

Análise da Complexidade: Resolução de Exercícios:

B. Exercícios de Testes:

1. A função começa por inicializar o array $nr[]$ com 1;

É comparada a posição i com j (imediatamente anterior) e se os valores de $a[i].s \leq a[j].s$, então $nr[i] = 0$.
(ints)

Então, esta função coloca 0 nas posições de $nr[]$, onde o valor mas é mínimo, o que garante que, no final, se tenha 1 na posição com o valor mínimo.

Para a análise assintótica, vamos considerar como operação relevante o número de comparações $a[i].s \leq a[j].s$ feitas. Não precisamos de distinguir casos, pois em qualquer situação, terá de se fazer os três ciclos. Então:

$$\begin{aligned} T(N) &= \sum_{i=0}^{m-1} 1 + \sum_{i=0}^{m-1} \sum_{j=0}^{i-1} 1 = m + \sum_{i=0}^{m-1} i = \\ &= m + \frac{(m-1)(m)}{2} = (2m + m^2 - m)/2 = (m + m^2)/2 = \Theta(m^2) \end{aligned}$$

Fórmula utilizada: $\sum_{i=a}^b i = \frac{(b-a+1)(b-a+2)}{2}$

3. A função `bsort` chama a função `bubble`, enquanto esta retorna valores positivos (ou seja, enquanto forem efectuadas trocas de índices no array passado). Isto garante que o array, no final da execução, está ordenado crescentemente. Sabemos que o tempo da `bsort` será o tempo da `bubble`. Contudo, o nº de swaps vai depender da comparação do if. Então, a análise relevante é no número de trocas feito.

Melhor caso: é feito um número mínimo de trocas. O array está ordenado crescentemente, logo, só efetuadas comparações, mas não há swaps.

$$T_{\text{bsort}}(N) = T_{\text{bubble}}(N) = \sum_{i=0}^{m-1} 1 = m-1+1 = m = \Omega(n)$$

Pior caso: é feito o número máximo de iterações (trocas). O array está ordenado decrescentemente. Neste caso, é relevante o nº de trocas:

$$T_{\text{bsort}}(N) = T_{\text{bubble}}(N) = \sum_{i=0}^{m-1} N_{\text{swaps}} = N(N-1) = \Omega(N^2)$$

Fórmula: $\sum_{i=0}^n i = \frac{n(n+1)}{2}$

Então: $T_{\text{bsort}} = \Omega(n^2); \Theta(n^2)$

5. O tamanho do input é relevante (mº bits).

O melhor e pior caso dependem do mº de bits de white.

O resto da divisão inteira pode ser calculado subtraindo o divisor ao dividendo enquanto o resultado for inferior ao divisor.

Melhor caso: quanto menor for o x

$$T(N) = 2^{N-1}$$

$$100 = 4$$

Pior caso: quanto maior for o x

$$T(N) = 2^N - 1$$

$$111 = 7$$

2.

$$T(N) = \begin{cases} 1 & , m = 0 \\ 1 + T\left(\frac{N}{2}\right) & , m > 0 \end{cases}$$

Chamada	Nível k	Árvore	Custo
Reunião			
$T(N)$	0	$T(N)$	N
$T\left(\frac{N}{2}\right)$	1	$T\left(\frac{N}{2}\right)$	$2N/2 = N$
$T\left(\frac{N}{4}\right)$	2	$T\left(\frac{N}{4}\right)$ $T\left(\frac{N}{4}\right)$ $T\left(\frac{N}{4}\right)$ $T\left(\frac{N}{4}\right)$	N
$T\left(\frac{N}{2^k}\right)$	k

$$T(1) = \log_2 N$$

• No último nível da árvore: $T\left(\frac{N}{2^k}\right) = T(1) \Leftrightarrow \frac{N}{2^k} = 1 \Leftrightarrow$

$$\Leftrightarrow N = 2^k \Leftrightarrow k = \log_2 N$$

• a árvore tem k níveis e cada nível tem custo N, logo:

$$\begin{aligned} T(N) &= \sum_{k=0}^{\log_2 N} N = N \cdot \sum_{k=0}^{\log_2 N} 1 = N \cdot \left(1 + \sum_{k=1}^{\log_2 N} 1 \right) = \\ &= N \left(1 + \log_2 N \right) = N + N \cdot \log_2 N = \Theta(N \log_2 N) \end{aligned}$$

$$7. T(N) = \begin{cases} 1 & , N = 1 \\ 1 + 2 \cdot T(N/2), & N > 1 \end{cases}$$

