Tempo de

Execução

Tempo de Execução

Vários fatores afetam o tempo de execução de um programa:

1) Hardware
2) Linguagem
3) Tamanho da Entrada
4) Algoritmo

Queremos um critério de Tempo que seja independente de

1) Hardware
2) Linguagem

Assim podemos realmente comparar o tempo de execução de um algoritmo em relação ao tamanho da entrada.

Nosso Modelo de Computação

- Nosso computador teórico captura a essência de um computador real.
  - → Operações primitivas: operações que podem ser executadas rapidamente sobre um número pequeno (ex: 32, 64, 128 bits)

Algoritmo 1: Assembly

```
fib:
        push    rbp
        mov     rbp, rsp
        push    rbx
        sub     rsp, 24
        mov     DWORD PTR [rbp-20], edi
        cmp     DWORD PTR [rbp-20], 2
        jg      .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-20]
        sub     eax, 1
        mov     edi, eax
        call    fib
        mov     ebx, eax
        mov     eax, DWORD PTR [rbp-20]
        sub     eax, 2
        mov     edi, eax
        call    fib
        add     eax, ebx
.L3:
        mov     rbx, QWORD PTR [rbp-8]
        leave
        ret
```

Operações Primitivas

- operações aritméticas: soma, subtração, multiplicação, divisão, resto, piso, teto
- Operações relacionais: $<$, $\leqslant$, $>$, $\geqslant$, $=$, $\neq$
- Operações lógicas: and, or, not
- Atribuição de valores, acesso a posição de memória
- Operações de controle de fluxo: if, while, for

Tempo de Execução

O <u>tempo de execução</u> de um algoritmo é dado pela qntidade de operações primitivas (passos básicos) executada por ele para uma certa instância de entrada.

→ o tempo de execução cresce junto com a entrada.

→ o tempo de execução é uma função T sobre o <u>tamanho da entrada</u>.

- → T(n) denota o número de operações primitivas realizada pelo algoritmo qndo executada sobre uma entrada de tamanho n.

<u>Receita de Bolo</u>: tamanho da entrada

→ Vetor, lista, Conjunto: n é a quantidade de elementos.

→ número: n é a qntidade de bits na representação binária.

Exemplo

```
BuscaLinear(A[1..n], k)
    i = 1
    Enquanto i ≤ n e A[i] ≠ k
        i = i + 1
    Se i ≤ n
        Devolva i
    Devolva -1
```

Exemplo

```
BuscaLinear(A[1..n], k)
    i = 1
    Enquanto i ≤ n e A[i] ≠ k
        i = i + 1
    Se i ≤ n
        Devolva i
    Devolva -1
```

Se $k \in A$ e $A[p] = k$

Se $k \notin A$

Se o elemento existe: $T(n) =$

Se o elemento n̄ existe: $T(n) =$

Exemplo

	BuscaLinear$(A[1..n], k)$	Se $k \in A$ e $A[p] = k$	Se $k \notin A$
1	$i = 1$	1	1
2	Enquanto $i \leq n$ e $A[i] \neq k$	$5p$	$5(n+1)$
3	$\quad i = i + 1$	$2(p-1)$	$2n$
4	Se $i \leq n$	2	2
5	$\quad$ Devolva i	1	
6	Devolva -1	0	1

$\rightarrow 7n+2$

Se o elemento existe: $T(n) = 1 + 5p + 2(p-1) + 2 + 1 = 7p + 2$

$\rightarrow 9$

Se o elemento $\bar{n}$ existe: $T(n) = 1 + 5(n+1) + 2n + 2 + 1 = 7n + 9$

Exemplo

BuscaLinear$(A[1..n], k)$	Se $k \in A$ e $A[p] = k$	Se $k \notin A$
$i = 1$	1	1
Enquanto $i \leq n$ e $A[i] \neq k$	$5p$	$5(n+1)$
$\quad i = i + 1$	$2(p-1)$	$2n$
Se $i \leq n$	2	2
$\quad$ Devolva i	1	
Devolva -1	0	1

