buffer;
		nl = memchr(line, '\n', end - buffer);
		if (!nl) {
			buffer = nl = end;
		} else {
			buffer = nl + 1;
			*nl = '\0';
		}

		/* Remove "--" comments */
		p = line;
	next_comment:
		while ((p = memchr(p, '-', nl - p))) {
			if (p[1] == '-') {
				/* Found a comment; see if there's a terminator */
				q = p + 2;
				while ((q = memchr(q, '-', nl - q))) {
					if (q[1] == '-') {
						/* There is - excise the comment */
						q += 2;
						memmove(p, q, nl - q);
						goto next_comment;
					}
					q++;
				}
				*p = '\0';
				nl = p;
				break;
			} else {
				p++;
			}
		}

		p = line;
		while (p < nl) {
			/* Skip white space */
			while (p < nl && isspace(*p))
				*(p++) = 0;
			if (p >= nl)
				break;

			tokens[tix].line = lineno;
			start = p;

			/* Handle string tokens */
```

Disponível em: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/scripts/asn1_compiler.c?h=v6.14-rc4

O que é complexidade?

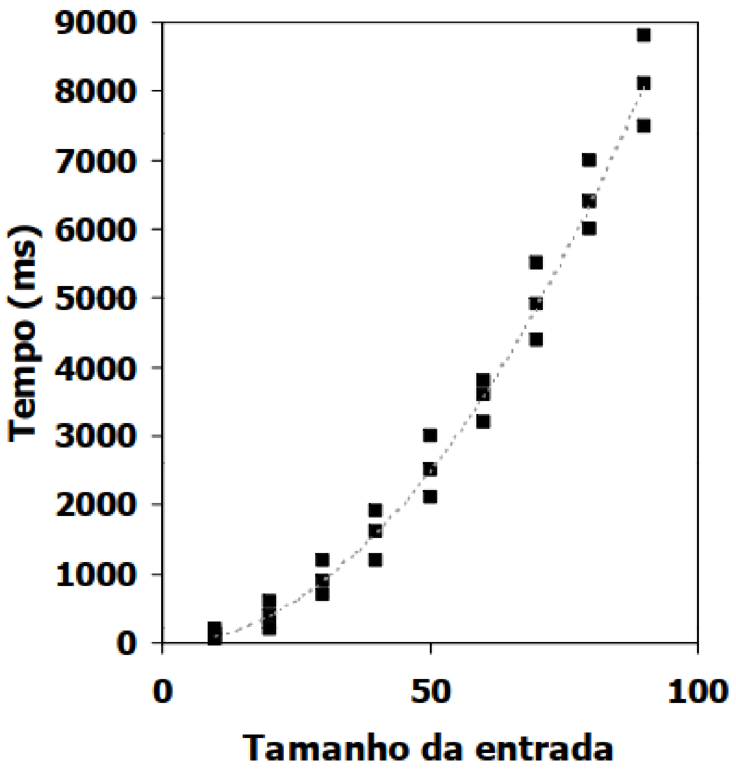
```c
int minDistance(int dist[], int sptSet[]) {
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++) {
        if (sptSet[v] == 0 && dist[v] <= min) {
            min = dist[v];
            min_index = v;
        }
    }
    return min_index;
}

void printSolution(int dist[], int n) {
    printf("Vertice \t Distancia da origem\n");
    for (int i = 0; i < n; i++) {
        printf("%d \t\t %d\n", i, dist[i]);
    }
}
```

```c
void dijkstra(int graph[V][V], int src) {
    int dist[V];
    int sptSet[V];
    for (int i = 0; i < V; i++) {
        dist[i] = INT_MAX;
        sptSet[i] = 0;
    }
    dist[src] = 0;
    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet);
        sptSet[u] = 1;
        for (int v = 0; v < V; v++) {
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }
    printSolution(dist, V);
}
```

Fonte: ChatGPT (Gerado em 12/01/2025)

Como resolver um problema?

1. Para resolver um problema, implemente um determinado algoritmo.
2. Execute esse algoritmo com diversas instâncias do problemas (valores e tamanhos variados)
3. Meça o tempo real dessas execuções.
4. Desenhe um gráfico com os resultados obtidos.



O que é complexidade?

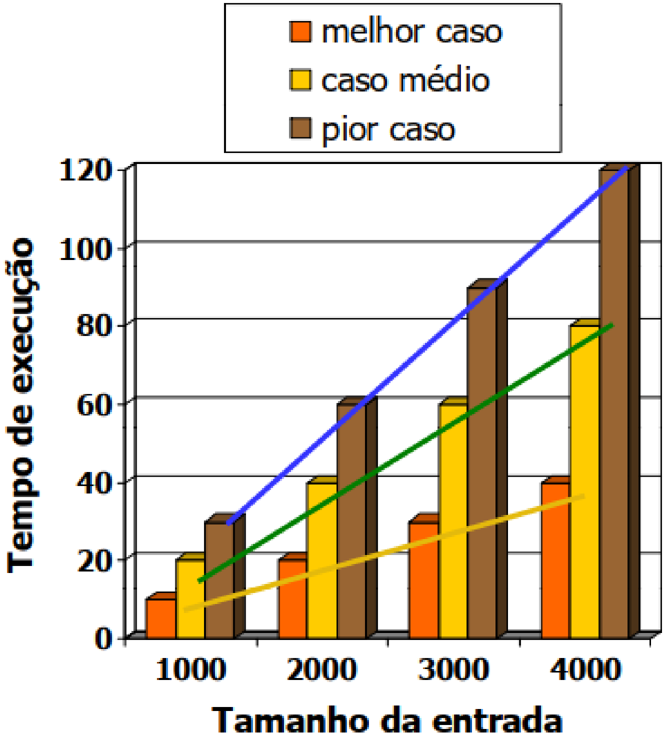
- O tempo de execução de um algoritmo varia – e normalmente cresce – com o tamanho da entrada do problema.

- Além disso, para instâncias de mesmo tamanho, também há variação no tempo de execução.

O que é complexidade?

- Geralmente, o tempo médio é difícil de determinar.

- Costuma-se estudar os tempos máximos (pior caso), porque:
  i. É mais fácil de analisar;
  ii. É crucial para a qualidade das aplicações.



O que é complexidade?

- A **Análise Teórica de Complexidade** leva em consideração todas as possíveis entradas de um algoritmo.

- Permite a avaliação do desempenho de um algoritmo, independentemente das características do hardware e do software utilizados.

O que é complexidade?

- De modo geral, são computações básicas realizadas por um algoritmo:
  - Atribuição de valor para uma variável
  - Comparação entre valores
  - Cálculo aritmético
  - Chamada de função
  - etc.

O que é complexidade?

- Existem duas características importantes na análise de algoritmos:
    i. Tempo de execução
    ii. Quantidade de memória (RAM) utilizada

- Para o tempo de execução, analisaremos a **Complexidade de Tempo**. Para a quantidade de memória utilizada, analisaremos a **Complexidade de Espaço**.

O que é complexidade?

- <u>Ex</u>: Construa uma função que devolva o maior elemento de um vetor.

