

BCC202 - Estruturas de Dados I

Aula 9: Análise de Algoritmos Recursivos

Pedro Silva

Universidade Federal de Ouro Preto – UFOP
Departamento de Computação – DECOM
Email: silvap@ufop.edu.br



Conteúdo

Definindo um problema

Recursividade

Análise de Complexidade de funções recursivas

Equação de Recorrência

Exemplos

Exemplo usando fatorial

Mais um exemplo

Algumas recorrências básicas

Tarefas

Considerações Finais

Exercícios

Bibliografia

Definindo um problema

Qual eu devo escolher considerando o número de comparações entre as chaves do vetor?

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

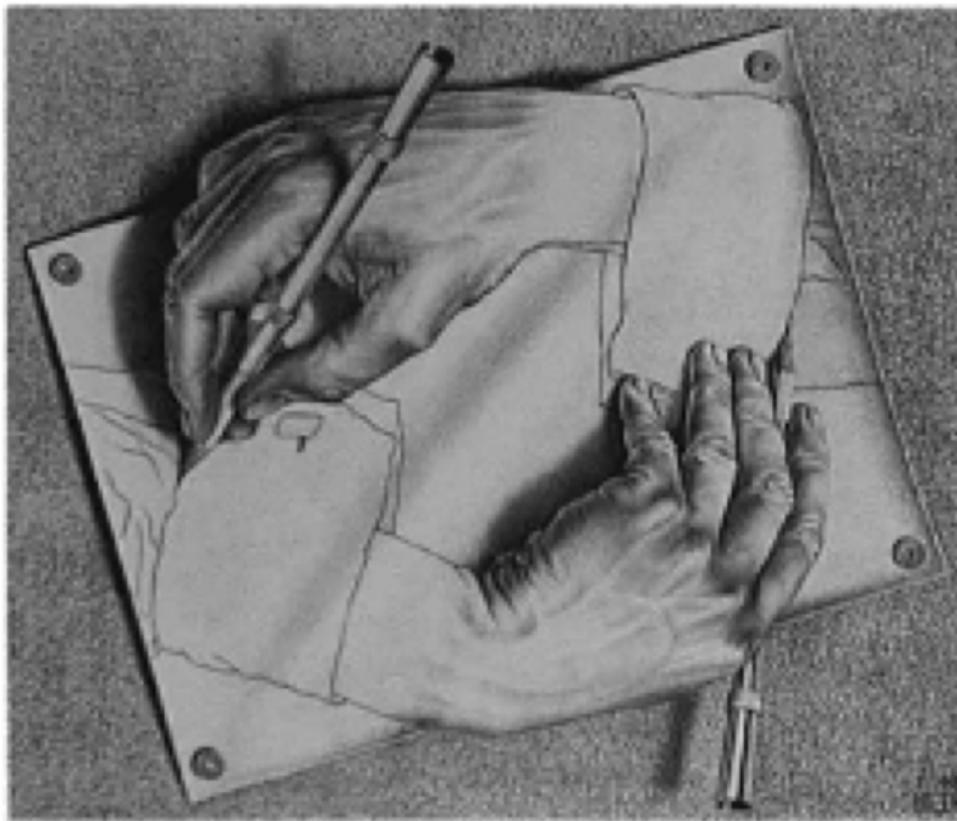
```
int maior(int *v, int n) {  
    if (n == 1)  
        return v[0];  
    int m = maior (v, n - 1);  
    if (m < v[n-1])  
        return v[n-1];  
    return m;  
}
```

Qual eu devo escolher considerando o número de comparações entre as chaves do vetor?