Não é necessário utilizar first e last, basta ver que:

- i) o input é o tamanho do array $N = \text{last} - \text{first} + 1$
- ii) se $\text{first} == \text{last} \Rightarrow N = 1$
- iii) se $\text{first} < \text{last} \rightarrow$ recursividade

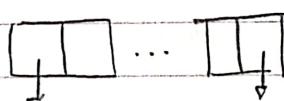
Chamada	Nível (K)	Ação	custo
<u>Recursiva</u>			
$T(N)$	0	$T(N)$	$1 (2^0)$
$T(N/2)$	1	$T(N/2)$	$2 (2^1)$
$T(N/4)$	2	$T(N/4)$	$4 (2^2)$
$T(N/2^K)$	K	...	2^K
$T(1)$	$\log_2 N$

- No último nível da árvore: $T(1) = T\left(\frac{N}{2^K}\right) \Leftrightarrow 1 = \frac{N}{2^K} \Leftrightarrow N = 2^K \Leftrightarrow K = \log_2 N$

- A árvore tem K níveis (de 0 a $\log_2 N$) e cada um tem custo 2^K ,

$$T(N) = \sum_{k=0}^{\log_2 N} 2^K = 2^{\log_2 N + 1} - 1 = 2^{\log_2 N} \cdot 2^1 - 1 = 2 \cdot N - 1 = \Theta(N)$$

- 10. • Armazenados por ordem decrescente da sua significância



mais significativo menos significativo

(1)

- Importa o tamanho (nº bits do número a armazenar)

Melhor caso: é apenas necessário fazer um bit flip, ou seja, os $N-1$ bits ser iguais a 1 e o último é 0 (1...110) $[m = 2^{N-1}]$

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

Pior caso: serão feitos $N-1$ bit flips, já que estes bits estar todos a 1 e o mais significativo é 0. $T(N) = N-1 = O(N)$

(0111...1)

$$[m = 2^N - 1]$$

(b) O método em questão é o agregado, entao:

input	output	custo real (c_i)	
0000	000 1	1	em
0001	001 0	$1+1 = 2$	N
0010	001 1	1	invocações
0011	0100	$1+1+1 = 3$	
0100	0101	1	
...	...	$1+1+1+1 = 4$	

- Em N invocações (linhas), há a parcela 1; $N \cdot 1$
- Melhor teoricamente a parcela $+1$; $N/2$
- Um quarto tem mais uma parcela $+1$; $N/4$

$$\begin{aligned}
 c_i &= \sum_{i=0}^{\lfloor \log_2 N \rfloor} \frac{N}{2^i} = N \sum = N \sum 2^{-i} = \\
 &= N \cdot \left[2^{-\lfloor \log_2 N \rfloor + 1} - 1 \right] = N \cdot 2^{-\lfloor \log_2 N \rfloor - 1} - 1 = \\
 &= N \left(2^{N-1} - 1 \right) = N \left(\frac{2}{N} - 1 \right) = 2N - 1
 \end{aligned}$$

$$C_i = \frac{2N-1}{N} = \frac{2-\frac{1}{N}}{\underset{\approx 0}{\underbrace{N}}} = 2 \rightarrow \Theta(1)$$

12.

$$T(N) = \begin{cases} 1 & n \leq 1 \rightsquigarrow \text{já feita anterior} \\ 1 + 2T(N/2) & \rightsquigarrow \Theta(m) \end{cases}$$

13. • tamanho input $\equiv N \equiv$ nº elementos da árvore
 $\equiv L \equiv$ comprimento do caminho

Melhor caso: é feito um nº mínimo de chamadas recursivas, ou seja, quando o elemento alcançável é a raiz

$$T(N)_l = 2 \text{ comparações} = \begin{cases} 1 & N=0 \\ 1 + T(0, L-1) & P=0 \\ 1 + T(0, L-1) & \Rightarrow 2 = \Theta(1) \end{cases}$$

Pior caso: é feito um nº máximo de chamadas recursivas, ou seja, o elemento alcançável é uma das folhas (árvore é desequilibrada)

$$T(N, L) = \begin{cases} 1 & , N=0 \\ 1 & , L=0 \\ 1 + T(N-1, L-1) & , L \neq 0 \wedge N \neq 0 \end{cases}$$

Neste caso, utilizaram-se duas regras para cada caso.