Se o elemento existe: $T(n) = 7p + 2$ $\quad \to 7n+2$ $\quad \to 9$

Se o elemento n̄ existe: $T(n) = 7n + 9$

$$9 \leq T(n) \leq 7n + 9$$

Exemplo

BuscaLinear$(A[1..n], k)$	Se $k \in A$ e $A[p] = k$	Se $k \notin A$
$i = 1$	1	1
Enquanto $i \leq n$ e $A[i] \neq k$	$5p$	$5(n+1)$
$\quad i = i + 1$	$2(p-1)$	$2n$
Se $i \leq n$	2	2
$\quad$ Devolva i	1	
Devolva -1	0	1

Se o elemento existe: $T(n) = 7p + 2$ $\;\to\; 7n+2$ $\;\to\; 9$

Se o elemento ñ existe: $T(n) = 7n + 9$

$$9 \leq T(n) \leq 7n + 9$$

Exemplo

```
BuscaBinaria(A[1..n], k)
esq = 1
dir = 1
Enquanto esq < dir faça
    meio = ⌊(esq + dir)/2⌋
    Se k > A[meio]
        esq = meio + 1
    Senão
        dir = meio
Se A[esq] == k
    Devolve esq
Devolve -1
```

Exemplo

Tempo de Execução

Busca Binaria(A[1..m], k)	
esq = 1	1
dir = 1	1
Enquanto esq < dir faça	$2n$, n é o número de vezes que esse teste executa
meio = ⌊(esq + dir)/2⌋	$4(n-1)$
Se k > A[meio]	$3(n-1)$
esq = meio + 1	$\leq 2(n-1)$
Senão	
dir = meio	
Se A[esq] == k	2
Devolve esq	1
Devolve -1	

$$T(n) = 1 + 1 + 2n + 4(n-1) + 3(n-1) + 2(n-1) + 2 + 1 = 11n - 4$$

Exemplo

Para n inteiro positivo e m e x reais arbitrarios, vale que

$$\left\lfloor \frac{\lfloor \frac{x}{m} \rfloor}{n} \right\rfloor = \left\lfloor \frac{x}{mn} \right\rfloor$$

$$\left\lceil \frac{\lceil \frac{x}{m} \rceil}{n} \right\rceil = \left\lceil \frac{x}{mn} \right\rceil$$

Em cada iteração do laço descartamos aproximadamente metade da entrada, restando um subproblema de tamanho $\lfloor \frac{n}{2} \rfloor$ ou $\lceil \frac{n}{2} \rceil$.

Fazemos isso até o subproblema ter tamanho 1, qndo o laço da linha 3 finaliza.

Exemplo

Para n inteiro positivo e m e x reais arbitrarios, vale que

$$\left\lfloor \frac{\left\lfloor \frac{x}{m} \right\rfloor}{n} \right\rfloor = \left\lfloor \frac{x}{mn} \right\rfloor$$

$$\left\lceil \frac{\left\lceil \frac{x}{m} \right\rceil}{n} \right\rceil = \left\lceil \frac{x}{mn} \right\rceil$$

No pior dos caso temos subproblemas de tamanho $\left\lceil \frac{n}{2} \right\rceil$. Então

$$1 = \left\lceil \frac{\left\lceil \frac{\left\lceil \frac{\left\lceil \frac{n}{2} \right\rceil}{2} \right\rceil}{2} \right\rceil \cdots}{2} \right\rceil = \left\lceil \frac{n}{2^{i}} \right\rceil$$

exec. do corpo do laço da linha 3

$r = i + 1$

$\left\lceil \frac{n}{2^i} \right\rceil = 1$ qndo $0 < \frac{n}{2^i} \leq 1$.

$$\frac{n}{2^i} \leq 1 \iff n \leq 2^i \iff \lg n \leq i$$

Exemplo

Para n inteiro positivo e m e x reais arbitrarios, vale que

$$\left\lfloor \frac{\left\lfloor \frac{x}{m} \right\rfloor}{n} \right\rfloor = \left\lfloor \frac{x}{mn} \right\rfloor$$

$$\left\lceil \frac{\left\lceil \frac{x}{m} \right\rceil}{n} \right\rceil = \left\lceil \frac{x}{mn} \right\rceil$$

No pior dos caso temos subproblemas de tamanho $\lceil \frac{n}{2} \rceil$. Então

$\left\lceil \frac{n}{2^i} \right\rceil = 1$ qndo $0 < \frac{n}{2^i} \leq 1$.