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

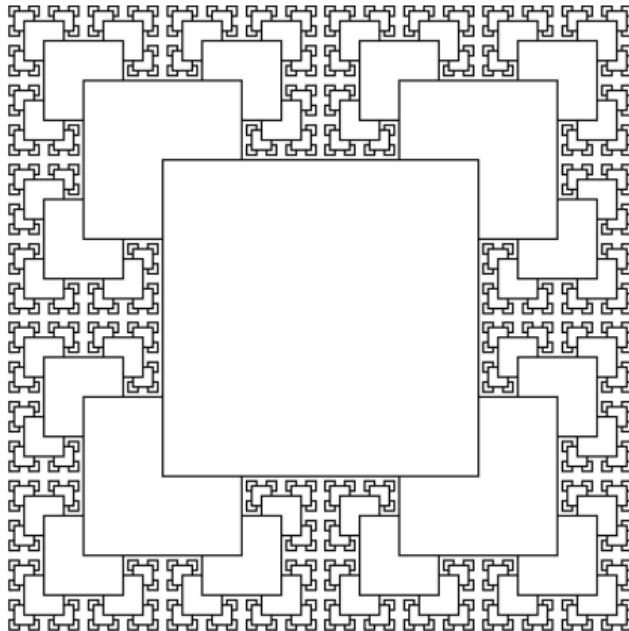
```
int maior(int *v, int n) {  
    if (n == 1)  
        return v[0];  
    int m = maior (v, n - 1);  
    if (m < v[n-1])  
        return v[n-1];  
    return m;  
}
```

Como encontrar a função de complexidade de um código recursivo?



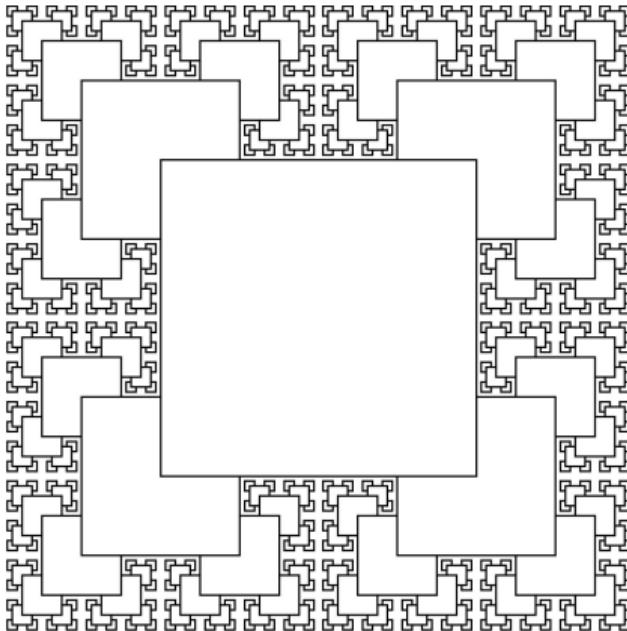
Recursividade

Construindo uma estrela



```
void estrela1(int x, int y, int r) {  
    if (r>0) {  
        estrela1(x - r, y + r, r / 2);  
        estrela1(x + r, y + r, r / 2);  
        estrela1(x - r, y - r, r / 2);  
        estrela1(x + r, y - r, r / 2);  
        box(x, y, r);  
    }  
}
```

OOutra implementação da construção de uma estrela



```
void estrela2(int x, int y, int r) {  
    if (r>1) {  
        estrela2(x - r, y + r, r / 2);  
        estrela2(x + r, y + r, r / 2);  
        estrela2(x - r, y - r, r / 2);  
        estrela2(x + r, y - r, r / 2);  
    }  
    box(x, y, r);  
}
```

Qual a melhor implementação?

```
void estrela1(int x, int y, int r) {  
    if (r>0) {  
        estrela1(x - r, y + r, r / 2);  
        estrela1(x + r, y + r, r / 2);  
        estrela1(x - r, y - r, r / 2);  
        estrela1(x + r, y - r, r / 2);  
        box(x, y, r);  
    }  
}
```

```
void estrela2(int x, int y, int r) {  
    if (r>1) {  
        estrela2(x - r, y + r, r / 2);  
        estrela2(x + r, y + r, r / 2);  
        estrela2(x - r, y - r, r / 2);  
        estrela2(x + r, y - r, r / 2);  
    }  
    box(x, y, r);  
}
```

Análise de Complexidade de funções recursivas

Análise de Complexidade - Notação O

- ▶ Define-se uma função de complexidade $f(n)$ desconhecida:
 - ▶ n mede o tamanho dos argumentos para o procedimento em questão.
- ▶ Identifica-se a **equação de recorrência** $T(n)$:
 - ▶ Especifica-se $T(n)$ como uma função dos **termos anteriores**.
 - ▶ Especifica-se a **condição de parada** (e.g. $T(1)$).