```
int arrayMax(int arr, int n) {
   currentMax = arr[0];
   for (int i = 1; i < n; i++) {
      if (arr[i] > currentMax) {
         currentMax = arr[i];
      }
   }
   return currentMax;
}
```

1 atribuição e 1 indexação
1 atribuição
Repete n-1 vezes: [
    1 teste
    1 indexação e 1 teste
    1 atribuição e 1 indexação (no máximo)
    1 incremento
]
1 teste
1 return

<u>Total</u>: 5 + (n – 1).6 = 6n - 1

O que é complexidade?

- Definições:
  - **a**: Tempo gasto na execução da operação primitiva mais rápida
  - **b**: Tempo gasto na execução da operação primitiva mais lenta
  - Seja T(n) o tempo real de execução de pior caso de *arrayMax*.

- Portanto, **$a(6n - 1) \leq T(n) \leq b(6n - 1)$**
  - O tempo de execução T(n) é limitado por duas funções lineares.

O que é complexidade?

- Alterações nos ambientes de *hardware* e *software* vão afetar T(n) somente por um fator constante.
  - Portanto, eles não alteram a taxa de crescimento de T(n) – ela continua linear!

- Portanto, a linearidade de T(n) é uma propriedade intrínseca do algoritmo *arrayMax*.

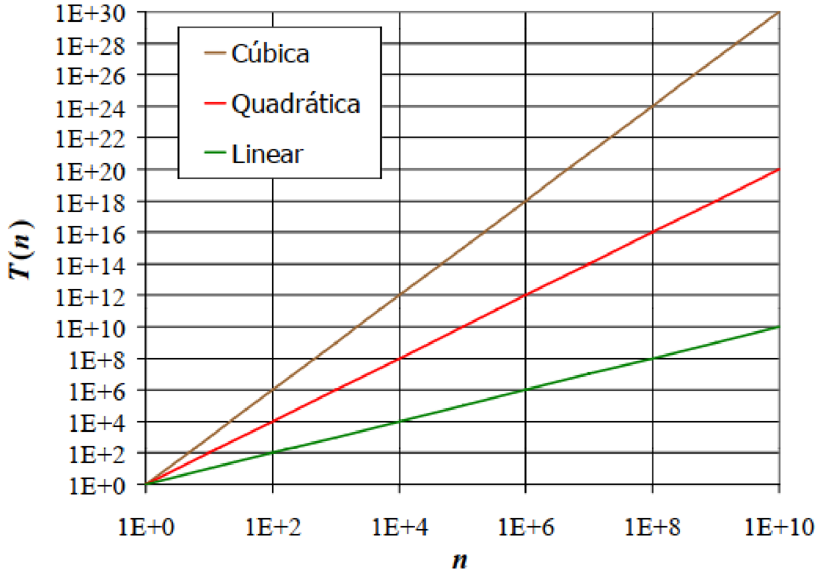
O que é complexidade?

- O que varia de ambiente para ambiente é **somente** o tempo absoluto de cada execução.

- Isso depende de fatores relacionados com o *hardware* e o *software* utilizados.

O que é complexidade?

- Exemplos de taxas de crescimento:
  - Linear ≈ n
  - Quadrática ≈ n^2
  - Cúbica ≈ n^3



- No gráfico *log-log* ao lado, a inclinação da reta corresponde à taxa de crescimento da função.

O que é complexidade?

- A taxa de crescimento **NÃO** é afetada por:
  - Fatores constantes ou;
  - Fatores de ordem mais baixa.

- Exemplos:
  - **$100n + 100000$** é uma função linear
  - **$100000n^2 + 1000000000n$** é uma função quadrática
