Análise de Algoritmos e Complexidade da Computação

Design e Análise de Algoritmos por Indução

Prof. Dr. Osvaldo Luiz de Oliveira

Estas anotações devem ser complementadas por apontamentos em aula.

Verificação exaustiva de um teorema

Seja T um teorema que nós queremos verificar a validade para n, inteiro, $1 \leq n \leq m$. Uma verificação exaustiva consiste em verificar a validade do teorema para:

- T (1);
- T(2);
- ...
- T (m).

Quais a limitações desta abordagem?

Princípio da indução matemática

Seja *T* um teorema que nós queremos provar a validade para *n*, inteiro, $n \geq 1$. Para provar que T vale para todo *n*:

1. Nós verificamos se T(1) é verdadeiro;

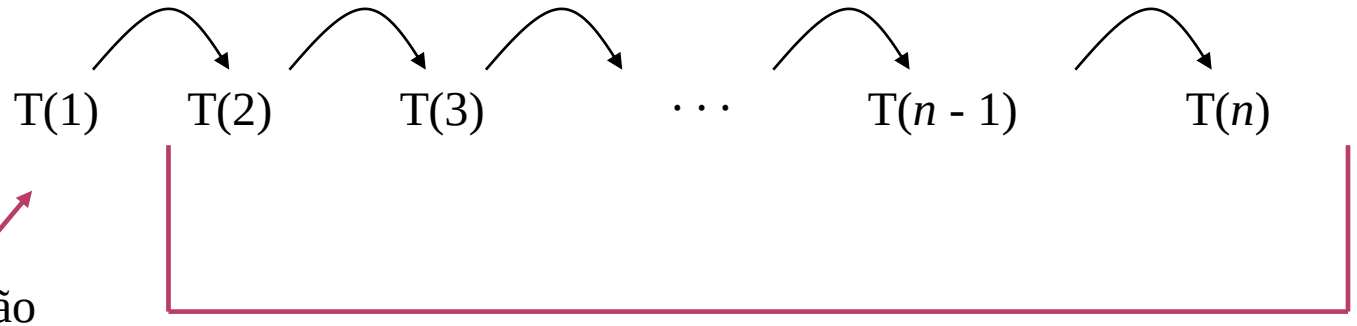
2. Admitindo, por hipótese, que T ($n - 1$) é verdadeiro ($n \geq 2$), nós provamos que a validade de T(n - 1) implica que T(n) é verdadeiro.

base

hipótese de indução

Prova do "motor de indução" que permite admitir que o fato do teorema valer para n - 1 implica que ele vale para n.

Efeito dominó



verificação de que T(1) (base) é verdade

hipótese de que T(n – 1) é verdade ($n \geq 2$)
+
prova de que a verdade de T(n – 1) implica que T(n) é verdade

Soma dos *n* primeiros naturais

Exemplo, prova do seguinte:

Teorema

A soma $1 + 2 + 3 + \ldots + n$ ($n \geq 1$, inteiro) é

$$S(n) = \frac{n(n+1)}{2}.$$

Solução

i) Base

Para $n = 1$ a soma é igual a 1.
Pela fórmula $S(1) = 1 \cdot (1 + 1) / 2 = 1$.
Logo, o Teorema é verdade para $n = 1$.

ii) Hipótese de indução

O Teorema afirma que $S(n) = n\,(n + 1) / 2$.

Admitimos, por hipótese, que
$S(n - 1) = (n - 1)\, n / 2$ é verdade.

Solução

iii) Relação entre S(*n*) e S(*n* – 1)

$S(n) = \quad 1 + 2 + \ldots + n - 1 + n.$

$S(n - 1) = 1 + 2 + \ldots + n - 1.$

Logo, $S(n) = S(n - 1) + n.$

Hipótese de indução

$S(n - 1) = (n - 1)\, n / 2$

iv) Prova de que S(*n* – 1) implica em S(*n*)

$S(n) = S(n - 1) + n.$

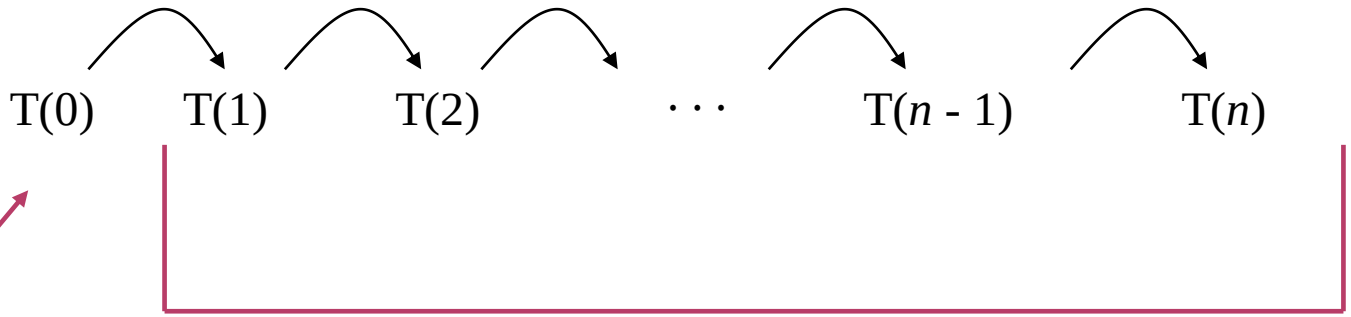
Usando a hipótese de indução, $S(n) = (n - 1)\, n / 2 + n.$