Recursão de exemplo

```
void exemplo(int n) {  
    int i;  
    if(n <= 1)  
        printf("%d", n);  
    else {  
        for(i = 0; i < n; i++)  
            printf("%d", n);  
        exemplo(n-1);  
    }  
}
```

Equação de Recorrência: Passo a Passo

- ▶ Define-se uma função de complexidade $f(n)$.
- ▶ Identifica-se a equação de recorrência $T(n)$:
 - ▶ Especifica-se $T(n)$ como uma função dos termos anteriores.
 - ▶ Especifica-se a **condição de parada** (e.g. $T(1)$).

Exemplo de recorrência

```
void exemplo(int n) {  
    int i;  
    if(n <= 1)  
        printf("%d", n);  
    else {  
        for(i = 0; i < n; i++)  
            printf("%d", n);  
        exemplo(n-1);  
    }  
}
```

Como podemos definir a função de recorrência?

Exemplo de recorrência

```
void exemplo(int n) {  
    int i;  
    if(n <= 1)  
        printf("%d", n);  
    else {  
        for(i = 0; i < n; i++)  
            printf("%d", n);  
        exemplo(n-1);  
    }  
}
```

Como podemos definir a função de recorrência?

$$\begin{cases} T(n) = n + T(n-1) \\ T(1) = 1 \end{cases}$$

Resolvendo o exemplo de recorrência

$$\begin{cases} T(n) = n + T(n - 1) \\ T(1) = 1 \end{cases}$$

Expandindo:

$$\begin{aligned} T(n) &= n + T(n - 1) \\ &= n + (n - 1) + T(n - 2) \\ &= n + (n - 1) + (n - 2) + T(n - 3) \\ &\quad \vdots \\ &= n + (n - 1) + (n - 2) + \dots + 2 + T(1) \\ &= n + (n - 1) + (n - 2) + \dots + 2 + 1 \end{aligned}$$

Resolvendo o exemplo de recorrência

$$T(n) = n + (n - 1) + (n - 2) + \dots + 2 + 1$$

$$\begin{aligned}2T(n) &= n + (n - 1) + (n - 2) + \dots + 2 + 1 + \\&\quad 1 + 2 + 3 + \dots + (n - 1) + n\end{aligned}$$

$$2T(n) = (n + 1) + (n + 1) + (n + 1) + \dots + (n + 1)$$

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

Resolvendo o exemplo de recorrência

$$T(n) = n + (n - 1) + (n - 2) + \dots + 2 + 1$$

$$\begin{aligned}2T(n) &= n + (n - 1) + (n - 2) + \dots + 2 + 1 + \\&\quad 1 + 2 + 3 + \dots + (n - 1) + n\end{aligned}$$

$$2T(n) = (n + 1) + (n + 1) + (n + 1) + \dots + (n + 1)$$

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

Logo: $T(n) = \frac{n(n + 1)}{2}$

Resolvendo o exemplo de recorrência

$$T(n) = n + (n - 1) + (n - 2) + \dots + 2 + 1$$

$$\begin{aligned} 2T(n) &= n + (n - 1) + (n - 2) + \dots + 2 + 1 + \\ &\quad 1 + 2 + 3 + \dots + (n - 1) + n \end{aligned}$$

$$2T(n) = (n + 1) + (n + 1) + (n + 1) + \dots + (n + 1)$$

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

Logo: $T(n) = \frac{n(n + 1)}{2} = O(n^2)$

Exemplos

Exemplo usando fatorial

```
int fatorial(int n) {  
    if(n == 1)  
        return 1;  
    else {  
        return n * fatorial(n-1);  
    }  
}
```