  - **$0{,}0000000001n^3 + 1000000000000000000000n^2$** é uma função cúbica

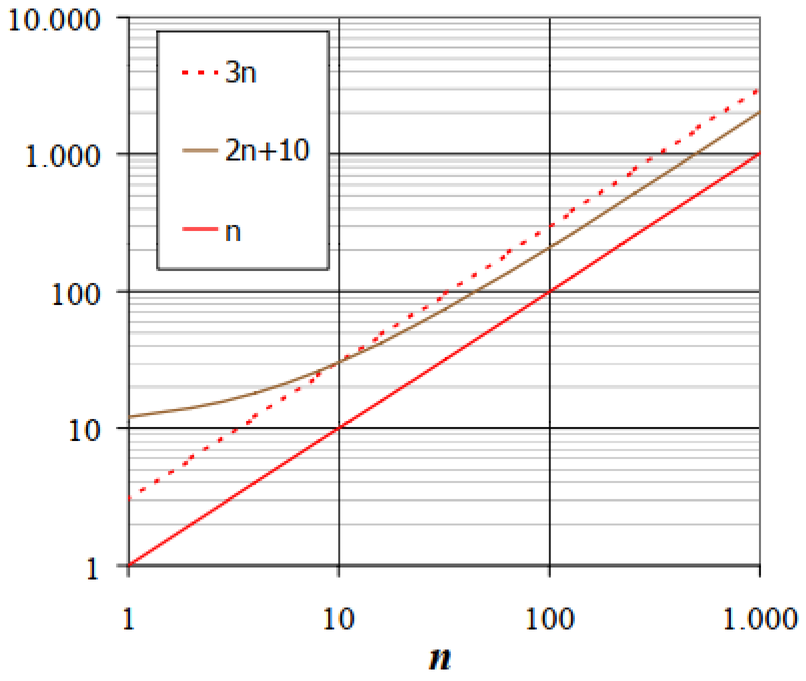
Notação O (Big-Oh)

- Dadas as funções *f(n)* e *g(n)*, dizemos que *f(n)* é *O(g(n))* se existem duas constantes positivas c e n_0 tais que $f(n) \leq c\, g(n)\ \forall n \geq n_0$

- **Exemplo**: 2n + 10 é O(n)

(i) $2n + 10 \leq cn$
(ii) $2n - cn \leq -10$
(iii) $n(2 - c) \leq -10$
(iv) $n \leq \frac{-10}{2 - c}$

- Um possível escolha é $c = 3$ e $n_0 = 10$



Notação O (Big-Oh)

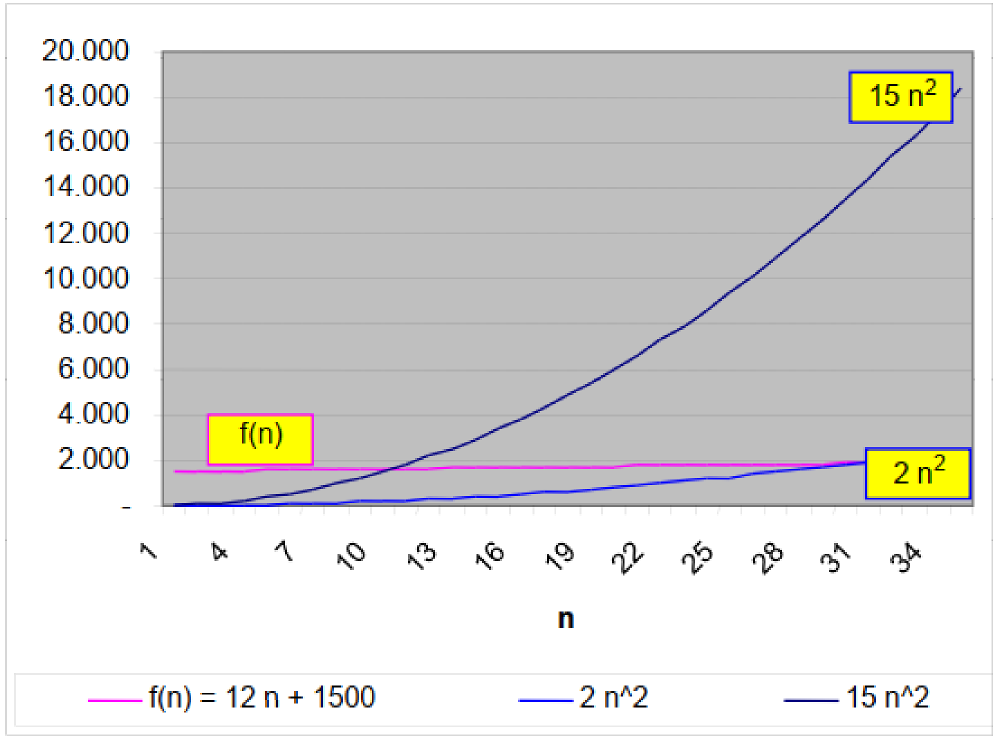
- **ATENÇÃO**: No uso da notação O, costuma ocorrer um "abuso de linguagem". O sinal de igualdade não tem o seu significado habitual!

$$\boxed{10\ n^2 + 10 \log n = O(n^2)} \qquad \boxed{2\ n^2 - 3 = O(n^2)}$$

$$\cancel{\boxed{10\ n^2 + 10 \log n = 2\ n^2 - 3}}$$

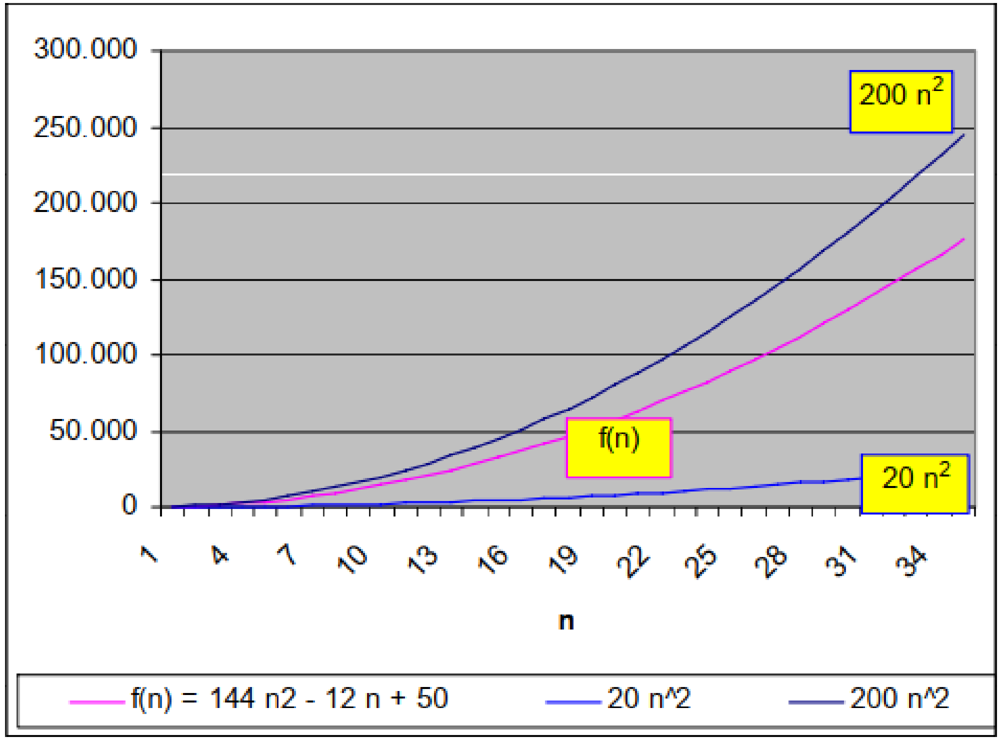
Notação O (Big-Oh) – Exemplos