Logo, $S(n) = (n^2 - n + 2n) / 2 = (n^2 + n) / 2$

Ou seja, $S(n) = n\,(n + 1) / 2.$ ■

Variantes do princípio de indução descrito

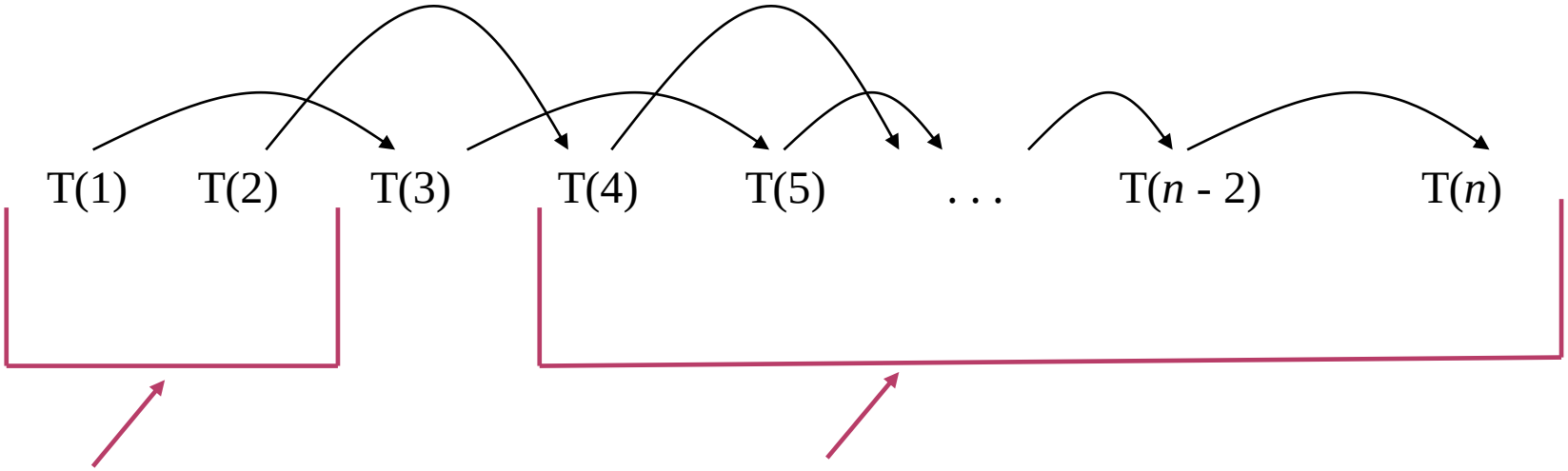
Base não necessariamente para $n = 1$



verificação de que T(0) (base) é verdade

hipótese de que T(n – 1) é verdade ($n \geq 1$)
+
prova de que a verdade de T(n – 1) implica que T(n) é verdade

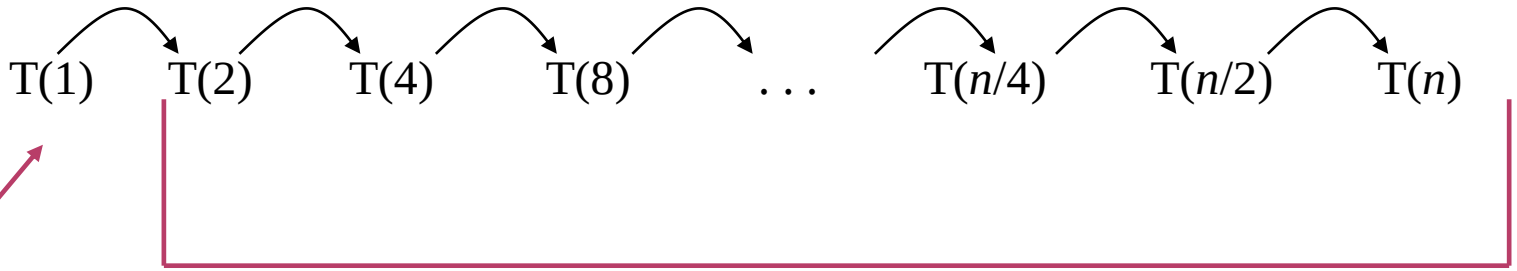
Redução subtrativa não necessariamente de 1 em 1



verificação de que T(1) e T(2) (bases) são verdade

hipótese de que T(n – 2) é verdade ($n \geq 3$)
+
prova de que a verdade de T(n – 2) implica que T(n) é verdade