Regras gerais:

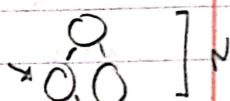
$$T(N, L) = \begin{cases} \Theta(1) & , N=0 \\ \Theta(1) & , L=0 \\ \Theta(1) + T(N', L-1) & , N \neq 0, L \neq 0 \end{cases}$$

$$N' = \begin{cases} N-1 & , \text{pior caso} \\ 0 & , \text{melhor caso} \end{cases}$$

17. - Input $\equiv N \equiv n^{\circ}$ modos da árvore

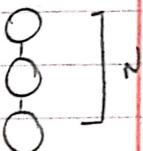
Melhor caso: árvore perfeitamente equilibrada

$$T(N) = \begin{cases} 1 & , N=0 \\ 1 + 2 \cdot T\left(\frac{N-1}{2}\right), N>0 \end{cases}$$



Pior caso: árvore perfeitamente desequilibrada (árvore - linha)

$$T(N) = \begin{cases} 1 & , N=0 \\ 1 + T(N-1), N>0 \end{cases}$$



Resolução:

C.R.	Node	tree	Cost
$T(N)$	0	$T(N)$	1
$T(N-1/2)$	1	$T\left(\frac{N-1}{2}\right)$	2
$T(N-3/4)$	2	$T\left(\frac{N-1}{2}\right)$	4
$T(N-7/8)$	3	$T\left(\frac{N-1}{2}\right)$	8

$$T(1) \xrightarrow{\log_2 N}$$

$$\frac{N - (2^{K-1} + K - 1)}{2^K} \Rightarrow 1 \Leftrightarrow$$

$$\Leftrightarrow N = 2^K + 2^{K-1} - K + 1$$

$$T(N) = (\log_2(N)) \times 2^K = \log_2 N \cdot (N - 2^K + K - 1) = \log_2 N \cdot (N - 2^{\log_2 N} + \log_2 N - 1) = \log_2 N \cdot (N - 2^{\log_2 N+1} + \log_2 n - 1) = \log_2 N \cdot ((\log_2 N)^2 - \log_2 n)$$

N	0	N	1
$N-1$	1	$N-1$	1
$N-2$	2	$N-2$	1
$N-3$	3	$N-3$	1
$N-4$	4	$N-4$	1
$T(1)$	K	$N-K$	1

$$\sum_{K=0}^N 1 = N-0+1 = N+1 = \Theta(N) \quad \Theta(\log_2 N^2)$$

48. • $N \in$ sequência de bits

$$T(N) = \begin{cases} 1 & , N=0 \\ 1 + 2 \cdot T(N-1) & , N > 0 \end{cases}$$

chamada	Nível	Árvore	Custo
Recursiva			
$T(N)$	0	$T(N)$	1
$T(N-1)$	1	$T(N-1)$	$T(N-1)$
$T(N-2)$	2	$T(N-2)$	$T(N-2)$
...		$T(N-2)$	$T(N-2)$
$T(N-K)$	K	$T(1)$	$T(1)$
		$T(1)$	$T(1)$
		$T(1)$	2^K

$$\cdot T(1) = T(N-K) \Leftrightarrow 1 = N-K \Leftrightarrow N = K$$

• Existem n níveis, logo:

$$T(N) = \sum_{K=0}^N 2^K = 2^{N+1} - 1 = 2^N \cdot 2 - 1 = \Theta(2^N)$$

51. • A função compara a posição i com a posição 0 ; sendo que se as duas forem iguais retorna imediatamente 0;

• Caso contrário, o ur vai ser o resultado de aplicar a função ao array, mas nas posições seguintes.