$f(n) = 12n + 1500$ É $O(n^2)$?



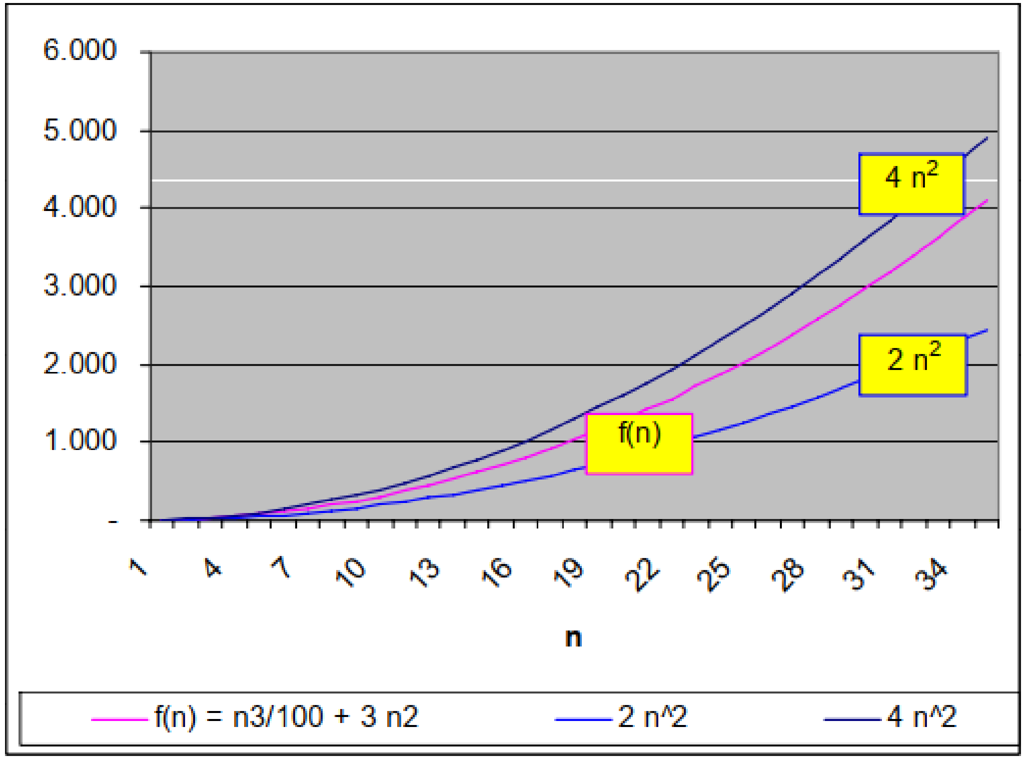
Notação O (Big-Oh) – Exemplos

$f(n) = 144\, n^2 - 12\, n + 50$ É $O(n^2)$?



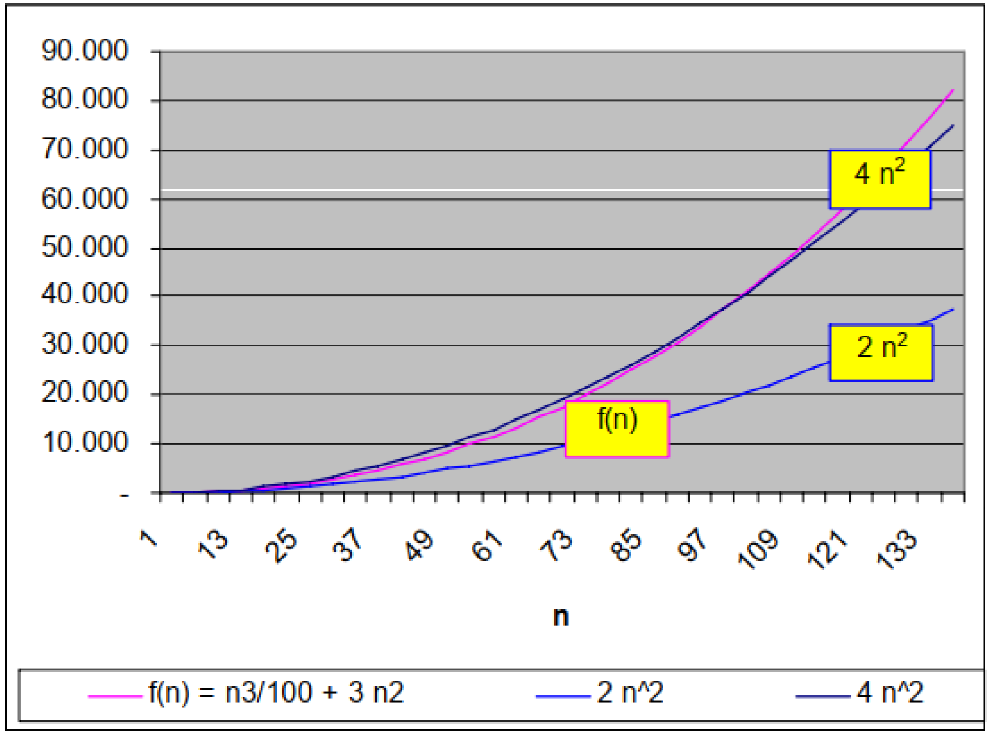
Notação O (Big-Oh) – Exemplos

$f(n) = n^3/100 + 3\,n^2$ É $O(n^2)$?



Notação O (Big-Oh) – Exemplos

$f(n) = n^3/100 + 3\ n^2$ É $O(n^2)$?



Não!!!

Notação O (Big-Oh)

- A notação O fornece um limite superior para a taxa de crescimento de uma determinada função.

- A afirmação **f(n)** é **O(g(n))** significa que a taxa de crescimento de **f(n)** não é maior que a de **g(n)**.

Notação O (Big-Oh)

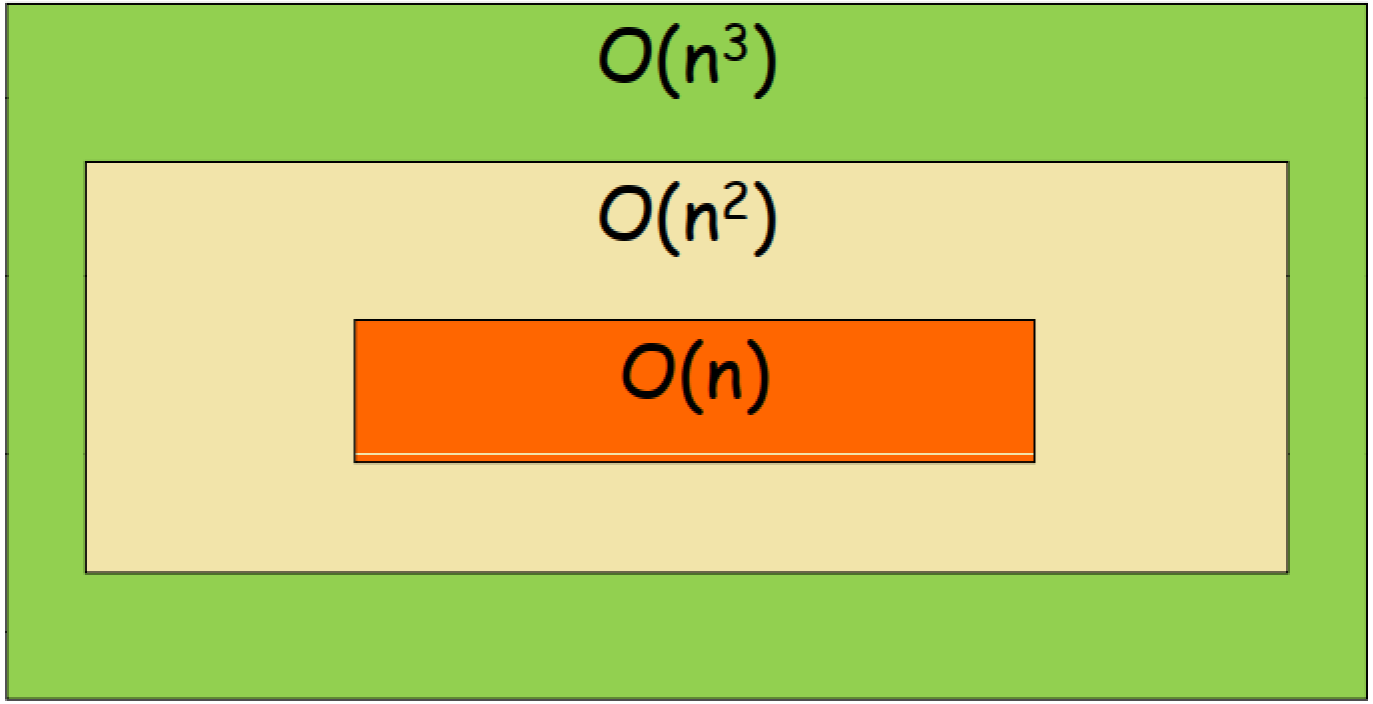
- A notação O permite ordenar as funções de acordo com as suas correspondentes taxas de crescimento.

	f(n) é O(g(n))?	g(n) é O(f(n))?
Se g(n) cresce mais que f(n):	Sim	Não
Se f(n) cresce mais que g(n):	Não	Sim
Se f(n) e g(n) têm a mesma taxa:	Sim	Sim