$$\frac{n}{2^i} \leq 1 \Leftrightarrow n \leq 2^i \Leftrightarrow \lg n \leq i$$

- i é inteiro
- $i = \lceil \lg n \rceil$ é qndo $\frac{n}{2^i} \leq 1$ pela primeira vez.

$$r = \lceil \lg n \rceil + 1$$

Exemplo

Para n inteiro positivo e m e x reais arbitrarios, vale que

$$\left\lfloor \frac{\left\lfloor \frac{x}{m} \right\rfloor}{n} \right\rfloor = \left\lfloor \frac{x}{mn} \right\rfloor$$

$$\left\lceil \frac{\left\lceil \frac{x}{m} \right\rceil}{n} \right\rceil = \left\lceil \frac{x}{mn} \right\rceil$$

No melhor dos caso temos subproblemas de tamanho $\left\lfloor \frac{n}{2} \right\rfloor$. Então

$\left\lfloor \frac{n}{2^i} \right\rfloor = 1$ qndo $1 \leq \frac{n}{2^i} < 2$.

$$\frac{n}{2^i} < 2 \iff n < 2^{i+1} \iff \lg n < i+1$$

$$\iff \lg n - 1 < i$$

$$\lfloor \lg n \rfloor + 1 = i$$

$$r = \lfloor \lg n \rfloor + 2$$

Exemplo

Busca Binaria(A[1..n], k)	Tempo de Execução
esq = 1	1
dir = 1	1
Enquanto esq < dir faça	$2r$, r é o número de vezes que esse teste executa
meio = $\lfloor (esq + dir)/2 \rfloor$	$4(r-1)$
Se k > A[meio]	$3(r-1)$
esq = meio + 1	$\leq 2(r-1)$
Senão	
dir = meio	
Se A[esq] == k	2
Devolve esq	1
Devolve -1	

$$T(n) = 11r - 4$$

$$\lceil \lg n \rceil + 1 \leq r \leq \lfloor \lg n \rfloor + 2$$

$$11\lceil \lg n \rceil + 7 \leq T(n) \leq 11\lfloor \lg n \rfloor + 18$$

Análise por casos

O <u>tempo de melhor caso</u> de um algoritmo é o menor tempo de execução do algoritmo dentre todos os tempos de execução de todas as instâncias de um dado tamanho n.

Busca linear: $9 \leq T(n) \leq 7n + 9$

Busca Binária: $11\lceil \lg n \rceil + 7 \leq T(n) \leq 11\lfloor \lg n \rfloor + 18$

Análise por casos

O tempo de pior caso de um algoritmo é o maior tempo de execução do algoritmo dentre todos os tempos de execução de todas as instâncias de um dado tamanho n.

Busca linear: $9 \leq T(n) \leq 7n + 9$

Busca Binária: $11\lceil \lg n \rceil + 7 \leq T(n) \leq 11\lfloor \lg n \rfloor + 18$

Análise por casos

Se $T(n)$ é o tempo de execução para uma entrada de tamanho n, entao

$$\text{Tempo no melhor caso} \leq T(n) \leq \text{Tempo no pior caso}$$

Esses tempos nos dão garantias:

$$9 \leq \text{Tempo BuscaLinear} \leq 7n + 9$$

$$11\lceil \lg n \rceil + 7 \leq \text{Tempo BuscaBinaria} \leq 11\lfloor \lg n \rfloor + 18$$

Usando notação Assintótica

Exemplo

BuscaLinear$(A[1..n], k)$	Se $k \in A$ e $A[p] = k$	Se $k \notin A$
$i = 1$	1	1
Enquanto $i \leq n$ e $A[i] \neq k$	$5p$	$5(n+1)$
$\quad i = i + 1$	$2(p-1)$	$2n$
Se $i \leq n$	2	2
$\quad$ Devolva i	1	
Devolva -1	0	1

Se o elemento existe: $T(n) =$

Se o elemento ñ existe: $T(n) =$

Exemplo

	BuscaLinear$(A[1..m], k)$	Se $k \in A$ e $A[p] = k$		Se $k \notin A$	
1	$i = 1$	1	$\Theta(1)$	1	$\Theta(1)$
2	Enquanto $i \leq m$ e $A[i] \neq k$	$5p$	$\Theta(p)$	$5(m+1)$	$\Theta(n)$
3	$\quad i = i + 1$	$2(p-1)$	$\Theta(p)$	$2m$	$\Theta(n)$
4	Se $i \leq n$	2	$\Theta(1)$	2	$\Theta(1)$
5	$\quad$ Devolva i	1	$\Theta(1)$		
6	Devolva -1	0		1	$\Theta(1)$

