Revisão P1

1: Sejam $f(n)$, $g(n)$ e $h(n)$ funções não-negativas tais que $f(n) = O(h(n))$ e $g(n) = O(h(n))$. Então $f(n) + g(n) = O(h(n))$.

Pela definição da notação assintótica O, sabemos que:

$f(n) = O(h(n))$: Existem constantes positivas c_1 e n_1 tais que:

$$f(n) \leq c_1 \cdot h(n), \quad \forall n \geq n_1.$$

$g(n) = O(h(n))$: Existem constantes positivas c_2 e n_2 tais que:

$$g(n) \leq c_2 \cdot h(n), \quad \forall n \geq n_2.$$

Nosso objetivo é provar que $f(n) + g(n) = O(h(n))$, ou seja, queremos mostrar que existem constantes positivas c e n_0 tais que:

$$f(n) + g(n) \leq c \cdot h(n), \quad \forall n \geq n_0.$$

Somamos as duas desigualdades $f(n) \leq c_1 \cdot h(n)$ e $g(n) \leq c_2 \cdot h(n)$. Para todo $n \geq \max(n_1, n_2)$, temos:

$$f(n) + g(n) \leq c_1 \cdot h(n) + c_2 \cdot h(n).$$

Colocando $h(n)$ em evidência:

$$f(n) + g(n) \leq (c_1 + c_2) \cdot h(n).$$

Definimos: - $c = c_1 + c_2$, - $n_0 = \max(n_1, n_2)$.

Assim, para todo $n \geq n_0$, temos:

$$f(n) + g(n) \leq c \cdot h(n).$$

Pela definição da notação O, concluímos que:

$$f(n) + g(n) = O(h(n)).$$

2: Sejam f (n) e g(n) funções não-negativas tais que g(n) = O(f (n)). Prove, utilizando as definições de notação assintótica, que f (n) + g(n) = (f (n)).

Pela definição da notação assintótica O, sabemos que:

$$g(n) = O(f(n)) \implies \text{existem constantes positivas } c_1 \text{ e } n_1 \text{ tais que:}$$

$$g(n) \leq c_1 \cdot f(n), \quad \forall n \geq n_1.$$

Queremos provar que $f(n) + g(n) = \Theta(f(n))$, ou seja, mostrar que existem constantes positivas c_2, c_3 e n_0 tais que:

$$c_2 \cdot f(n) \leq f(n) + g(n) \leq c_3 \cdot f(n), \quad \forall n \geq n_0.$$

Passo 1: Limitante superior
Começamos mostrando que $f(n) + g(n) \leq c_3 \cdot f(n)$.
Sabemos que $g(n) \leq c_1 \cdot f(n)$ para $n \geq n_1$. Logo:

$$f(n) + g(n) \leq f(n) + c_1 \cdot f(n).$$

Colocando $f(n)$ em evidência:

$$f(n) + g(n) \leq (1 + c_1) \cdot f(n).$$

Definimos $c_3 = 1 + c_1$. Assim, temos:

$$f(n) + g(n) \leq c_3 \cdot f(n), \quad \forall n \geq n_1.$$

Passo 2: Limitante inferior Agora mostramos que $f(n) + g(n) \geq c_2 \cdot f(n)$.
Como $f(n)$ é uma função não-negativa, temos:

$$f(n) + g(n) \geq f(n).$$

Portanto, basta escolher $c_2 = 1$, e temos:

$$f(n) + g(n) \geq c_2 \cdot f(n), \quad \forall n \geq 0.$$

Passo 3: Conclusão Juntando os resultados dos passos 1 e 2, concluímos que:

$$c_2 \cdot f(n) \leq f(n) + g(n) \leq c_3 \cdot f(n), \quad \forall n \geq n_0,$$

onde $c_2 = 1$, $c_3 = 1 + c_1$, e $n_0 = n_1$.
Pela definição de Θ, temos que:

$$f(n) + g(n) = \Theta(f(n)).$$

3: Faça a análise da complexidade do melhor caso para este algoritmo.