Redução não necessariamente subtrativa

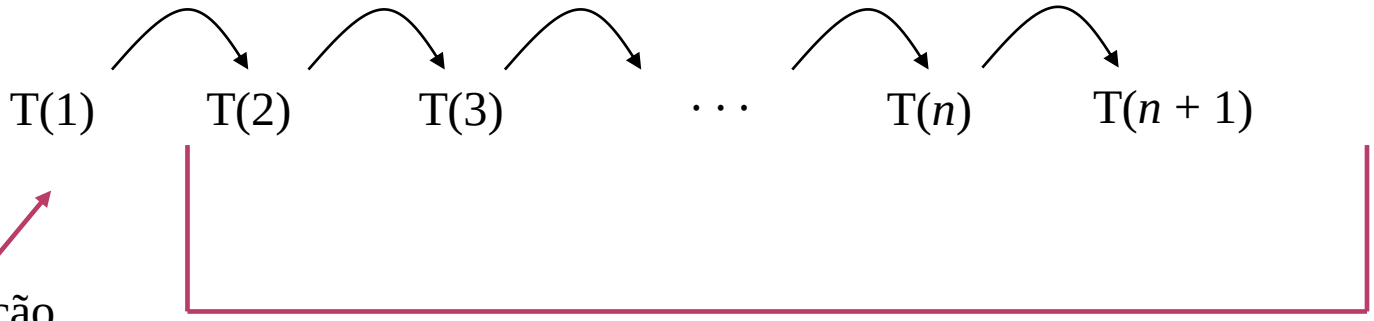


verificação de que T(1) é verdade

hipótese de que T(n/2) é verdade ($n \geq 2$)
+
prova de que a verdade de T(n/2) implica que T(n) é verdade

Obs.: nesta variante, a prova não valeria para todos os números naturais mas para 1, 2, 4, 8, ...

Término não necessariamente em T(n)



verificação de que T(1) (base) é verdade

hipótese de que T(n) é verdade ($n \geq 1$)
+
prova de que a verdade de T(n) implica que T(n + 1) é verdade

Partes de uma prova indutiva

Base ou bases

Hipótese de indução T(α)

Redução de T(β) para T(α)
(ou ampliação T(α) para T(β)
ou relação indutiva entre T(α) e T(β))

Prova de que T(α) implica
em T(β).

Não importa a ordem

As partes devem formar um todo consistente.

Redução de T(β) para T(α)

Hipótese de indução T(α)

Base ou bases

Prova de que T(α) implica em T(β).

Estratégia de prova

1 – Proponha uma redução de T(β) para T(α).

2 – Como a redução proposta foi para T(α), escreva a hipótese de indução T(α).

3 – Prove que o fato: se T(α) é verdadeiro então T(β) é verdadeiro.

4 – Observando a redução proposta, crie uma ou mais bases. Verifique se o teorema é verdadeiro para a(s) base(s).

Nova prova do teorema da soma dos *n* primeiros naturais

Use na prova a seguinte redução:

$$S(n) = S(n - 2) + 2n - 1.$$

Outras provas

Triângulo de Pascal

Ache uma expressão para a soma de uma linha do triângulo de Pascal e prove que sua expressão está correta.

```
1             = 1
1 1           = 2
1 2 1         = 4
1 3 3 1       = 8
1 4 6 4 1     = 16
```

Árvore binária cheia

Teorema
A quantidade de nós no nível *i* de uma árvore binária cheia não vazia é

$Q(i) = 2^i$, $i \geq 0$.

Obs.: estamos assumindo que a raiz da árvore binária cheia está no nível $i = 0$.

Uma inequação

Teorema

Se n é um número natural e $1 + x > 0$ então

$(1 + x)^n \geq 1 + nx$.

Quantidade de arestas e vértices de uma árvore

Teorema

A quantidade de arestas de uma árvore com $n \geq 1$ vértices é igual a

$$E(n) = n - 1.$$

Design de algoritmos por indução

Relação entre prova indutiva e algoritmo recursivo

i) Base

Caracterizada por $n = 1$.

Para $n = 1$ a soma é igual a 1.

Pela fórmula $S(1) = 1 \cdot (1 + 1) / 2 = 1$.

ii) Hipótese de indução

$S(n - 1) = (n - 1)\, n / 2$

iii) Relação entre S(*n*) e S(*n* – 1)

$S(n) = 1 + 2 + ... + n - 1 + n$

$S(n - 1) = 1 + 2 + ... + n - 1$

Logo,

$S(n) = S(n - 1) + n$.

iv) Prova de que S(*n* – 1) implica em S(*n*)

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

Usando a hipótese de indução,

$S(n) = (n - 1)\, n/2 + n$.

Logo,

$S(n) = n\,(n + 1) / 2$,

como queríamos provar.