Podemos definir a recorrência como:

$$\begin{cases} T(n) = 1 + T(n-1) \\ T(1) = 1 \end{cases}$$

Fatorial: Análise da função

$$\begin{cases} T(n) = 1 + T(n - 1) \\ T(1) = 1 \end{cases}$$

Expandindo:

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

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

$$T(n - 2) = 1 + T(n - 3)$$

⋮

$$T(3) = 1 + T(2)$$

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

$$T(1) = 1$$

Fatorial: Análise da função

$$\begin{cases} T(n) = 1 + T(n-1) \\ T(1) = 1 \end{cases}$$

Expandindo:

$$\begin{aligned} T(n) &= 1 + T(n-1) \\ &= 1 + 1 + T(n-2) \\ &\vdots \\ &= 1 + 1 + 1 + \dots + 1 + T(1) \\ &= 1 + 1 + 1 + \dots + 1 + 1 \\ &= n \end{aligned}$$

Logo: $T(n) = n$

Fatorial: Análise da função

$$\begin{cases} T(n) = 1 + T(n-1) \\ T(1) = 1 \end{cases}$$

Expandindo:

$$\begin{aligned} T(n) &= 1 + T(n-1) \\ &= 1 + 1 + T(n-2) \\ &\vdots \\ &= 1 + 1 + 1 + \dots + 1 + T(1) \\ &= 1 + 1 + 1 + \dots + 1 + 1 \\ &= n \end{aligned}$$

Logo: $T(n) = n = O(n)$

- ▶ **Atenção:** lembre-se de que, além da análise de custo de **tempo**, deve-se analisar também o custo de **espaço**.
- ▶ Qual a complexidade de espaço da função factorial (qual o tamanho da pilha de execução)?
 - ▶ Proporcional ao número de chamadas?

Somatórios úteis

$$\sum_{i=m}^n 1 = n + 1 - m$$

$$\sum_{i=m}^n i = \frac{n(n+1)}{2}$$

$$\sum_{i=m}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=m}^n x^i = \frac{x(x^n - x^{m-1})}{x - 1}$$

$$\sum_{i=0}^n 2^i = 2^{n+1} - 1$$

$$\sum_{i=0}^n \frac{1}{2^i} = 2 - \frac{1}{2^n}$$

$$\sum_{i=1}^n a^i = \frac{a^{n+1} - 1}{a - 1}$$

$$\sum_{i=0}^{k-1} q^i = a_1 \frac{1 - q^n}{1 - q}$$

Mais um exemplo

Considere a seguinte função:

```
void func(int n) {  
    if (n <= 1)  
        printf("*\\n");  
    else {  
        for (int i=0;i<n;i++)  
            printf("*");  
        printf("\\n");  
        func(n/3);  
    }  
}
```

Detalhando a Equação de Recorrência

Qual a equação de recorrência?

Detalhando a Equação de Recorrência

Qual a equação de recorrência?

$$\begin{cases} T(n) = n + T(n/3) \\ T(1) = 1 \end{cases}$$

Detalhando a Equação de Recorrência

Qual a equação de recorrência?

$$\begin{cases} T(n) = n + T(n/3) \\ T(1) = 1 \end{cases}$$

- ▶ Resolva a equação de recorrência por expansão.
 - ▶ Considere a simplificação de que n seja **sempre divisível por 3**. Ou seja, $n = 3^k$, $k \geq 0$.
 - ▶ **Dica:** Somatório de uma PG finita = $a_1 \frac{1 - q^n}{1 - q}$, onde n = número de termos da PG.

Resolvendo a Equação de Recorrência

$$\begin{cases} T(n) = n + T(n/3) \\ T(1) = 1 \end{cases}$$

Resolvendo por expansão:

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

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

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

⋮

$$T(n/3/3.../3) = n/3/3.../3 + T(n/3/3.../3)$$

Resolvendo a Equação de Recorrência

$$\begin{cases} T(n) = n + T(n/3) \\ T(1) = 1 \end{cases}$$

Resolvendo por expansão:

$$\begin{aligned} T(n) &= n + T(n/3) \\ &= n + n/3 + T(n/3/3) \\ &= n + n/3 + n/3/3 + T(n/3/3/3) \\ &\vdots \\ &= n + n/3 + n/3/3 + \dots + n/3/3/\dots/3 + T(n/3/3/\dots/3) \end{aligned}$$

