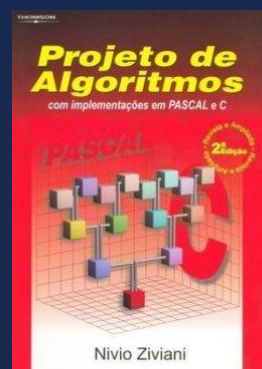
Unidade 7 - Recorrências

Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecidovfreitas@gmail.com

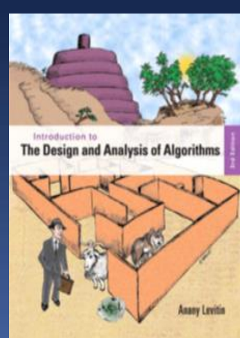
QualitSys

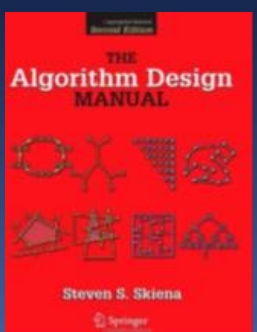
Bibliografia

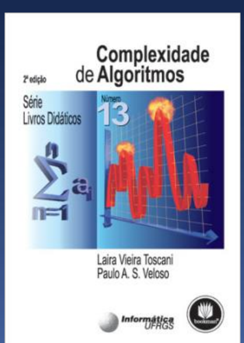
- **Algoritmos – Teoria e Prática – Cormen – Segunda Edição – Editora Campus, 2002**
- **Projeto de Algoritmos – Nivio Ziviani – Pioneira Informática - 1993**
- Algorithm Design and Applications – Michael T. Goodrich, Roberto Tamassia, Wiley, 2015
- Introduction to the Design and Analysis of Algorithms – Anany Levitin, Pearson, 2012
- The Algorithm Design Manual – Steven S. Skiena, Springer, 2008
- Complexidade de Algoritmos – Série Livros Didáticos – UFRGS
- Algorithms – Design and Analysis – Harsh Bhasin – Oxford University Press – 2015
- Notas de Aulas – Prof. Dr. Marcelo Henriques de Carvalho – UFMS – FACOM

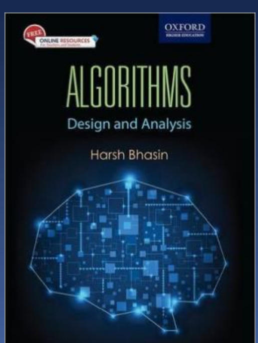


UFMS

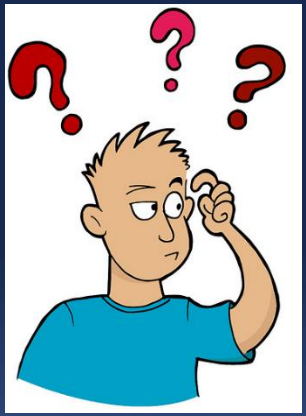








Como analisar Algoritmos Recursivos?



Recorrência

- Algoritmos **Recursivos**, em geral, são muito **elegantes**;
- Usualmente, programas funcionais baseiam-se em Recursividade;
- Nessas linguagens, as estruturas de Repetição são implementadas por Recursividade;
- Assim, é comum afirmar-se, erroneamente, que funções recursivas possuem complexidade O(1), uma vez que o programador não codifica repetições de forma explícita,
- Porém, tais algoritmos são expressos por **relações de recorrência** e, assim, a análise de tais algoritmos é feita por meio da **solução** de uma equação de **Recorrência**.

Recorrência

- Uma Recorrência é uma expressão que dá o valor de uma função em termos dos valores anteriores da mesma função.
- Assim, a análise de Algoritmos Recursivos é mais trabalhosa, uma vez que se emprega uma **Equação de Recorrência**;
- Para se analisar o consumo de tempo de um algoritmo recursivo é necessário resolver-se uma **Equação de Recorrência**;
- Na Análise de Algoritmos, uma **Equação de Recorrência** define sentenças matemáticas que corrspondem ao tempo de execução;
- Por exemplo:

$$T(n) = \begin{cases} 3 & \text{se } n = 1 \\ T(n-1) + 7, & \text{caso contrário} \end{cases}$$

O que significa Resolver uma Recorrência ?

Resolver uma Recorrência

- ✓ **Resolver uma Recorrência** significa encontrar uma **fórmula explícita** (**FÓRMULA DIRETA**) que forneça o valor da função (quantidade de operações) diretamente em termos de seu argumento (n).