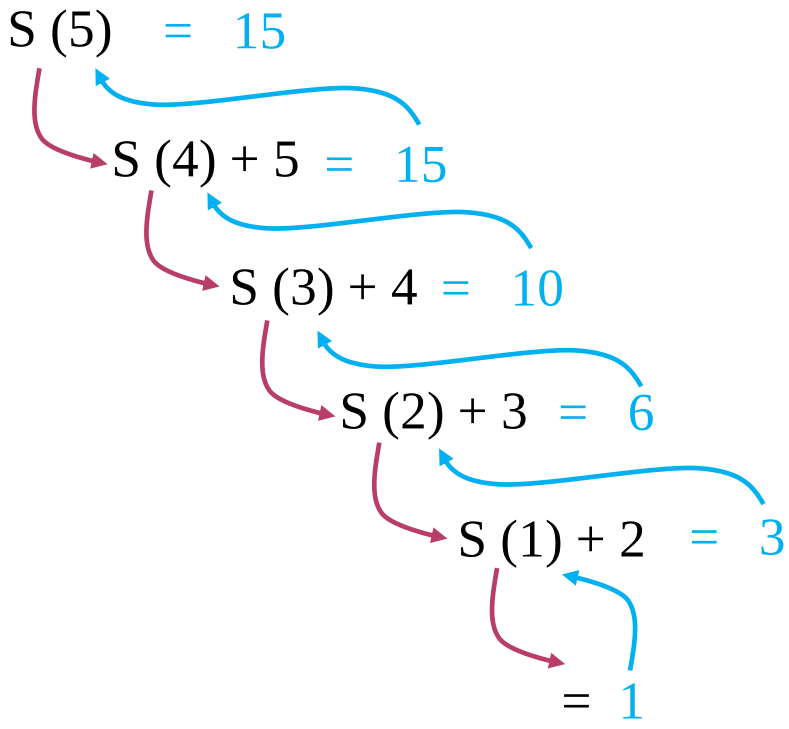
Algoritmo S (*n*)
Entrada: *n*, inteiro, $n \geq 1$.
Saída: a soma $1 + 2 + 3 + ... + n$.

```
{
    se ( n = 1 )
        retornar 1
    senão
        retornar S(n - 1) + n
}
```

O cálculo de S(5)

Algoritmo S (*n*)
Entrada: *n*, inteiro, $n \geq 1$.
Saída: a soma 1 + 2 + 3 + ... + *n*.

```
{
    se ( n = 1 )
        retornar 1
    senão
        retornar S(n - 1) + n
}
```



Relação entre prova indutiva e algoritmo recursivo

i) Bases

- Para $n = 1$ a soma é igual a 1 e pela fórmula $S(1) = 1.(1 + 1) / 2 = 1$.
- Para $n = 2$ a soma é igual a 3 e pela fórmula $S(2) = 2 \cdot (2 + 1) / 2 = 3$.

ii) Hipótese de indução

$S(n – 2) = (n - 2)(n – 1) / 2$

iii) Relação entre S(*n*) e S(*n* – 2)

$S(n) = 1 + 2 + ... + n - 2 + n - 1 + n$

$S(n - 2) = 1 + 2 + ... + n - 2$

Logo,

$S(n) = S(n - 2) + 2n - 1.$

iv) Prova de que S(*n* – 2) implica em S(*n*)

$S(n) = S(n - 2) + 2n – 1.$

Usando a hipótese de indução,

$S(n) = (n - 2)(n – 1) / 2 + 2n – 1.$

Logo,

$S(n) = n (n + 1) / 2,$

como queríamos provar.

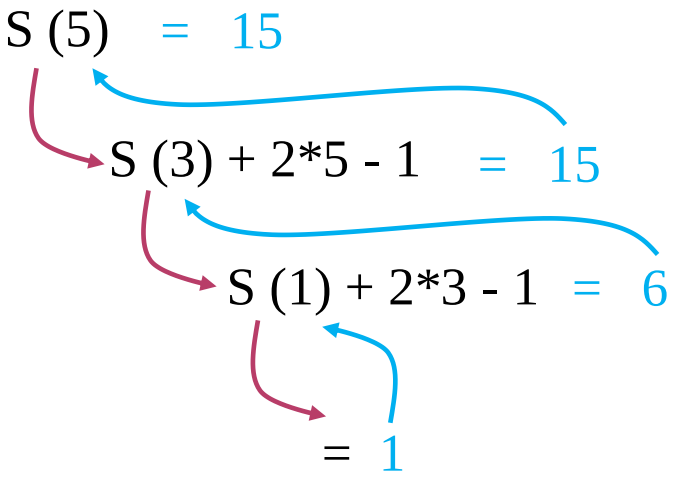
Algoritmo S (*n*)
Entrada: *n*, inteiro, $n \geq 1$.
Saída: a soma $1 + 2 + 3 + ... + n$.

```
{
    se ( n = 1 )
        retornar 1
    senão
        se ( n = 2 )
            retornar 3
        senão
            retornar S(n - 2) + 2*n - 1
}
```

O cálculo de S(5)

Algoritmo S (*n*)
Entrada: *n*, inteiro, $n \geq 1$.
Saída: a soma 1 + 2 + 3 + ... + *n*.

```
{
    se ( n = 1 )
        retornar 1
    senão
        se ( n = 2 )
            retornar 3
        senão
            retornar S(n - 2) + 2*n - 1
}
```



Uma técnica de design de algoritmos por indução

I – Proponha uma **interface** para o algoritmo.

II – Descreva o **significado** da interface proposta.

A sentença que descreve o significado deve especificar os parâmetros de entrada e saída.

III – Desenvolva uma **redução** de tamanho da solução do problema.

IV – Tendo como referência a dinâmica das reduções, proponha uma ou mais **bases**.

V – Escreva o **algoritmo**.

O algoritmo para o problema da soma dos *n* primeiros naturais obtido pela técnica descrita

I – Interface

S (*n*)

II – Significado

Retorna a soma dos *n* primeiros números naturais.