Resolvendo a Equação de Recorrência

Pela expansão chegamos a:

$$\begin{aligned}T(n) &= n + n/3 + n/3/3 + \dots + n/3/3/\dots/3 + T(n/3/3/\dots/3) \\&= \frac{n}{3^0} + \frac{n}{3^1} + \frac{n}{3^2} + \dots + \frac{n}{3^{k-1}} + \frac{n}{3^k}\end{aligned}$$

Considerando que:

$$1 = \frac{n}{3^k} \Rightarrow n = 3^k$$

Tem-se que:

$$T\left(\frac{n}{3^k}\right) = T(1)$$

Resolvendo a Equação de Recorrência

Considerando que $T\left(\frac{n}{3^k}\right) = T(1)$, temos:

$$T(n) = \sum_{i=0}^{k-1} \left(\frac{n}{3^i}\right) + T(1) = n \sum_{i=0}^{k-1} (1/3^i) + 1$$

Mas, $\sum_{i=0}^{k-1} \left(\frac{1}{3^i}\right)$ é uma PG finita, com $a_1 = 1$, $q = \frac{1}{3}$ e $n = 3^k$.

Resolvendo a Equação de Recorrência

Tem-se:

Dado: $T(n) = \sum_{i=0}^{k-1} (1/3^i) + 1$

$$T(n) = n \frac{1 - \left(\frac{1}{3}\right)^k}{1 - \frac{1}{3}} + 1$$

Sabe-se que:

- Somatório de uma PG finita é

$$\sum_{i=0}^{k-1} q^i = a_1 \frac{1 - q^n}{1 - q}.$$

- Com $a_1 = 1$, $q = \frac{1}{3}$ e $n = 3^k$.

Resolvendo a Equação de Recorrência

Dado: $T(n) = \sum_{i=0}^{k-1} (1/3^i) + 1$

Sabe-se que:

- Somatório de uma PG finita é

$$\sum_{i=0}^{k-1} q^i = a_1 \frac{1 - q^n}{1 - q}.$$

- Com $a_1 = 1$, $q = \frac{1}{3}$ e $n = 3^k$.

Tem-se:

$$\begin{aligned} T(n) &= n \frac{1 - \left(\frac{1}{3}\right)^k}{1 - \frac{1}{3}} + 1 \\ &= n \frac{1 - \frac{1}{n}}{1 - \frac{1}{3}} + 1 \end{aligned}$$

Resolvendo a Equação de Recorrência

Dado: $T(n) = \sum_{i=0}^{k-1} (1/3^i) + 1$

Sabe-se que:

- Somatório de uma PG finita é

$$\sum_{i=0}^{k-1} q^i = a_1 \frac{1 - q^n}{1 - q}.$$

- Com $a_1 = 1$, $q = \frac{1}{3}$ e $n = 3^k$.

Tem-se:

$$\begin{aligned} T(n) &= n \frac{1 - \left(\frac{1}{3}\right)^k}{1 - \frac{1}{3}} + 1 \\ &= n \frac{1 - \frac{1}{n}}{1 - \frac{1}{3}} + 1 \\ &= \frac{n-1}{\frac{2}{3}} + 1 \end{aligned}$$

Resolvendo a Equação de Recorrência

Dado: $T(n) = \sum_{i=0}^{k-1} (1/3^i) + 1$

Sabe-se que:

- Somatório de uma PG finita é

$$\sum_{i=0}^{k-1} q^i = a_1 \frac{1 - q^n}{1 - q}.$$

- Com $a_1 = 1$, $q = \frac{1}{3}$ e $n = 3^k$.