Recorrência

- O exemplo clássico de Recorrência, provavelmente o mais famoso, é a fórmula de **Fibonacci**:

$$F(n) = \begin{cases} 0 & \text{se } n=0 \\ 1 & \text{se } n=1 \\ F(n-1) + F(n-2) & \text{se } n>1 \end{cases}$$

Derivando Fórmula Explícita a partir da Fórmula de Recorrência

- ✓ Embora uma fórmula de **recorrência** nos dê uma boa ideia de como um determinado termo está relacionado com o anterior, ela não nos ajuda a encontrar – de forma direta – um determinado termo sem passar pelos termos relacionados na recorrência;

- ✓ Essa fórmula explícita fornece uma melhor visão da ordem de complexidade do algoritmo. Por exemplo, **prova-se** na série de **Fibonacci** que:

$$F(n) = \begin{cases} 0, & \text{se } n=0 \\ 1, & \text{se } n=1 \\ F(n-1) + F(n-2) & \text{se } n > 1 \end{cases}$$

Fórmula Fechada

Fórmula Direta

$$F_n = \frac{1}{\sqrt{5}} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}} \cdot \left(\frac{1-\sqrt{5}}{2}\right)^n$$

Ordem de Complexidade Exponencial

Como resolver a Recorrência ?

No nosso curso, resolveremos a Equação de Recorrência usando o Método da Substituição!

Derivando Fórmula Explícita a partir da Fórmula de Recorrência

- ✓ Uma das formas mais simples de se resolver a recorrência é pelo método da **Substituição**.

Método da Substituição

- A **solução** de uma equação de recorrência por **substituição** requer uma prévia instância da fórmula para ser **substituída** na equação dada;

- O processo prossegue até sermos capaz de alcançar a condição inicial (**caso base**);

QualitSys

Exemplo

```
package br.maua;
import java.util.Scanner;

public class Recursive_01 {
        public static void main(String[] args) {
                Scanner in = new Scanner(System.in);
                Long n = in.nextLong();
                System.out.println("Resposta: "  + func(n));
                in.close();
        }
        public  static Long func(Long n) {
                if (n==1)
                        return 1L;
                return n * func(n-1);
        }
}
```

Qual a equação de Recorrência ?

```
package br.maua;
import java.util.Scanner;

public class Recursive_01 {
	public static void main(String[] args) {
		Scanner in = new Scanner(System.in);
		Long n = in.nextLong();
		System.out.println("Resposta: "  + func(n));
		in.close();
	}
	public  static Long func(Long n) {
		if (n==1)
			return 1L;
		return n * func(n-1);
	}
}
```

Qual a equação de Recorrência ?

```
package br.maua;
import java.util.Scanner;

public class Recursive_01 {
        public static void main(String[] args) {
                Scanner in = new Scanner(System.in);
                Long n = in.nextLong();
                System.out.println("Resposta: "  + func(n));
                in.close();
        }
        public  static Long func(Long n) {
                if (n==1)
                        return 1L;
                return n * func(n-1);
        }
}
```

$$T(n) = 1 \quad ; n=1$$

$$T(n) = n * T(n-1) \quad ; n>1$$

Qual a equação de Recorrência ?

- ✓ Essa recorrência está represendo a complexidade do Algoritmo;
- ✓ n representa a entrada de dados do algoritmo;
- ✓ T(n) corresponde ao tempo (quantidade de operações) em função de n.

$$T(n) = 1 \quad ; n=1$$

$$T(n) = n * T(n-1) \quad ; n>1$$

n	T(n)
1	1
2	2
3	6
4	24
5	120
6	720
7	5040

Método da Substituição - Equação de Recorrência

$T(n) = 1$, se $n = 1$;

$T(n) = n * T(n-1)$; $n > 1$

Equação de Recorrência

$T(n) = 1$, se $n = 1$;

$T(n) = n * T(n-1)$; $n > 1$

$$
\begin{aligned}
T(n) &= n * T(n-1) \\
&= n * (\ (n-1) * T(n-2)\ ) \\
&= n * (\ (n-1) * (\ (n-2) * T(n-3)\ )\ ) \\
&= \ldots
\end{aligned}
$$

Equação de Recorrência

$T(n) = 1$, se $n = 1$;

$T(n) = n * T(n-1)$; $n > 1$

$$
\begin{aligned}
T(n) &= n * T(n-1) \\
&= n * (\ (n-1) * T(n-2)\ ) \\
&= n * (\ (n-1) * (\ (n-2) * T(n-3)\ )\ ) \\
&= \ldots.
\end{aligned}
$$

$$T(n) = n * (n-1) * (n-2) * \ldots .\ k * T(k-1)$$

Equação de Recorrência

$T(n) \quad = n * (n-1) * (n-2) * \ldots k * T(k-1)$ Fórmula Geral

Esse processo de expansão termina quando atinge o caso base;

Caso Base: $(k-1) = 1$, pois $T(1) = 1$

Logo: $(k-1) = 1 \Rightarrow k = 2$

Equação de Recorrência

T(n) = n * (n-1) * (n-2) * . . . . k * T(k-1)

Esse processo de expansão termina quando atinge o caso base;

Caso Base: (k-1) = 1 , pois T(1) = 1

Logo: (k-1) = 1 => k = 2

Substituindo K na expressão acima tem-se:

T(n) = n * (n-1) * (n-2) * . . . . 2 * T(2-1)

= n * (n-1) * (n-2) * . . . . 2 * T(1)

= n * (n-1) * (n-2) * . . . . 2 * 1 = n! => O(n!)