III – Redução

$S(n) = 1 + 2 + ... + n - 1 + n$

$S(n - 1) = 1 + 2 + ... + n - 1$

Logo, $S(n) = S(n - 1) + n$.

Comandos (tendo como referência o significado):

retornar $S(n - 1) + n$

IV – Base

A redução ocorre de forma subtrativa de 1 em 1, i.e., $S(n) \rightarrow S(n - 1) \rightarrow ... \rightarrow S(2) \rightarrow S(1)$.

Propomos a base para $n = 1$. Para $n = 1$ o resultado da soma é igual a 1.

Comandos (tendo como referência o significado):

retornar 1

O algoritmo para o problema da soma dos *n* primeiros naturais obtido pela técnica descrita

V – Algoritmo

Algoritmo S (*n*)
Entrada: *n*, inteiro, $n \geq 1$.
Saída: a soma 1 + 2 + 3 + ... + *n*.
{
    se (*n* = 1)
        retornar 1
    senão
        retornar S(*n* - 1) + *n*
}

Cálculo do maior elemento de um conjunto

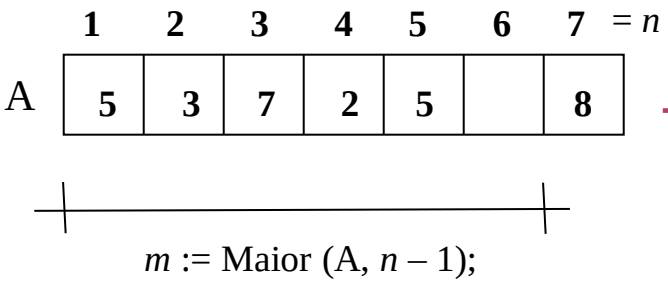
I – Interface

Maior (A, *n*)

II – Significado

Retorna o índice do maior elemento do vetor A que possui *n* elementos.

III – Redução



Comandos:

m := Maior (A, *n* – 1);

se (A[*n*] > A[*m*]) **retornar** *n*

senão *retornar m*

m recebe o índice do maior elemento entre os *n* - 1 primeiros

IV – Base

A redução ocorre de forma subtrativa de 1 em 1, assim propomos base para *n* = 1.

Um vetor com um único elemento tem o maior elemento na posição 1.

Comandos:

retornar 1

Cálculo do maior elemento de um conjunto

V – Algoritmo

Algoritmo Maior (A, *n*)
Entrada: A, vetor, de *n* elementos, $n \geq 1$.
Saída: o índice do maior elemento do vetor A.

```
{
    se ( n = 1 )
        retornar 1
     senão
    {
        m := Maior (A, n – 1);
        se ( A[n] > A[m] ) retornar n
        senão retornar m
     }
}
```

Complexidade de algoritmos recursivos

T(n)

Complexidade para resolver um problema de tamanho igual a n.

Algoritmo S (n)
Entrada: n, inteiro, $n \geq 1$.
Saída: a soma 1 + 2 + 3 + ... + n.
{
    se (n = 1)
        retornar 1
    senão
        retornar S(n - 1) + n
}

T(1) = 1

Complexidade para resolver um problema de tamanho 1. Seja 1 o valor do tempo para executar o comando "retornar 1".

T(n - 1) + 1

Complexidade para resolver um problema de tamanho igual a n – 1 mais o tempo para realizar uma soma. Seja 1 o valor do tempo para realizar uma soma.

Resolução de relações de recorrência

$T(1) = 1$

$T(n) = T(n-1) + 1, n \geq 2.$

Resolução de relações de recorrência

- Método iterativo progressivo;
- Método “dividir para conquistar”;
- Método da substituição (ver Cormen, não será abordado).

Obs.: existem outros métodos.

Exercícios sobre método iterativo progressivo

Resolver as seguintes relações de recorrência:

1) $T(1) = 1$
$T(n) = T(n-1) + 1, n \geq 2.$

2) $T(1) = 1$
$T(n) = T(n-1) + n, n \geq 2.$

Complexidade do algoritmo Maior

T(*n*)

Complexidade para resolver um problema de tamanho igual a *n*.

Algoritmo Maior (A, *n*)

Entrada: A, vetor, de *n* elementos, $n \geq 1$.

Saída: o índice do maior elemento do vetor A.

```
{
    se ( n = 1 )
        retornar 1
    senão
    {
        m := Maior (A, n – 1);
        se ( A[n] > A[m] )
            retornar n
        senão retornar m
    }
}
```

T(1) = 1

Complexidade para resolver um problema de tamanho 1. Seja 1 o valor do tempo (constante) para executar o comando "retornar 1".

T(*n* – 1) +

Tempo para executar uma Instância de tamanho igual a *n* – 1.

1

Seja 1 o valor do tempo (constante) para executar estes comandos.

Complexidade do algoritmo Maior

$T(1) = 1$

$T(n) = T(n - 1) + 1, n \geq 2.$

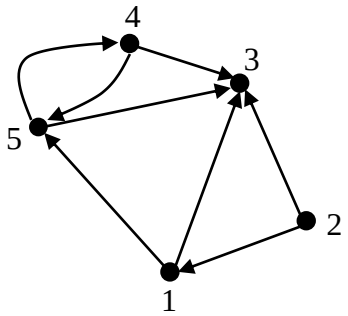
Resolvendo esta relação de recorrência, teremos $T(n) = n = O(n)$.

