Lenir Moreira 9,2 Exagerou!!

1º) Um algoritmo de cota superior é o melhor algoritmo conhecido para um determinado problema. ~~[illegible]~~ É aquele cuja complexidade temporal não ultrapasse a cota superior do problema. ✓1,0

Como exemplo temos o algoritmo de Coppersmith e Winograd que resolve multiplicação de matrizes em tempo $n^{2,36}$. Ele é um algoritmo de cota superior, pois sua complexidade temporal não ultrapassa a cota superior do problema, que ele próprio definiu. Um outro algoritmo que resolvesse o mesmo problema com o mesmo tempo também seria de cota superior.

2º) $T(n) = T(\frac{n}{2}) + n^2$, Substituindo $n = 2^k$

$0 \quad T(2^k) = T(\cancel{2^{k-1}}) + (2^k)^2$

$1 \quad T(\cancel{2^{k-1}}) = T(\cancel{2^{k-2}}) + (2^{k-1})^2$

$2 \quad T(\cancel{2^{k-2}}) = T(\cancel{2^{k-3}}) + (2^{k-2})^2$

$\vdots$

$k-2 \quad T(\cancel{2}) = \cancel{T}(1) + (2)^2$

$k-1 \quad T(\cancel{1}) = 1$

✓1,0

$$T(n) = \sum_{i=0}^{i=k} 4^k = 1\left(\frac{1-4^{k+1}}{1-4}\right) = \frac{4^{k+1}-1}{3} = \frac{4 \cdot 4^k - 1}{3} = \frac{4n^2-1}{3} \quad \Theta(n^2)$$

3º) Melhor caso se dá quando o ~~alg~~ vetor já está ordenado (crescente) desse jeito o algoritmo não entrará no loop do enquanto, executando apenas o loop externo, lhe conferido uma complexidade de melhor caso de $O(n)$.

O pior caso ocorre quando o vetor está ordenado de ordem decrescente, assim para cada iteração do loop externo o loop interno também será executado.

A complexidade nesse pior caso é $O(n^2)$, pois o loop externo executa $n-1$ vezes e o interno $n-2$ vezes (quando $i = n-1$, $j = n-2$ e o loop interno itera até $j=0$, por isso $n-2$ vezes). Assim a complexidade temporal seria $(n-1)(n-2) \in O(n^2)$.

A complexidade espacial do algoritmo é constante, pois ele usa as variáveis extra pivo, j e i que tem tamanho constante.

O algoritmo é eficiente, pois ~~executa~~ sua complexidade temporal $O(n^2)$ é limitada por um polinômio no tamanho da entrada (n).

2,7

O algoritmo não é de cota inferior, pois sua complexidade temporal não é igual à complexidade intrísecã ao problema de ordenação (presumivelmente $\Omega(n)$, já que a saída é um vetor ordenado de tamanho n).

4ª a)

chamada 1

Inicio = 0
fim = 4

Inicio é menor que fim
Troca os elementos das posições V[0] e V[4]
L = {8, 5, 4, 17, 12}

Faz chamada recursiva

Chamada 2

Inicio = 1
fim = 3

Inicio é menor que fim
Troca os elementos das posições V[1] e V[3]
L = {8, 17, 4, 5, 12}

Faz chamada recursiva

chamada 3

Inicio = 2
fim = 2

Inicio não é menor que fim

L = {8, 17, 4, 5, 12}

0,5

d) $T(n) = T(\cancel{n-2}) + C$

1 $T(\cancel{n-2}) = T(\cancel{n-4}) + C$

2 $T(\cancel{n-4}) = T(\cancel{n-6}) + C$

⋮ ⋮

n-1 $T(\cancel{1}) = C$

$T(n) = C \cdot n \quad \Theta(n)$.

Complexidade Temporal.

O algoritmo tem complexidade espacial constante, já que é uma ~~algoritmo~~ recursão de cauda.

O algoritmo é eficiente, pois executa um número de instruções elementares limitada por um polinomio no tamanho de entrada.