Se o elemento existe: $T(n) = \Theta(1) + \Theta(p) + \Theta(p) + \Theta(1) + \Theta(1)$

$$= \Theta(p) \rightarrow \left.\begin{array}{l} \Omega(1) \\ O(n) \end{array}\right\} \text{em função da entrada}$$

Se o elemento n̄ existe: $T(m) = 7m + 9$

Exemplo

BuscaLinear$(A[1..n], k)$	Se $k \in A$ e $A[p] = k$		Se $k \notin A$	
$i = 1$	1	$\Theta(1)$	1	$\Theta(1)$
Enquanto $i \leq n$ e $A[i] \neq k$	$5p$	$\Theta(p)$	$5(n+1)$	$\Theta(n)$
$\quad i = i + 1$	$2(p-1)$	$\Theta(p)$	$2n$	$\Theta(n)$
Se $i \leq n$	2	$\Theta(1)$	2	$\Theta(1)$
$\quad$ Devolva i	1	$\Theta(1)$		
Devolva -1	0		1	$\Theta(1)$

Se o elemento existe: $T(n) = \Theta(p)$ $\to \Omega(1)$, $\to O(n)$

Se o elemento $\bar{n}$ existe: $T(n) = \Theta(1) + \Theta(n) + \Theta(n) + \Theta(1) + \Theta(1)$

$= \Theta(n)$

Exemplo

```
BuscaLinear(A[1..n], k)
    i = 1
    Enquanto i ≤ n e A[i] ≠ k
        i = i + 1
    Se i ≤ n
        Devolva i
    Devolva -1
```

Conclusão

$T(n) = O(n)$

$T(n) = \Omega(1)$

Se o elemento existe: $T(n) = \Theta(p)$ $\to \Omega(1)$
$\to O(n)$

Se o elemento ñ existe: $T(n) = \Theta(n)$

Exemplo

BuscaLinear(A[1..n], k)

```
i = 1
Enquanto i ≤ n e A[i] ≠ k
    i = i + 1
Se i ≤ n
    Devolva i
Devolva -1
```

Conclusão

$T(n) = O(n)$

$\cancel{T(n) = \Omega(1)}$

Na maioria dos caso Vamos nos preocupar apenas com a análise de pior caso.

Se o elemento existe: $T(n) = \Theta(p)$ → $\cancel{\Omega(1)}$, → $O(n)$

Se o elemento ñ existe: $T(n) = \Theta(n)$

Análise - Formalização

FAZER Assim nas Listas

```
BuscaLinear(A[1..n], k)
1   i = 1
2   Enquanto i ≤ n e A[i] ≠ k
3       i = i + 1
4   Se i ≤ n
5       Devolva i
6   Devolva -1
```

- As instruções das linhas 1 e 4-6 executam apenas um vez e todas levam tempo constante. logo o custo para executá-las é $\Theta(1)$
- O corpo do laço da linha 2 e o teste do laço levam tempo $\Theta(1)$ para executar apenas uma vez. O laço é executado $O(n)$ vezes. Assim, o custo total das linhas 2-3 é $O(n)$

Análise - Formalização

- As instruções das linhas 1 e 4-6 executam apenas um vez e todas levam tempo constante. Logo o custo para executá-las é $\Theta(1)$
- O corpo do laço da linha 2 e o teste do laço levam tempo $\Theta(1)$ para executar apenas uma vez. O laço é executado $O(n)$ vezes. Assim, o custo total das linhas 2-3 é $O(n)$

- Assim, o tempo de execução do algoritmo é $\Theta(1) + O(n) = O(n)$.