Notação O (Big-Oh)

- Com relação às funções polinomiais, é fácil concluir que:

$$O(n) \subset O(n^2) \subset O(n^3) \subset O(n^4) \subset O(n^5) \subset \ \dots$$



Notação O (Big-Oh)

- Se $p(n)$ é um polinômio de grau k, então $p(n)$ é O(n^k) .
  - Pode-se descartar seus termos de menor ordem, inclusive as constantes.

- Convém utilizar a menor ordem:
  - "$\mathbf{2n}$ é $\mathrm{O}(n)$" é preferível a "$\mathbf{2n}$ é $\mathrm{O}(n^2)$"
  - "$\mathbf{3n+5}$ é $\mathrm{O}(n)$" é preferível a "$\mathbf{3n+5}$ é $O(3n)$"

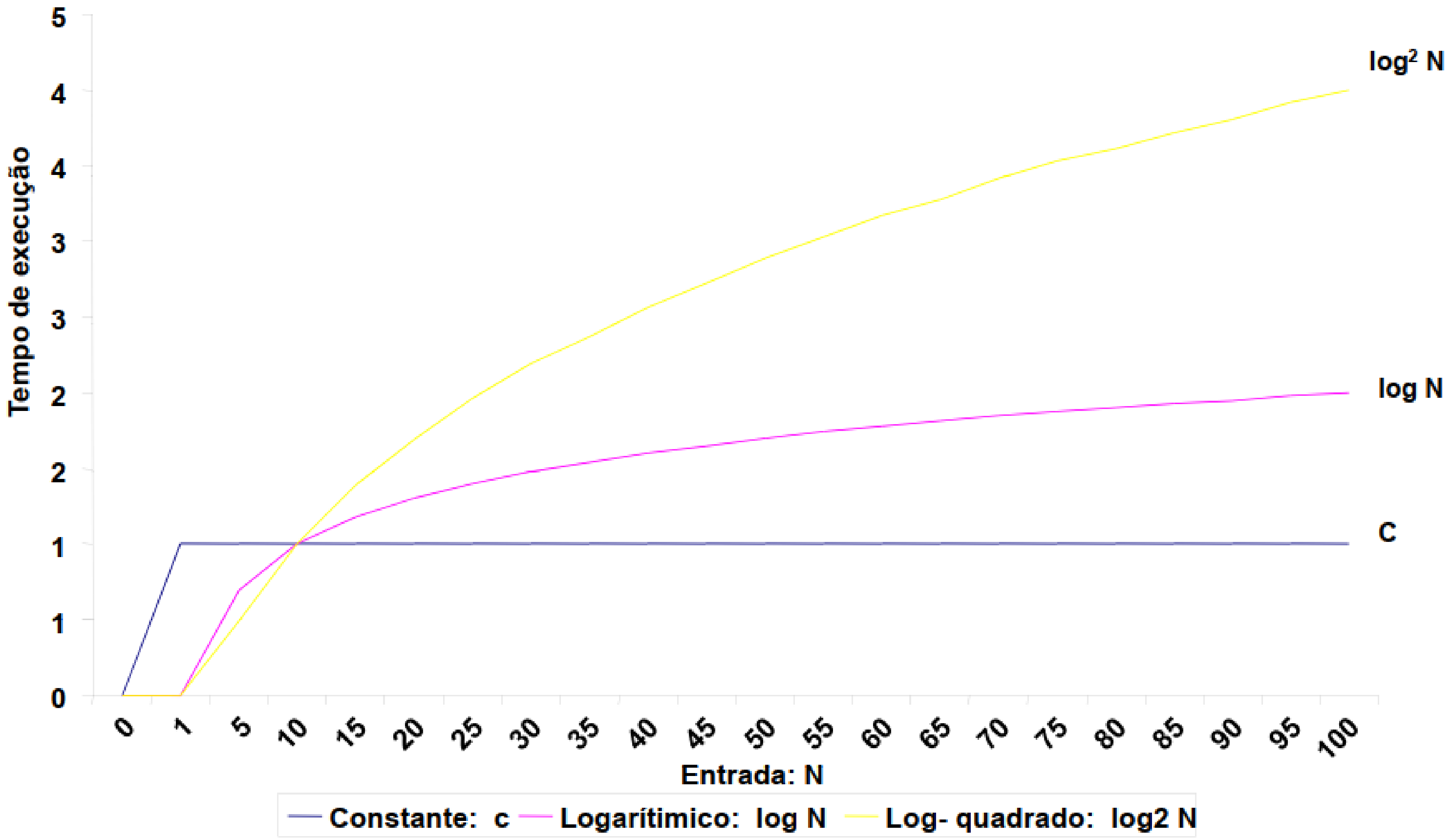
Notação O (Big-Oh) – Exemplos

$6n^4 + 12n^3 + 12$	$\in O(n^4)$	$\in O(n^5)$	$\notin O(n^3)$
$3n^2 + 12n.\log n$	$\in O(n^2)$	$\in O(n^4)$	$\notin O(n.\log n)$
$5n^2 + n(\log n)^2 + 12$	$\in O(n^2)$	$\in O(n^3)$	$\notin O(n.\log^9 n)$
$\log n + 4$	$\in O(\log n)$	$\in O(n)$	
$\log^k n, k>1$	$\in O(n)$	$\notin O(\log n)$	