Tem-se:

$$\begin{aligned} T(n) &= n \frac{1 - \left(\frac{1}{3}\right)^k}{1 - \frac{1}{3}} + 1 \\ &= n \frac{1 - \frac{1}{n}}{1 - \frac{1}{3}} + 1 \\ &= \frac{n-1}{\frac{2}{3}} + 1 \\ &= \frac{3n}{2} - \frac{1}{2} \end{aligned}$$

Resolvendo a Equação de Recorrência

Dado: $T(n) = \sum_{i=0}^{k-1} (1/3^i) + 1$

Sabe-se que:

- Somatório de uma PG finita é

$$\sum_{i=0}^{k-1} q^i = a_1 \frac{1 - q^n}{1 - q}.$$

- Com $a_1 = 1$, $q = \frac{1}{3}$ e $n = 3^k$.

Tem-se:

$$\begin{aligned} T(n) &= n \frac{1 - \left(\frac{1}{3}\right)^k}{1 - \frac{1}{3}} + 1 \\ &= n \frac{1 - \frac{1}{n}}{1 - \frac{1}{3}} + 1 \\ &= \frac{n-1}{\frac{2}{3}} + 1 \\ &= \frac{3n}{2} - \frac{1}{2} \end{aligned}$$

Portanto, $T(n) = O(n)$

Algumas recorrências básicas

Algumas recorrências básicas

$$\begin{cases} T(n) = T\left(\frac{n}{2}\right) + 1 & (n \geq 2) \\ T(1) = 0 & (n = 1) \end{cases}$$

Algumas recorrências básicas

$$\begin{cases} T(n) = T\left(\frac{n}{2}\right) + 1 & (n \geq 2) \\ T(1) = 0 & (n = 1) \end{cases}$$

Vamos supor que:

$$n = 2^k \Rightarrow k = \log n$$

Algumas recorrências básicas

Resolvendo por expansão temos:

$$\begin{cases} T(n) = T\left(\frac{n}{2}\right) + 1 & (n \geq 2) \\ T(1) = 0 & (n = 1) \end{cases}$$

Vamos supor que:

$$n = 2^k \Rightarrow k = \log n$$

$$\begin{aligned} T(n) &= T(2^{k-1}) + 1 \\ &= (T(2^{k-2}) + 1) + 1 \\ &= (T(2^{k-3}) + 1) + 1 + 1 \\ &\quad \vdots \\ &= (T(2) + 1) + 1 + \dots + 1 \\ &= (T(1) + 1) + 1 + \dots + 1 \\ &= 0 + \underbrace{1 + 1 + \dots + 1}_k \\ &= k \end{aligned}$$

Algumas recorrências básicas

Resolvendo por expansão temos:

$$\begin{cases} T(n) = T\left(\frac{n}{2}\right) + 1 & (n \geq 2) \\ T(1) = 0 & (n = 1) \end{cases}$$

Vamos supor que:

$$n = 2^k \Rightarrow k = \log n$$

$$\begin{aligned} T(n) &= T(2^{k-1}) + 1 \\ &= (T(2^{k-2}) + 1) + 1 \\ &= (T(2^{k-3}) + 1) + 1 + 1 \\ &\quad \vdots \\ &= (T(2) + 1) + 1 + \dots + 1 \\ &= (T(1) + 1) + 1 + \dots + 1 \\ &= 0 + \underbrace{1 + 1 + \dots + 1}_k \\ &= k \end{aligned}$$

$$T(n) = \log n \Rightarrow O(\log n)$$

Outro exemplo - Pesquisa Binária

```
int pesquisa(int* vetor, int x, int ini, int fim) {  
    if (ini > fim)  
        return -1;  
  
    int meio = (ini + fim) / 2;  
  
    if (vetor[meio] == x)  
        return meio;  
    else if (vetor[meio] < x)  
        return pesquisa(vetor, x, meio + 1, fim);  
    else  
        return pesquisa(vetor, x, ini, meio - 1);  
}
```

Tarefa 1

- ▶ Implemente uma função recursiva que calcula a potência de um número:
 - ▶ Qual a condição de parada?

Tarefa 1

- ▶ Implemente uma função recursiva que calcula a potência de um número:
 - ▶ Qual a condição de parada?
- ▶ Apresente a equação de recorrência e resolva-a.