Exemplo

$$\lceil \lg n \rceil + 1 \leq r \leq \lfloor \lg n \rfloor + 2$$

	Tempo de Execução
BuscaBinaria(A[1..n], k)	
esq = 1	1
dir = 1	1
Enquanto esq < dir faça	$2r$, r é o número de vezes que esse teste executa
meio = $\lfloor (esq + dir)/2 \rfloor$	$4(r-1)$
Se k > A[meio]	$3(r-1)$
esq = meio + 1	$\leq 2(r-1)$
Senão	
dir = meio	
Se A[esq] == k	2
Devolve esq	1
Devolve -1	

Exemplo

$$\boxed{\lceil \lg n \rceil + 1 \leq r \leq \lfloor \lg n \rfloor + 2}$$

Tempo de Execução

Busca Binaria(A[1..n], k)		
esq = 1	1	$\Theta(1)$
dir = 1	1	$\Theta(1)$
Enquanto esq < dir faça	$2r$, r é o número de vezes que esse teste executa	$\Theta(\lg n)$
meio = $\lfloor (esq + dir)/2 \rfloor$	$4(r-1)$	$\Theta(1) \cdot \Theta(\lg n) = \Theta(\lg n)$
Se k > A[meio]	$3(r-1)$	
esq = meio + 1	$\leq 2(r-1)$	
Senão		
dir = meio		
Se A[esq] == k	2	$\Theta(1)$
Devolve esq	1	
Devolve −1		

$$T(n) = \Theta(1) + \Theta(1) + \Theta(\lg n) + \Theta(\lg n) + \Theta(1) = \Theta(\lg n).$$

Exemplo - Formalização

```
  Busca Binaria(A[1..n], k)
1 esq = 1
2 dir = 1
3 Enquanto esq < dir faça
4     meio = ⌊(esq + dir)/2⌋
5     Se k > A[meio]
6         esq = meio + 1
7     Senão
8         dir = meio
9 Se A[esq] == k
10    Devolve esq
11 Devolve -1
```

- As linhas 1, 2 e 9-11 levam tempo constante para executar e executam apenas uma vez. Logo o tempo total de execução dessas linhas é $\Theta(1)$

- O custo para executa o teste da linha 3 e o corpo do laço da linha 3 uma única vez é contante. Essas linha são executadas $\Theta(\lg n)$. Assim, o custo total com esse trecho é $\Theta(\lg n)$.

Exemplo - Formalização

- As linhas 1, 2 e 9-11 levam tempo constante para executar e executam apenas uma vez. Logo o tempo total de execução dessas linhas é $\Theta(1)$

- O custo para executa o teste da linha 3 e o corpo do laço da linha 3 uma única vez é contante. Essas linha são executadas $\Theta(\lg n)$. Assim, o custo total com esse trecho é $\Theta(\lg n)$.

- Assim, o tempo de execução do algoritmo é $\Theta(1) + \Theta(\lg n) = \Theta(\lg n)$

Análise por casos

O <u>tempo de caso médio</u> de um algoritmo é o tempo esperado da execução do algoritmo para uma entrada de tamanho n

- → Consideramos algo sobre a distribuição das entradas e fazemos uma análise probabilística.
- → Pode ser tão ruim qnto o pior caso.

Exemplo

```
BuscaLinear(A[1..n], k)
    i = 1
    Enquanto i ≤ n e A[i] ≠ k
        i = i + 1
    Se i ≤ n
        Devolva i
    Devolva -1
```

Vamos supor que $k \in A$ e que cada entrada de A tem a mesma probabilidade de conter A

distribuição uniforme!

Exemplo

```
BuscaLinear(A[1..n], k)
    i = 1                            } Θ(1)
    Enquanto i ≤ n e A[i] ≠ k        }
        i = i + 1                    } Θ(1)·X
    Se i ≤ n                         }
        Devolva i                    } Θ(1)
    Devolva -1                       }
```

$\Theta(1) \cdot X$ — # de iterações

Variável indicadora

$$X_i = \begin{cases} 1 & \text{se } A[i] = k \\ 0 & \text{caso contrário} \end{cases}$$

$$X = \sum_{i=1}^{n} X_i \cdot i$$

Exemplo

Então o número esperado de iterações é $\mathbb{E}[X]$

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i \cdot i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i \cdot i] = \sum_{i=1}^{n} \mathbb{E}[X_i] \cdot i$$

$$= \sum_{i=1}^{n} \mathbb{P}(A[i] == k) \cdot i = \sum_{i=1}^{n} \frac{1}{n} \cdot i = \frac{1}{n} \sum_{i=1}^{n} i$$