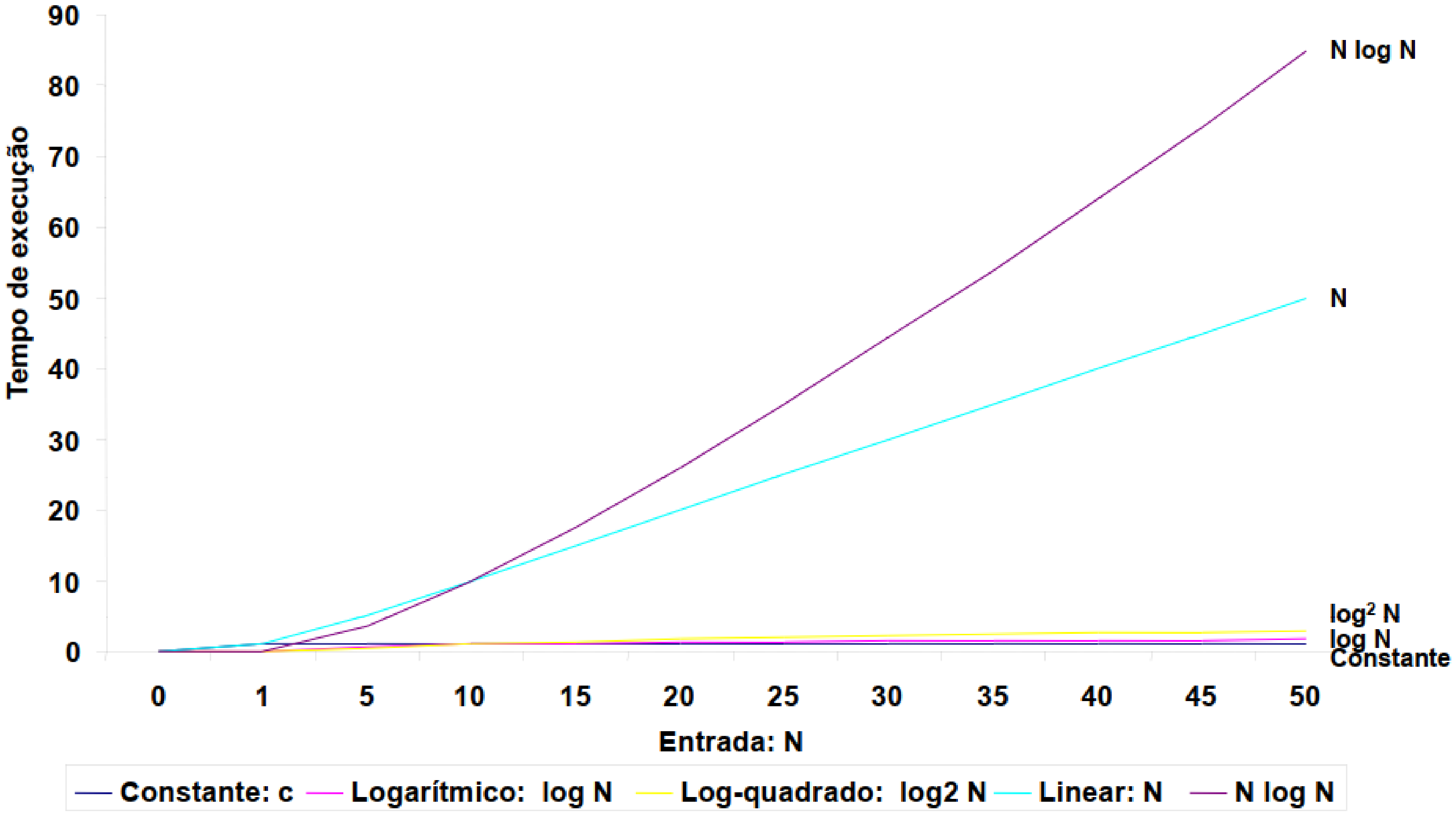
Notação O (Big-Oh)

- Além da notação O, existem também duas notações adicionais:
  - Anotação Ω (ômega)
    - Quando comparamos o limite inferior (melhor caso).
  - Anotação θ (theta)
    - Quando os limites inferiores e superiores são iguais.
    - Neste cenário, todas as execuções sempre acontecerão com a mesma complexidade (ou seja, não existe um melhor caso, caso médio ou pior caso).

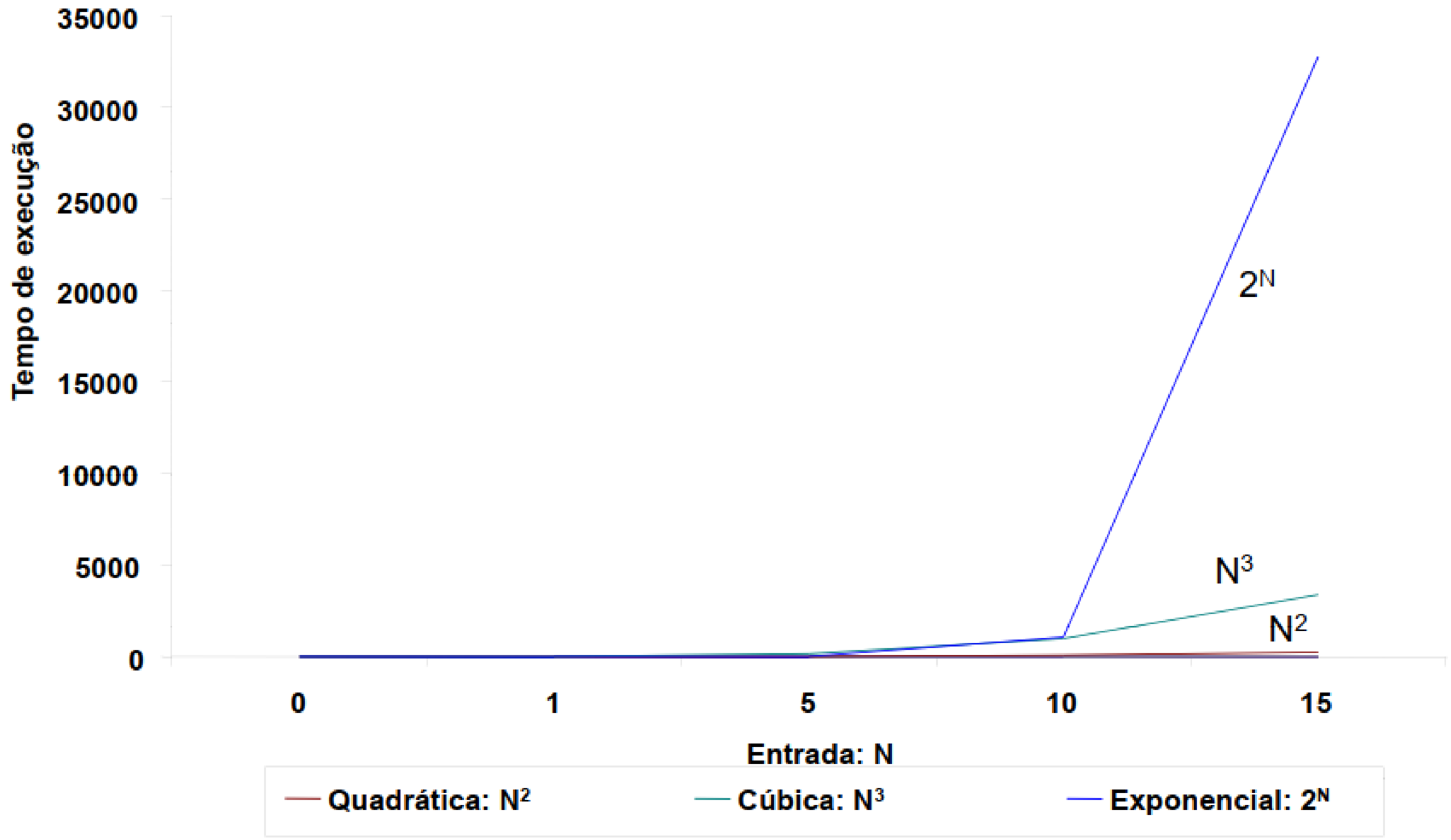
Comparação entre funções



Comparação entre funções



Comparação entre funções



Comparação entre funções

- A partir da notação O, é possível estabelecer uma hierarquia entre as funções:

Constante	O(1)
Logarítmica	O(log n)
Linear	O(n)
n.log n	O(n.log n)
Quadrática	O(n^2)
Cúbica	O(n^3)
Polinomial	O(n^k), com $k \geq 4$
Exponencial	O(k^n) , com $k > 1$
Fatorial	O(n!)

Maior Ordem

Exemplos