É de cota inferior, pois sua complexidade temporal é igual a complexidade intrísica* do problema

(*Já que a saída é um vetor com n posições, supõe-se $\Omega(n)$.)

O algoritmo serve para inverter a ordem dos elementos de um vetor.

1,5

e) Primeiramente, sabemos que o algoritmo para. Visto que inicio sempre terá ~~decrementado~~ um valor maior que antes (que a chamada anterior). O Fim sempre terá um valor menor que oda chamada anterior, até que inicio ≥ fim, quando a chamada recursiva não acontecerá.

Façamos indução em $n = -inicio + fim + 1$, o tamanho do vetor

1,0

Caso base, $n = 1$, isso indica que inicio=fim e o vetor já está invertido.

HI: Suponha que para [illegible] um vetor de tamanho $n-2$, o algoritmo funciona.

Sendo assim, quando o algoritmo for chamado para um vetor $0 \ldots n-1$ ou seja de tamanho n, ele efetuará a troca dos elementos no extremo do vetor e depois irá chamar o algoritmo recursivamente sobre os elementos de $1 \ldots n-2$, que é um vetor de tamanho $n-2$ que pela HI o algoritmo resolve.

5º Entrada: um vetor indexado desde o 0, de tam n, e ordenado. [illegible] Moreira
um valor x a ser buscado.

Saída: VERDADEIRO se x está no vetor
falso se x não está no vetor

Algoritmoo BB

Se inicio > fim
devolva falso
Senão
$$meio = \left\lfloor \frac{ini + fim}{2} \right\rfloor$$
Se x = V[meio] devolva verdadeiro
se x < V[meio] devolva bb(X, V, inicio, meio-1)
devolva bb (X, V, meio+1, Fim)

SEU ALGORITMO DEVERIA FUNCIONAR NUM VETOR NÃO ORDENADO

1,5

Para determinar a complexidade temporal:
$$T(n) = T\left(\frac{n}{2}\right) + C, \quad T(1) = C$$
Substitua $n = 2^k$

$$T(n) = (k+1) \cdot C = (\log_2 n + 1) \cdot C \quad \Theta(\log n)$$

$0 \; T(2^k) = \cancel{T(2^{k-1})} + C$
$1 \; \cancel{T(2^{k-1})} = T(\cancel{2^{k-2}}) + C$
$2 \; \cancel{T(2^{k-2})} = T(\cancel{2^{k-3}}) + C$
$\vdots$
$k-1 \; T(\cancel{2^{1}}) = \cancel{T(1)} + C$
$k \; T(\cancel{2^0}) = C$

A complexidade espacial é constante, pois este é um algoritmo de cauda.

O algoritmo é eficiente, pois é logaritmico no tamanho de entrada

~~Não alocamos nada a fazer~~

Prova:
é fácil ver que se o vetor não contém elementos ini > fim é verdadeiro e o algoritmo retorna falso. Correto.

HI Suponha que o algoritmo funciona para um vetor de tamanho n-1. Na hora de executar o algoritmo sobre o vetor de tamanho n, caso $X \neq V[meio]$ ele fará uma chamada recursiva sobre um vetor de tamanho aproximadamente a metade do vetor original*. Mas pela hipotese de indução o algoritmo funciona para vetores de tamanho n-1, que é um vetor maior que um de tamanho $\frac{n}{2}$, então o algoritmo funciona para um vetor de tamanho $\frac{n}{2}$.

Observe que inicio e fim são sempre chamados com valores novos a cada chamada recursiva. Inicio sempre será incrementado ou fim será decrementado. Assim chegará um momento que inicio > fim. E o algoritmo para. Observe também que se x está no vetor, por conta do valor de inicio e fim estarem mudando constantemente, haverá um momento em que meio será o indice de V tal que $V[meio] = X$.

B

* Não é que um novo vetor é passado a cada chamada, mas sim um novo "range" (limite) é definido. Assim é mais fácil de explicar, por isso utilizei "novo vetor".

Desconheço a cota inferior do problema.