$$= \frac{1}{n} \cdot \frac{n(n+1)}{2} = \frac{n+1}{2} \in \Theta(n)$$

Exemplo

BuscaLinear$(A[1..n], k)$

$i = 1$ $\quad \Theta(1)$

Enquanto $i \leq n$ e $A[i] \neq k$

$\quad i = i + 1$

\# de iterações

$\Theta(1) \cdot x = \Theta(1) \cdot \Theta(n) = \Theta(n)$

Se $i \leq n$

$\quad$ Devolva i

Devolva -1

$\Theta(1)$

Custo médio: $\Theta(1) + \Theta(n) + \Theta(1)$

$= \Theta(n)$

Exemplo - Formalização

```
BuscaLinear(A[1..n], k)
1   i = 1
2   Enquanto i ≤ n e A[i] ≠ k
3       i = i + 1
4   Se i ≤ n
5       Devolva i
6   Devolva -1
```

- Vamos assumir que $k \in A$ e que cada posição de A possui a mesma probabilidade de conter k.

- Os custos de execução das linhas 1, 4-6 são contantes e tais linhas executam apenas uma vez, portanto o tempo total empregado em tal trecho é $\Theta(1)$

Exemplo - Formalização

```
BuscaLinear(A[1..n], k)
1   i = 1
2   Enquanto i ≤ n e A[i] ≠ k
3       i = i + 1
4   se i ≤ n
5       Devolva i
6   Devolva -1
```

- O tempo empregado para executa o código das linhas 2-3 uma única vez é $\Theta(1)$.

  Vamos agora computar o número esperado de execuções desse trecho.

- Seja X_i uma variável indicadora definida como

$$X_i = \begin{cases} 1 & \text{se } A[i] = k \\ 0 & \text{se } A[i] \neq k \end{cases}$$

Exemplo - Formalização

```
BuscaLinear(A[1..n], k)
1   i = 1
2   Enquanto i ≤ n e A[i] ≠ k
3       i = i + 1
4   Se i ≤ n
5       Devolva i
6   Devolva -1
```

- Seja $X = \sum_{i=1}^{n} X_i \cdot i$ e note que X é uma variável aleatória que conta o número de iterações do laço da linha 2

- O valor esperado de X é

Exemplo - Formalização

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i \cdot i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i \cdot i] = \sum_{i=1}^{n} \mathbb{E}[X_i] \cdot i$$

$$= \sum_{i=1}^{n} \mathbb{P}(A[i] == k) \cdot i = \sum_{i=1}^{n} \frac{1}{n} \cdot i = \frac{1}{n} \sum_{i=1}^{n} i$$

$$= \frac{1}{n} \cdot \frac{n(n+1)}{2} = \frac{n+1}{2} \in \Theta(n)$$

$(*)$

Exemplo - Formalização

```
BuscaLinear(A[1..n], k)
1  i = 1
2  Enquanto i ≤ n e A[i] ≠ k
3      i = i + 1
4  se i ≤ n
5      Devolva i
6  Devolva -1
```

- Seja $X = \sum_{i=1}^{n} X_i \cdot i$ e note que X é uma variável aleatória que conta o número de iterações do laço da linha 2

- O valor esperado de X é

$$(*)$$

- Como $X \in \Theta(n)$, temos que o custo com o trecho

Exemplo - Formalização

```
BuscaLinear(A[1..n], k)
1    i = 1
2    Enquanto i ≤ n e A[i] ≠ k
3        i = i + 1
4    Se i ≤ n
5        Devolva i
6    Devolva -1
```

das linhas 2-3 é $O(n)$

- Assim, o custo médio do Algoritmo é $\Theta(1) + \Theta(n)$ $= \Theta(n)$

Análise Amortizada

- Na análise amortizada, nós calculamos o tempo médio necessário para executar uma sequência de n operações sobre uma estrutura de dados.

  → Com ela podemos mostrar que o custo de uma operação é baixo, se nós tirarmos a média sobre uma sequência de chamadas, mesmo que algumas chamadas sejam custosas

Análise Amortizada: Vetor Dinâmico

```
vector-insert(T, x)
    se T.size == 0
        T.table = nova tabela 1 slot
        T.size = 1
        T.num = 0
    se T.num == T.size
        newtable = nova tabela 2 x T.size slots
        Para i=0 até T.num - 1
                newtable[i] = T.table[i]
        free T.table
        T.table = newtable
        T.size = 2 x T.size
    T.table[T.num] = x
    T.num = T.num + 1
```

Análise Amortizada: Vetor Dinâmico

vector-insert(T, x)