Outros problemas

Celebridade

- **Definição**: entre *n* pessoas uma **celebridade** é uma pessoa que é conhecida por todas as outras mas que não conhece ninguém.

Pessoa “A” conhece pessoa ‘B”



	1	2	3	4	5
1			1		1
2	1		1		
3					
4			1		1
5			1	1	

C

C [*v*, *w*] = 1: *v* CONHECE *w*

Celebridade: 3.

Solução 1

I – Interface

Celebridade (C, n)

II – Significado

Retorna o vértice celebridade v ($1 \leq v \leq n$) da matriz de conhecimentos C de tamanho n. Retorna $v = 0$, caso não haja um vértice celebridade.

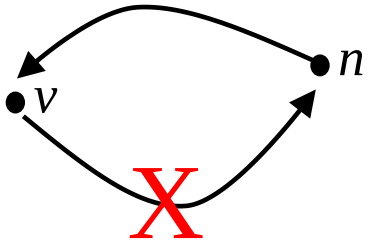
Solução 1

III – Redução

v := Celebridade (C, n - 1);

CASO 1: $1 \leq v \leq n - 1$, i.e., existe celebridade entre os vértices de 1 a $n - 1$.

CASO 1.1:



retornar v; // *v é celebridade.*

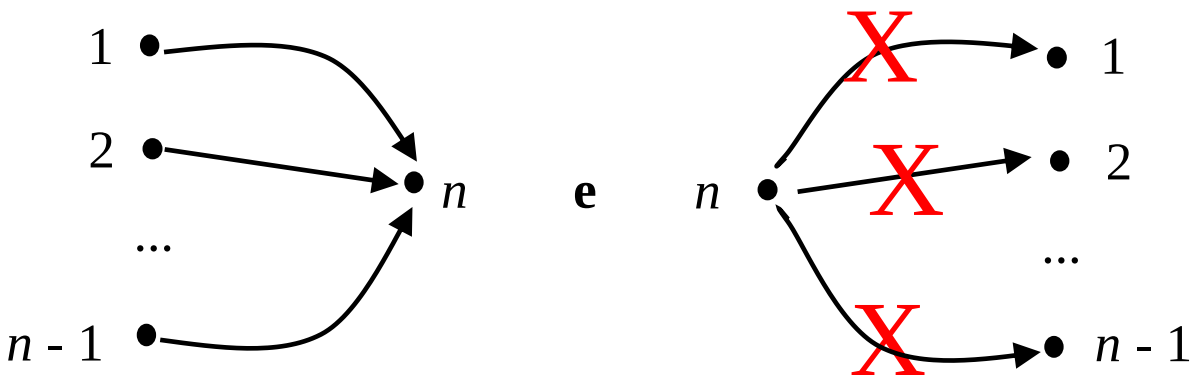
CASO 1.2: alguma das condições indicadas na figura acima falham.

retornar 0; // *não há celebridade*.

Solução 1

CASO 2: $v = 0$, i.e., não existe celebridade entre os vértices de 1 a $n - 1$.

CASO 2.1:



retornar *n*; // *n é celebridade.*

CASO 2.2: alguma das condições indicadas nas figuras acima falham.

retornar 0; // *não há celebridade*.

Solução 1

III – Base

Caracterizada por $n = 1$.

- 1

retornar 1; // 1 é (trivialmente) *celebridade*.

Complexidade da solução 1

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

$T(1) = 0$

Resolvendo: $T(n) = n\,(n-1) = O\,(n^2)$.

Esta solução tem a complexidade igual ao tamanho n^2 da entrada (uma matriz n x n).

Solução 2

Idéia para outro tipo de redução:

Descobrir as **não celebridades**.

Sejam A e B duas pessoas.

Se A conhece B então A não é celebridade. Caso contrário, B não é celebridade.

Em qualquer caso, a reduz-se a quantidade de não celebridades em 1.

Exercício

Escrever o algoritmo Celebridade tendo como motivação a ideia da solução 2.

Solução 2 - conclusões

• Tamanho da entrada do problema: n^2.

• Complexidade da solução 2: $O(n)$.

• Conseguimos resolver o problema em uma quantidade de tempo menor que o tamanho da entrada.

• Não reduzimos simplesmente o tamanho da entrada de n para $n - 1$. Escolhemos uma redução específica de n para $n - 1$ que nos promoveu o ganho de complexidade.

Divisão e conquista

- Em vez de reduções subtrativas, usaremos reduções que dividem o tamanho do problema.

- Duas fases:

  - Divisão: por exemplo, resolver dois problemas de tamanho $n/2$;

  - Conquista: combinar as soluções individuais para chegar na solução do problema de tamanho igual a n.