Melhor caso : Existem duplicados no array exatamente na posição 0 e 1. Ser feitas 1 comparação $V[0] == V[i]$, logo :

$$T(N) = 1 = \tau(1)$$

Pior caso : Não existem elementos duplicados no array . Ser feitas comparações entre a posição i e todas as outras:

$$T(N) = \sum_{i=1}^{N-1} 1 + T(N-1) = N-1 - 1 + 1 + T(N-1) =$$

$$= N-1 + T(N-1) \Rightarrow T(N-1) + N - 1$$

Resolvendo :

$$T(N) = T(N-1) + \underline{(N-1)}$$

Chamada Recursiva	Nível (k)	Gravar	custo
$T(N)$	0	$T(N)$	$N - 1$
$T(N-1)$	1	$T(N-1)$	$N - 1 - 1 = N - 2$
$T(N-2)$	2	$T(N-2)$	$N - 2 - 1 = N - 3$
$T(N-3)$	3	$T(N-3)$	$N - 4$
...
$T(N-k)$	k	$T(N-k)$	
$T(1)$			

- $T(1) = T(N-k) \Leftrightarrow N = k$

- Existem k níveis com custo de $N-k-1$:

$$T(N) = \sum_{K=0}^N N - k - 1 = \sum_{K=0}^N N - \sum_{K=0}^N k - \sum_{K=0}^N 1$$

$$(A) \quad \sum_{K=0}^N N = N^2 \quad (A) - (B) - (C)$$

$$(B) \quad \sum_{K=0}^N k = \frac{(N+0)(N-0+1)}{2} = \frac{N(N+1)}{2}$$

$$(C) \quad \sum_{K=0}^N 1 = N - 0 + 1 = N + 1$$

Então

$$T(N) = N^2 - \left[\frac{N(N+1)}{2} \right] - (N+1) \approx \Theta(N^2)$$

22. $4.2x^5 - 3.2x^2 - 0.5$

(5) (2) (0)

$$-0.5 ; 0 ; -3.2 ; 0 ; 0 ; 4.2 \quad \left| \begin{array}{l} i = \text{grau} \\ \sqrt{[Ei]} = \text{coef} \end{array} \right.$$

$T(A, M) \rightarrow$ multiplicações
 ↳ adição

$$T(A, M) = 1 + \sum_{i=0}^{N-1} 1 \cdot A + 1 \cdot M \xrightarrow{\text{valores constantes}} 1 + A \sum_{i=0}^{N-1} 1 + M \sum_{i=0}^{N-1} 1 =$$

$$= 1 + A \cdot N + M \cdot N = N(A+M) + 1 = \Theta(N(A+M))$$

24. Dados: $T_{\text{break-list}}(N) = \Theta(N)$
 \downarrow divide a lista para metade

- Seja N o comprimento da lista ligada de inteiros,

$$T(N) = \begin{cases} 1 & , N=0 \wedge N=1 \\ 1 + T\left(\frac{N}{2}\right) + T\left(\frac{N}{2}\right) + T_{\text{break-list}}(N), & N>1 \end{cases}$$

$$\downarrow$$

$$\Theta(1) + 2 \cdot T(N/2) + \Theta(N)$$

chamada	nível (k)	árvore	custo
$T(N)$	0	$T(N)$	$N+1$
$T(N/2)$	1	$T(N/2)$ $T(N/2)$	$(N/2+2)N+2$
$T(N/4)$	2	$T(N/4)$ $T(N/4)$ $T(N/4)$ $T(N/4)$	$\frac{N}{4} \cdot 4 + 4 = N+4$
$T(N/2^k)$	k	$T\left(\frac{N}{2^k}\right)$	\dots
$T(1)$.	.	.

$$\bullet T(1) = T\left(\frac{N}{2^k}\right) \Rightarrow \frac{N}{2^k} = 1 \Leftrightarrow N = 2^k \Leftrightarrow k = \log_2 N$$

$$\bullet T(N) = \sum_{k=0}^{\log_2 N} N + (k+1) = \underbrace{\sum_{k=0}^{\log_2 N} N}_{N \cdot (\log_2 N + 1)} + \underbrace{\sum_{k=0}^{\log_2 N} k}_{\frac{\log_2 N \cdot (\log_2 N + 1)}{2}} + \underbrace{\sum_{k=0}^{\log_2 N} 1}_{\log_2 N + 1} =$$