Tarefa 1

- ▶ Implemente uma função recursiva que calcula a potência de um número:
 - ▶ Qual a condição de parada?
- ▶ Apresente a equação de recorrência e resolva-a.
- ▶ Qual a complexidade de tempo desta função?

Tarefa 1 - Resolução

```
int pot(int base, int exp) {  
    if (exp == 0)  
        return 1;  
    /* else */  
    return (base * pot(base, exp - 1));  
}
```

Análise de complexidade:

$$\begin{cases} T(b, 0) = 1 \\ T(b, n) = 1 + T(b, n - 1) \end{cases}$$

Tarefa 1 - Resolução

```
int pot(int base, int exp) {  
    if (exp == 0)  
        return 1;  
    /* else */  
    return (base * pot(base, exp - 1));  
}
```

Análise de complexidade:

$$\begin{cases} T(b, 0) = 1 \\ T(b, n) = 1 + T(b, n - 1) \end{cases}$$

$$O(n)$$

Tarefa 2

- ▶ Implemente uma função recursiva para computar o valor de 2^n .
 - ▶ Qual a condição de parada.

Tarefa 2

- ▶ Implemente uma função recursiva para computar o valor de 2^n .
 - ▶ Qual a condição de parada.
- ▶ Apresente a equação de recorrência e resolva-a.

Tarefa 2

- ▶ Implemente uma função recursiva para computar o valor de 2^n .
 - ▶ Qual a condição de parada.
- ▶ Apresente a equação de recorrência e resolva-a.
- ▶ Qual a complexidade de tempo desta função?

Tarefa 2 - Resolução

```
int pot2(int exp) {  
    if (exp == 0)  
        return 1;  
    /* else */  
    return 2 * pot2(base, exp - 1);  
}
```

Análise de complexidade:

$$\begin{cases} T(0) = 1 \\ T(n) = 1 + T(n-1) \end{cases}$$

Tarefa 2 - Resolução

```
int pot2(int exp) {  
    if (exp == 0)  
        return 1;  
    /* else */  
    return 2 * pot2(base, exp - 1);  
}
```

Análise de complexidade:

$$\begin{cases} T(0) = 1 \\ T(n) = 1 + T(n-1) \end{cases}$$

$$O(n)$$

Tarefa 3

Indique e resolva a equação de recorrência da função abaixo. Qual a complexidade assintótica?

```
void func(int a, int b) {  
    if (b==0)  
        return a;  
    return func(a + a, b - 1);  
}
```

Considerações Finais

Conclusão

- ▶ Revisão de conceitos de **recursividade**.
- ▶ Noção geral sobre complexidade de funções recursivas por meio de **equações de recorrência**.

Listas.

Exercícios

Exercício 01

- ▶ Crie uma função recursiva que calcula a média de um vetor.
 - ▶ Qual a condição de parada?
 - ▶ Qual a complexidade de sua função? Apresente a equação de recorrência e resolva-a.

Exercício 02

Indique o algoritmo e a equação de recorrência do algoritmo de Euclides (calcula o MDC - Máximo Divisor Comum - de dois números a e b).

Exercício 03

Resolva a seguinte equação de recorrência:

$$\begin{cases} T(n) = T(n/2) + 1 \\ T(1) = 0 \end{cases}$$

Exercício 04

Descubra o que faz o algoritmo abaixo.

```
int algoritmo(int a, int b) {  
    if(b == 0)  
        return a;  
    else  
        return algoritmo(b, a % b);  
}
```

Bibliografia

Bibliografia

Os conteúdos deste material, incluindo figuras, textos e códigos, foram extraídos ou adaptados do livro-texto indicado a seguir:

-  **Celes, Waldemar and Cerqueira, Renato and Rangel, José**
Introdução a Estruturas de Dados com Técnicas de Programação em C.
Elsevier Brasil, 2016. ISBN 978-85-352-8345-7.
-  **Anany Levitin.**
Introduction To Design And Analysis Of Algorithms.
3rd Edition, 2012.
-  **T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein.**
Introduction to Algorithms.
The MIT Press, 3rd edition, 2009. ISBN-13: 978-0-262-53305-8.