Solução. No melhor caso, o elemento procurado está na posição central, e portanto não teremos chamadas recursivas na execução do algoritmo. Logo Tb(n) = (1).

4: Faça a análise da complexidade do pior caso para este algoritmo.

No pior caso, o elemento procurado não está presente no vetor. O algoritmo realiza uma comparação em cada passo e reduz o tamanho do problema pela metade a cada iteração (ou chamada recursiva).

A complexidade do pior caso pode ser expressa pela seguinte relação de recorrência:

$$T_w(n) = T_w\left(\frac{n}{2}\right) + \Theta(1),$$

onde: - $T_w(n)$: Tempo de execução no pior caso para um vetor de tamanho n, - $\Theta(1)$: O tempo constante gasto em cada passo para comparar o elemento com o valor central e decidir o próximo subproblema.

Resolvendo a recorrência com o TM: A relação de recorrência $T_w(n) = T_w(n/2) + \Theta(1)$ está na forma padrão:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

onde: - $a = 1$: Número de subproblemas, - $b = 2$: Fator de divisão do problema, - $f(n) = \Theta(1)$: Custo fora das chamadas recursivas.

Aplicamos o TM:

Conclusão:

A complexidade do pior caso para este algoritmo é:

$$T_w(n) = O(\log(n)).$$

Isso reflete o fato de que, no pior caso, o algoritmo realiza $\log_2(n)$ divisões até alcançar um tamanho de problema irrelevante (subproblema vazio).

5: A correção deste algoritmo pode ser estabelecida em duas etapas. A primeira delas consiste em provar que se a chave key não ocorre no vetorA[1..n],então BinarySearch(A[1..n], 1, n, key) retorna o valor -1. Prove o lema a seguir: Seja A[1..n] um vetor ordenado de inteiros distintos. Mostre que se a chave key não ocorre em A[1..n], então BinarySearch(A[1..n], 1, n, key) retorna o valor -1.

A demonstração será feita por **indução** sobre o tamanho do vetor n.

—

Base da indução $[n = 1]$: Se $n = 1$, o vetor $A[1..n]$ contém apenas um elemento. Seja $A[1]$ o único elemento do vetor. Temos dois casos: 1. Se key $\neq A[1]$, a busca binária verifica $A[1] \neq$ key e retorna -1, pois não há outros elementos para verificar. 2. Se key $= A[1]$, o algoritmo retorna a posição 1, mas isso não se aplica ao enunciado, que considera key ausente.

Portanto, para $n = 1$, a busca binária retorna -1 se key não estiver presente.

Passo indutivo: Assuma que o lema é válido para vetores de tamanho n: se a chave key não ocorre em $A[1..n]$, então BinarySearch$(A[1..n], 1, n, \text{key})$ retorna -1.

Agora, provamos que o lema vale para $n+1$, ou seja, para um vetor $A[1..n+1]$.

1. O algoritmo de busca binária escolhe o **elemento central** do vetor, $A[m]$, onde $m = \lfloor (1 + n + 1)/2 \rfloor$. 2. Há três casos: - Caso 1: Se key $= A[m]$, o algoritmo retorna m. Porém, isso não ocorre neste contexto, já que key $\notin A[1..n + 1]$. - Caso 2: Se key $< A[m]$, o algoritmo recursivamente chama BinarySearch$(A[1..m - 1], 1, m - 1, \text{key})$. Pelo passo indutivo, como key $\notin A[1..n + 1]$, também key $\notin A[1..m - 1]$, e a busca retorna -1. - Caso 3: Se key $> A[m]$, o algoritmo recursivamente chama BinarySearch$(A[m+1..n+1], m+1, n+1, \text{key})$. Pelo passo indutivo, como key $\notin A[1..n+1]$, também key $\notin A[m + 1..n + 1]$, e a busca retorna -1.

Conclusão: Para todos os casos possíveis, o algoritmo retorna -1 quando key $\notin A[1..n + 1]$. Pelo princípio da indução, o lema é válido para todos os $n \geq 1$.

6: Mostre como podemos ordenar n inteiros contidos no intervalo de 0 a n3 ou seja, em tempo O(n).