$$= \frac{2N(\log_2 N + 1)}{2} + \frac{(\log_2 N)^2}{2} + \frac{\log_2 N}{2} + \frac{2(\log_2 N + 1)}{2} =$$

$$= \frac{2N \log_2 N}{2} + \frac{2N}{2} + \frac{5 \log_2 N + 2}{2} = N \log_2 N + N + \frac{5}{2} \cdot \log_2 N + 1$$

$$\text{Alternativa} \Rightarrow \sum_{k=0}^{\log_2 N} N + 2^k = N \cdot \log_2 N + (2N - 1) \quad \left| \begin{array}{l} \Theta(N \cdot \log_2 N) \\ = \Theta(N \cdot \log_2 N) \end{array} \right.$$

- 25.
- seja $N \in \text{input} \geq n$ elementos da árvore
 - Árvore perfeitamente balanceada
 - Elemento pertence à BTREE
 - Elemento a procurar com igual probabilidade em qualquer posição

caso
base
 $\Theta(\log N)$

$$\bar{T}(N) = p_{\text{succ}} \cdot \bar{T}(N) + p_{\text{ans}} \cdot \bar{T}_{\text{ans}}(N)$$

Melhor caso: elemento está na raiz

$$p_{\text{succ}} \cdot \bar{T}(N) =$$

$$\frac{1}{N} \cdot \Theta(1) =$$

$$= \frac{1}{N}$$

Pior caso:

elemento a procurar está numa das folhas

$$\begin{aligned} \bar{T}(N) &= \sum_{k=0}^{\log_2 N} \xrightarrow{①} [a \rightarrow \text{valor} > x] \cdot \frac{1}{N} = \\ &= \frac{(\log_2 N + 1)}{N} \end{aligned}$$

Então

$$\begin{aligned} \bar{T}(N) &= \frac{1}{N} + (\log_2 N + 1) \cdot \frac{1}{N} = \\ &= \frac{1}{N} (\log_2 N + 1 + 1) = \\ &= \frac{1}{N} \cdot \log_2 N + 2 \approx \Theta(\log_2 N) \end{aligned}$$

(b) Sendo a árvore balanceada, se o elemento não existe, tem-se:

$$\begin{aligned} T(N) &= \underset{\substack{\text{worst} \\ \text{case}}}{\sum_{k=0}^{\log_2 N} 1} \cdot p_{\text{ans}} = \log_2 N \\ &= 1. \end{aligned}$$

↓
o elemento não existe mas há sucesso!

∴ O tempo médio aproxima-se ao constante do pior caso.

- 28.
- $\bar{T}_{\text{quantes}}(N) = \Theta(N)$;
 - Aleatoriedade dos valores do vector; $(1/N)$

• Adota-se a estratégia de $\bar{T}(N) = \bar{T}_{\text{succ}}(N) \cdot p_{\text{succ}}$
 $+ \bar{T}_{\text{ans}}(N) \cdot p_{\text{ans}}$

- Pior caso: Mediana está no índice $N-1$ (última posição)

$$\begin{aligned} \bar{T}(N) &= \frac{1}{N} \cdot \sum_{i=0}^{N-1} N + 1 = \\ &= \frac{1}{N} (N \cdot N + 1) = N + \frac{1}{N} \end{aligned}$$

- Melhor caso: Mediana está na posição 0 (1ª posição)

$$\begin{aligned} \bar{T}_{\text{melhor}}(N) &= \sum_{i=0}^{N-1} \frac{1}{N} \cdot (i+1) = \frac{N+1}{2} \\ \bar{T}(N) &= \overbrace{\frac{1}{N} \cdot \bar{T}(N)}^{\approx 0} + \overbrace{\frac{1}{N} \cdot \bar{T}(N)}^{\approx m} + \overbrace{\frac{1}{N} \cdot \bar{T}(N)}^{\left(1 - \frac{1}{2^k}\right)^N \approx 1} \\ &= 0 \cdot \left(\frac{N+1}{2}\right) + 1 \cdot \left(\frac{N+1}{N}\right) = \\ &= N + \frac{1}{N} \rightarrow \Theta(N) \\ &\quad \approx 0 \end{aligned}$$

29. • Calcular raiz quadrada (int) de um número;