Exemplo de Design por “divisão e conquista”

Merge Sort (ordenação por intercalação)

Desenvolvendo o Merge Sort

I) Interface

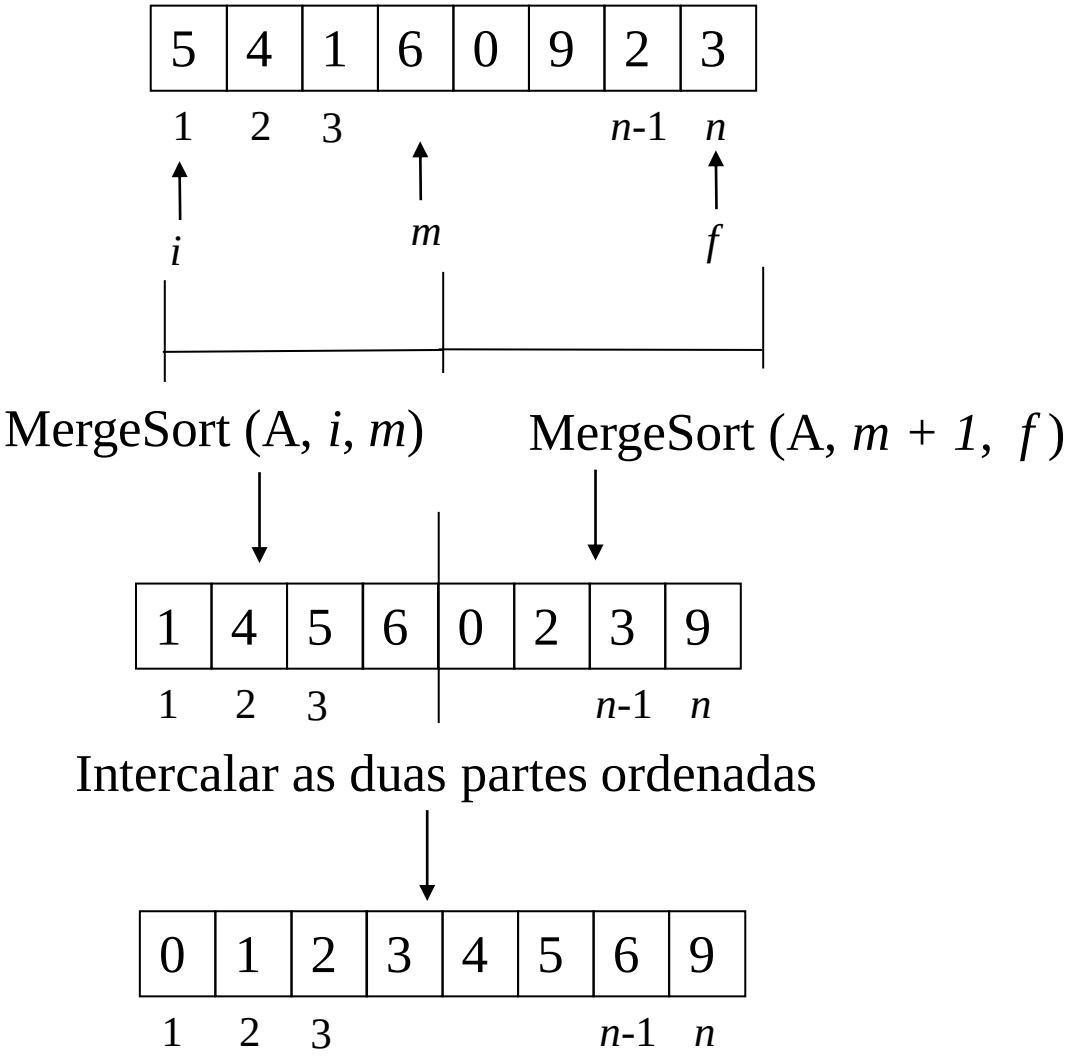
MergeSort (A, *i*, *f*)

II) Significado

Ordena “in-loco” o vetor A do índice *i* até o índice *f*.

Desenvolvendo o Merge Sort

III) Redução



Desenvolvendo o Merge Sort

Comandos:

m := $\lfloor$ (i + f) / 2$\rfloor$;

MergeSort (A, i, m);

MergeSort (A, m + 1, f)

Intercalar (A, i, m, f)

IV) Base

É caracterizada por i = f (veja reduções “dividir para conquistar”).

Comandos (para ordenar uma faixa de vetor com 1 elemento):

Nenhum comando é necessário.