```c
int buscaLinear(int arr[], int n, int chave) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == chave) {
            return i;
        }
    }
    return -1;
}
```

(a)

```c
int somaArray(int arr[], int n) {
    int soma = 0;
    for (int i = 0; i < n; i++) {
        soma += arr[i];
    }
    return soma;
}
```

(b)

Qual dos dois algoritmos possui a maior complexidade?

- Ambos possuem a mesma complexidade com relação ao tempo e ao espaço que é de O(n)

Exemplos

```c
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int min_idx = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[min_idx]) {
                min_idx = j;
            }
        }
        int temp = arr[i];
        arr[i] = arr[min_idx];
        arr[min_idx] = temp;
    }
}
```

(a)

```c
void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

(b)

Qual dos dois algoritmos possui a maior complexidade?

- Ambos possuem a mesma complexidade com relação ao tempo, que é de $O(n^2)$
- Ambos possuem a mesma complexidade com relação ao espaço, que é de $O(n)$

Exemplos

```c
void encontraTripletos(int arr[], int n) {
    for (int i = 0; i < n - 2; i++) {
        for (int j = i + 1; j < n - 1; j++) {
            for (int k = j + 1; k < n; k++) {
                if (arr[i] + arr[j] + arr[k] == 0) {
                    printf("Tripletos encontrados: %d, %d, %d\n", arr[i], arr[j], arr[k]);
                }
            }
        }
    }
}
```

(a)

Qual dos dois algoritmos possui a maior complexidade?

- Ambos possuem a mesma complexidade com relação ao tempo, que é de $O(n^3)$
- O algoritmo (a) possui complexidade de espaço de $O(n)$
- O algoritmo (b) possui complexidade de espaço de $O(n^2)$

```c
void multiplicacaoMatrizes(int A[][3], int B[][3], int C[][3], int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            C[i][j] = 0;
            for (int k = 0; k < n; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}
```

(b)

Exemplos

```c
void permutacao(int arr[], int inicio, int fim) {
    if (inicio == fim) {
        for (int i = 0; i <= fim; i++) {
            printf("%d ", arr[i]);
        }
        printf("\n");
    } else {
        for (int i = inicio; i <= fim; i++) {
            int temp = arr[inicio];
            arr[inicio] = arr[i];
            arr[i] = temp;
            permutacao(arr, inicio + 1, fim);
            temp = arr[inicio];
            arr[inicio] = arr[i];
            arr[i] = temp;
        }
    }
}
```

Exemplo de um algoritmo com complexidade de tempo O(n!)

Exemplos – Torre de Hanoi

```c
void moveDisk(char from, char to) {
    printf("Move disk from %c to %c\n", from, to);
}

void towerOfHanoi(int n, char from, char to, char aux) {
    int totalMoves = pow(2, n) - 1;
    int move;
    char temp;

    if (n % 2 == 0) {
        temp = to;
        to = aux;
        aux = temp;
    }

    for (move = 1; move <= totalMoves; move++) {
        if (move % 3 == 1) {
            moveDisk(from, to);
        } else if (move % 3 == 2) {
            moveDisk(from, aux);
        } else {
            moveDisk(aux, to);
        }
    }
}
```

(a)

```c
void moveDisk(int disk, char from, char to) {
    printf("Move disk %d from %c to %c\n", disk, from, to);
}

void towerOfHanoi(int n, char from, char to, char aux) {
    if (n == 1) {
        moveDisk(1, from, to);
        return;
    }

    towerOfHanoi(n - 1, from, aux, to);
    moveDisk(n, from, to);
    towerOfHanoi(n - 1, aux, to, from);
}
```

(b)

Qual dos dois algoritmos possui a maior complexidade?

- Ambos possuem a mesma complexidade com relação ao tempo, que é de $O(2^n)$.
- Contudo, possuem complexidade com relação ao espaço diferente. A solução (a) é O(1) e a solução (b) é O(n)

Algoritmos e Estrutura de Dados II

Prof. Fellipe Guilherme Rey de Souza

Aula 01 – Complexidade de Algoritmos