- $N \in \text{nº bits do número em questão}$

- Adição, subtração e multiplicação em $\Theta(1)$

$$N = 2 \cdot (10)_2$$

$$\sqrt{2} = 1,4 \approx 1$$

Quanto maior for o valor

R	m	$r \times r$	$n < h$	de m, mais trabalhoso é
1	2	1	sim	o algoritmo
2	2	$2 \times 2 = 4$	nao	
1	2			

no bits

Pior caso: n é um número $2^{\frac{N}{2}} - 1$. O ciclo while é feito tantas vezes quanto o número de bits do elemento

$$T(N) = \sqrt{N} = O(\sqrt{N})$$

Melhor caso: 2^{N-1} :

$$T(N) = \sum_{r=1}^{\sqrt{m-1}} \Theta(1)$$

$$\sqrt{m-1} \leq r \leq \sqrt{m-1}$$

$$\sum_{r=1}^{\sqrt{m-1}} \Theta(1) = \frac{\sqrt{m-1} - 1 + 1}{2} = \frac{m-1}{2} = \Theta(m^{1/2}) = \Theta(\sqrt{m})$$

31. Resente: comprimento do maior prefixo crescente de $v[1:N]$
marcuse: comprimento do maior segmento crescente de $v[1:N]$

• Melhor caso: há um n° mínimo de iterações do ciclo while, ou seja, n° mínimo de chamadas recursivas de "crescente". É o caso em que a função crescente falha a comparar $v[i] < v[i-1]$, sempre que é chamada, terminando o ciclo. $T(N) = 1 = \omega(1)$ (ocorre que o vetor é decrescente estritamente).

$$T(N) = \sum_{i=0}^{N-2} T(N)_{\text{crescente}} = \sum_{i=0}^{N-2} 1 = N-2-1+1 = \omega(N)$$

• Pior caso: há um n° máximo de iterações, ou seja, n° máximo de chamadas recursivas de "crescente". É o caso em que a função só falha quando o ciclo termina. $T(N) = \sum_{i=1}^{N-1} 1 = N-1 = \Theta(N)$ ocorre quando o vetor é estritamente crescente:

$$T(N) = \sum_{i=0}^{N-2} T(N)_{\text{resente}} = \sum_{i=0}^{N-2} (N-i-1) = \Theta(N^2)$$

↓
até que
a dimensão
do vetor
muda

32. Melhor caso: os elementos são todos iguais.

$$T(N) = \begin{cases} 1, & N=0 \\ T(N-1)+1, & N>0 \end{cases}$$

! Distingue
 $\Theta(1)$ de $\underline{\underline{1}}$

Pior caso: os elementos são todos diferentes

$$T(N) = \begin{cases} 1, & N=0 \\ T(N-1)+(N-1), & N>0 \end{cases}$$

$\sum_{i=1}^{N-1} 1 = N-1-1+1 = \underline{\underline{N-1}} \neq \Theta(1)$

33. • N° médio de comparações feitas no array seu "crescente"
• Probabilidade de cada 0.5

$$\bar{T}(N) = \bar{T}_{\text{succ}} \cdot p_{\text{succ}} + \bar{T}_{\text{ins}} \cdot p_{\text{ins}}$$

$$\cdot \bar{T}_{\text{ins}}(N) = \bar{T}_{\substack{\text{Pior} \\ \text{caso}}} = \sum_{i=1}^{N-1} 1 \cdot \frac{1}{N} = (N-1) \cdot \frac{1}{N} = \frac{N-1}{N} = 1 - \frac{1}{N}$$

$$\lim_{x \rightarrow \infty} \left(\frac{1}{N} \right) = \frac{1}{\infty} \approx 0^+$$

$$\cdot \bar{T}_{\text{succ}}(N) = \bar{T}_{\substack{\text{melhor} \\ \text{caso}}} = \sum_{i=1}^{N-1} \frac{1}{N} \cdot \underbrace{(i)}_{\substack{\text{custo} \\ \downarrow \\ \text{fita}}} = \frac{1}{N} \cdot \left[\frac{(N-1+1)(N-2)}{2} \right] =$$

$$= \frac{N(N-2)}{2} \cdot \frac{1}{N} = \frac{N(N-2)}{2N} = \frac{N-2}{2} =$$

$$= \frac{N-1}{2}$$

∴

$$\begin{aligned} \bar{T}(N) &= \left(\frac{N-1}{2} \right) \cdot \frac{1}{2} + \left(1 - \frac{1}{2} \right) = \\ &= \frac{N}{4} - \cancel{\frac{1}{2}} + \cancel{\frac{2}{2}} - \cancel{\frac{1}{2}} = \frac{N}{4} = \Theta(N) \end{aligned}$$