Versão Iterativa

```
package br.maua;
import java.util.Scanner;

public class Iterative_01 {
        public static void main(String[] args) {
                Scanner in = new Scanner(System.in);
                Long n = in.nextLong();
                System.out.println("Resposta: "  + func(n));
                in.close();
        }
        public  static Long func (Long n) {
                Long result=n;
                while (n > 1L) {
                        result = result * (n - 1L);
                        n--;
                }
                return result;
        }
}
```

Versão Iterativa

```java
package br.maua;
import java.util.Scanner;

public class Iterative_01 {
        public static void main(String[] args) {
                Scanner in = new Scanner(System.in);
                Long n = in.nextLong();
                System.out.println("Resposta: "  + func(n));
                in.close();
        }
        public  static Long func (Long n) {
                Long result=n;
                while (n > 1L) {
                        result = result * (n - 1L);
                        n--;
                }
                return result;
        }
}
```

O(n)

Outro Exemplo

```java
package br.maua;
import java.util.Scanner;

public class Recursive_2 {
    public static void main(String[] args) {

        Scanner in = new Scanner(System.in);
        int n = in.nextInt();

        System.out.println(func(n));
        in.close();
    }
    public static int func (int n) {
        if (n == 1) return 5;
        return func(n-1) + 3;
    }
}
```

QualitSys

Equação de Recorrência

$$T(n) = 5 \qquad , \text{ se } n = 1;$$

$$T(n) = T(n-1) + 3 \qquad ; n > 1$$

- ✓ Essa recorrência está represendo a complexidade do Algoritmo;
- ✓ n representa a entrada de dados do algoritmo;
- ✓ T(n) corresponde ao tempo (quantidade de operações) em função de n.

n	T(n)
1	5
2	8
3	11
4	14
5	17

Equação de Recorrência

$T(n) = 5$, se $n = 1$;

$T(n) = T(n-1) + 3$; $n > 1$

QualitSys

Equação de Recorrência

$T(n) = 5$, se $n = 1$;

$T(n) = T(n-1) + 3$; $n > 1$

$T(n) = T(n-1) + 3$

$= (T(n-2) + 3) + 3$

$= (T(n-3) + 3) + 3) + 3$

....

$= T(n-k) + 3k$

Equação de Recorrência

$$T(n) = T(n-k) + 3k$$

Esse processo de expansão termina quando atinge o caso base;

Caso Base: $(n-k) = 1$, pois $T(1) = 5$

Logo: $(n-k) = 1 \Rightarrow k = n - 1$

Equação de Recorrência

$$T(n) = T(n-k) + 3k$$

Esse processo de expansão termina quando atinge o caso base;

Caso Base: $(n-k) = 1$, pois $T(1) = 5$

Logo: $(n-k) = 1 \Rightarrow k = n - 1$

Substituindo K na expressão acima tem-se:

$$
\begin{aligned}
T(n) &= T(n-k) + 3k \\
&= T(1) + 3.(n-1) \\
&= T(1) + 3n - 3 \\
&= 5 + 3n - 3 \Rightarrow \quad T(n) = 3n+2 \Rightarrow O(n) \text{ (Linear)}
\end{aligned}
$$

Recorrências – Busca Binária

✓Algoritmo de busca em arrays no qual se aplica o Paradigma **Divisão e Conquista**;

✓ Parte-se do pressuposto que o array de entrada está **ordenado**;

✓ Realizam-se **sucessivas divisões** do espaço de busca, comparando-se o elemento de pesquisa (**chave**) com o elemento na **metade** do array;

✓O processo continua, descartando-se sucessivamente as metades não convenientes até se encontrar a **chave** procurada ou encerra-se a busca **sem** sucesso;

Recorrências – Busca Binária

✓Assim, em cada iteração o algoritmo **descarta** metade do array em processamento;

✓Se denotarmos por **T(n)** o número máximo de iterações realizadas pela busca binária sobre um array com **n** elementos, tem-se:

$$T(n) = T(n/2) + 1$$

$$T(1) = 1$$

✓ Ou seja, no início do processamento, gasta-se um tempo correspondente a **1 iteração** (array inicialmente completo) acrescido do tempo necessário para buscar a chave na metade considerada a partir da segunda iteração.

Recorrências – Busca Binária

Prova-se matematicamente que :

$$\boxed{T(n) = \log_2 n}$$