Se há espaço p/ x

se T.size == 0 | $\Theta(1)$

T.table = nova tabela 1 slot

T.size = 1

T.num = 0

se T.num == T.size | $\Theta(1)$

newtable = nova tabela 2 × T.size slots

Para i=0 até T.num - 1

newtable[i] = T.table[i]

free T.table

T.table = newtable

T.size = 2 x T.size

$T(n) = \Theta(1)$

T.table[T.num] = x

T.num = T.num + 1 | $\Theta(1)$

Análise Amortizada: Vetor Dinâmico

```
vector-insert(T, x)
    se T.size == 0                                  | Θ(1)
        T.table = nova tabela 1 slot
        T.size = 1
        T.num = 0
    se T.num == T.size
        newtable = nova tabela 2×T.size slots
        Para i=0 até T.num-1
                newtable[i] = T.table[i]            Θ(n)
        free T.table
        T.table = newtable
        T.size = 2 x T.size
T.table[T.num] = x                                  | Θ(1)
T.num = T.num + 1
```

Se <u>não</u> há espaço p/ x

$n := T.num$

$T(n) = \Theta(n)$

Análise Amortizada: Vetor Dinâmico

Análise de pior caso: $O(n)$

```
vector-insert(T, x)
    se T.size == 0
        T.table = nova tabela 1 slot
        T.size = 1
        T.num = 0
    se T.num == T.size
        newtable = nova tabela 2×T.size slots
        Para i=0 até T.num-1
            newtable[i] = T.table[i]
        free T.table
        T.table = newtable
        T.size = 2 x T.size
    T.table[T.num] = x
    T.num = T.num + 1
```

Análise Amortizada: Vetor Dinâmico

```
vector-insert(T, x)
    se T.size == 0
        T.table = nova tabela 1 slot
        T.size = 1
        T.num = 0
    se T.num == T.size
        newtable = nova tabela 2 x T.size slots
        Para i=0 até T.num - 1
                newtable[i] = T.table[i]
        free T.table
        T.table = newtable
        T.size = 2 x T.size
    T.table[T.num] = x
    T.num = T.num + 1
```

Análise Amortizada

- Em uma sequência de n chamadas vector-insert, seja c_i o custo da i-ésima chamada
- Custo médio de vector-insert é, portanto,

$$\frac{1}{n} \sum_{i=1}^{n} c_i$$

Análise Amortizada: Vetor Dinâmico

```
vector-insert(T, x)
    se T.size == 0
        T.table = nova tabela 1 slot
        T.size = 1
        T.num = 0
    se T.num == T.size
        newtable = nova tabela 2×T.size slots
        Para i=0 até T.num-1
            newtable[i] = T.table[i]
        free T.table
        T.table = newtable
        T.size = 2×T.size
    T.table[T.num] = x
    T.num = T.num+1
```

Análise Amortizada

- Note que

$$c_i = \begin{cases} i & \text{se } i-1 = 2^k \\ 1 & \text{caso contrário} \end{cases}$$

- Assim

$$\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j$$

$$< n + 2n$$

$$= 3n$$

Análise Amortizada: Vetor Dinâmico

```
vector-insert(T, x)
  se T.size == 0
      T.table = nova tabela 1 slot
      T.size = 1
      T.num = 0
  se T.num == T.size
      newtable = nova tabela 2×T.size slots
      Para i=0 até T.num-1
          newtable[i] = T.table[i]
      free T.table
      T.table = newtable
      T.size = 2 x T.size
  T.table[T.num] = x
  T.num = T.num + 1
```

Análise Amortizada

- Portanto, o custo médio é

$$\frac{1}{n}\sum_{i=1}^{n} c_i \leq \frac{1}{n}\cdot 3n = 3$$

- Concluímos que vector-insert(T, x) é $O(1)$ amortizado!

Algoritmos e Tempo de Execução

Um algoritmo A é polinomial se existe uma constante $k \geq 1$ tal que $T_A(n) \in O(n^k)$, onde $T_A(n)$ é a função do tempo de execução do algoritmo A no pior caso.

→ Se $T_A(n) \in \Theta(n)$, então dizemos que A é um algoritmo Linear.

→ Se $T_A(n) \in \Theta(n^2)$, então dizemos que A é um algoritmo quadrático.

Um algoritmo A é eficiente se é polinomial.