	Custo	Prob	
MC :	1 2 3 ⋮ L	1/2 1/2 · 1/2 1/2 · 1/2 · 1/2 ⋮ $(1/2)^i$	$\bar{T}(N) = \sum_{n=1}^{N-1} c_n \cdot p_n =$ $= \sum_{i=1}^{N-1} i \cdot \underbrace{\left(\frac{1}{2}\right)^i}_{\substack{i \leq N-1}}$
PC :	N-1		$(1/2)^i \quad (1 \leq i \leq N-1)$

35. Melhor caso: o elemento da última posição é superior à dimensão do array.

$$T(N) = 1 = \Theta(1) \quad \left\{ \Theta(1), N=0 \vee N=1 \right.$$

Pior caso: menor elemento do array é superior à dimensão do mesmo

$$T(N) = \begin{cases} 1 & , N=1 \wedge N=0 \\ 1 + T(N-1) & , N > 1 \end{cases} \Rightarrow \Theta(N)$$

37.

(a) Pior caso: os bits no array char são do gênero:

$$\text{1111111} = 2^N - 1$$

$$T(N) = \sum_{i=0}^{N-2} 1 = N-2+0+1 = N-1$$

Seriam alterados $N-1$ bits, todos

(b)

Eixo	Ciclo 1; Ciclo 2	Ciclo 1; Ciclo 2
MC Alterar 0 bits	0; $N-1$	$\frac{1}{2}; \frac{1}{2}$
Alterar 1 bit	1; $N-2$	$(\frac{1}{2})^2; (\frac{1}{2})^2$
⋮	⋮	⋮
Alterar K bits	$k; N-(K+1)$	$(\frac{1}{2})^{K+1}$

$$\bar{T}(N) = \sum_{k=0}^{N-1} \frac{N-(k+1)}{2^{k+1}} \approx N-2 - \frac{2N-1}{2^{N-1}}$$

$$\frac{\text{C. ciclo 2}}{\text{ciclo 2}} \cdot \frac{p}{\text{ciclo 2}} =$$

so fuires em
conta o ciclo 2

51

- fila com a utilização de 2 stacks
- Stack A
- Stack B

• inserção: pop(SA)

• remoções:

$\#SB > 0 \Rightarrow \text{pop(SB)}$

c.c. $\Rightarrow \text{move(SA, SB)}, \text{pop(SB)}$

pop(SA)

• $T(\text{push}) = T(\text{pop}) = \Theta(1)$ push(B)

• $T_{\text{push}}(n) = O(n)$

OBJ: $T(\text{deque}) = T(\text{enque}) = \Theta(1)$

usar

$$\phi(Q) = 2 \cdot \text{size}(QA)$$

(Pot)

Pelo método do Potencial, assuma-se um estado i e seu antecessor $i-1$, tem-se:

$$\hat{c}_i = c_i + \phi_0 - \phi_{i-1}$$

- Pot é máximo quando stack está cheia (valor = N)
- Pot é mínimo quando stack está vazia (valor = 0)

$$Pot = 2 \cdot Tamanho (Q.A)$$

→ Se a stack ^B está cheia: (inseri elemento)

- $c = 1$
- Ocup-depois = Ocup-antes + 1
- Livres-depois = Livres-antes - 1
- Pot-depois = Ocup-depois - Livres-depois =
= $(Ocup\text{-antes} + 1) - (Livres\text{-antes} - 1) =$
= Ocup-antes - Livres-depois + 2
- Pot-antes = Ocup-antes - Livres-antes

$$\hat{c}_i = 1 + Pot\text{-depois} - Pot\text{-antes} = 3$$

→ stack ^B cheia

- $c = N+1$
- Ocup-depois = Ocup-antes + i = $N+1$
- Livres-depois = $N - 1$

$$\hat{c}_i = (N+1) + (N+1 - (N-1)) - (N-0) = 3$$