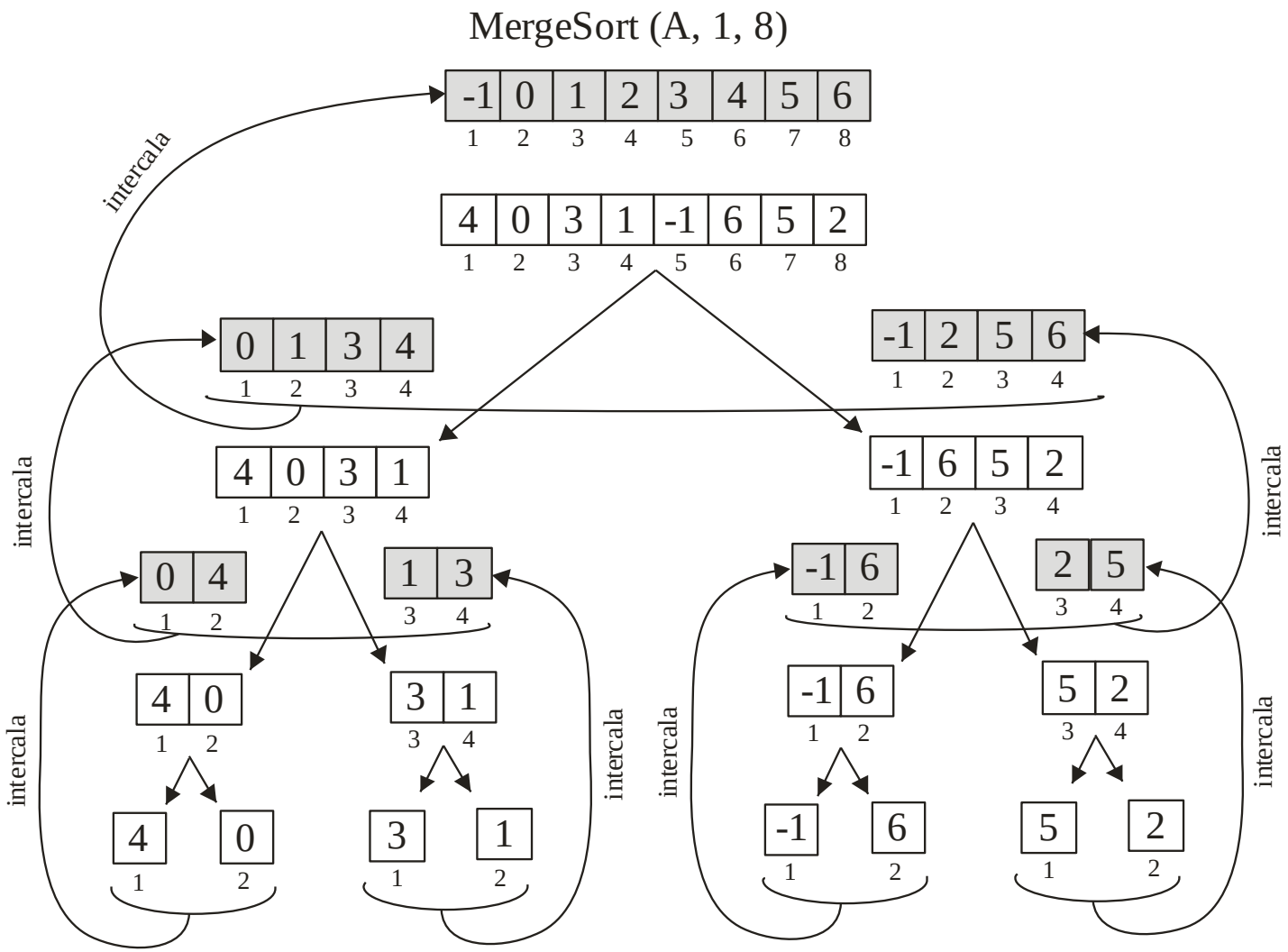
Desenvolvendo o Merge Sort

V) O Algoritmo

Algoritmo MergeSort (A, i, f)
Entrada: vetor ‘A’ e índices ‘i’ e ‘f’ do vetor (i ≤ f).
Saída: o vetor A, ordenado “in-loco” do índice ‘i’ até o índice ‘f’.

```
{
    se ( i < f )   // Ou i ≠ f .
    {
        m := ⌊ (i + f ) / 2⌋;
        MergeSort (A, i, m);
        MergeSort (A, m + 1, f );
        Intercalar (A, i, m, f )
    }
}
```

Ilustrando o funcionamento



Complexidade

Seja a quantidade de elementos $n = f - i + 1$

```
Algoritmo MergeSort (A, i, f )
Entrada: vetor 'A' e índices 'i' e 'f' do vetor (i ≤ f ).
Saída: o vetor A, ordenado "in-loco" do índice 'i' até o índice 'f'.
{
    se ( i < f )   // Ou i ≠ f .
    {
        m := ⌊ (i + f ) / 2 ⌋;
        MergeSort (A, i, m);
        MergeSort (A, m + 1, f );
        Intercalar (A, i, m, f )
    }
}
```

T(*n*)

T(*n*/2)

T(*n*/2)

n - 1

T(1) = 0

Algoritmos “Dividir para Conquistar” dão origem a relações de recorrência “Dividir para Conquistar”

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

$$T(1) = 0$$

Solução geral de relações de recorrência do tipo "dividir para conquistar" (Método da "Divisão e Conquista")

Teorema:
A solução da relação de recorrência $T(n) = aT(n/b) + cn^k$, onde a e b são constantes inteiras, $a \geq 1$, $b \geq 2$ e c e k constantes positivas, é

$$T(n) = \begin{cases} O(n^{\log_b a}) & se\ a > b^k \\ O(n^k \log n) & se\ a = b^k \\ O(n^k) & se\ a < b^k \end{cases}.$$

Exercícios: resolva as seguintes relações de recorrência

1) Merge Sort

$T(n) = 2T(n/2) + n$ (desprezamos o -1)

$T(1) = 0$

2) $T(n) = 4T(n/2) + n$

$T(1) = 0$