EXERCÍCIOS - Complexidade de Tempo de Algoritmos

Teoria da Computação – 2025-2

Dupla:

Gustavo Aragão Guedes - 10376534

João Pedro de Souza Costa Ferreira - 10400720

Observação 1: Este exercício não deve ser manuscrito e nem conter imagens com detalhes manuscritos.
Observação 2: Deve ser mantida a formatação do texto e os enunciados dos exercícios propostos.

1. Apresente, de forma direta e objetiva, a definição formal da notação assintótica Θ.

   A notação Theta é usada para descrever a complexidade assintótica exata de uma função, fornecendo uma limitação inferior e superior para o comportamento de crescimento de uma função. Ela indica que o limite superior e o limite inferior são da mesma ordem de grandeza.

   Visualização:

   f(n) = Θ(g(n)) se existem constantes positivas c_1, c_2 e n_0 tais que, para todo $n \geq n_0$,

   $$c_1\, g(n) \leq f(n) \leq c_2\, g(n).$$

   Θ(g(n)) significa que f(n) cresce assintoticamente no mesmo ritmo que g(n) (limite superior e inferior simultâneos).

2. A somatória dos números de 1 até n pode ser calculada através da fórmula

   somat(n)= ((1+n)/2)*n

   Escreva uma função (usando uma notação similar à linguagem C) que calcule a somatória de 1 até n usando esta fórmula e calcule a complexidade de pior caso da sua função obtida. Qual função é mais eficiente, a sua ou aquela anteriormente apresentada em aula? Justifique adequadamente.

   Função:

```
Int somat(int n) {
   return (n * (n + 1)) / 2;
}
```

   Complexidade (pior caso) = O(1). A execução realiza um número constante de operações aritméticas independentemente de n. Tempo constante.

   Complexidade do algoritmo dado em sala de aula = O(n). Tempo linear.

   A função mais eficiente é a nossa, pois o tempo de execução não depende do tamanho da entrada n, enquanto a função iterativa dada em aula cresce linearmente com n.

3. Escreva uma função para realizar busca sequencial em um vetor de números inteiros e calcule detalhadamente a complexidade da sua função nas seguintes situações:
   - melhor caso: quando o elemento buscado se encontra na primeira posição do vetor, e
   - pior caso: quando o elemento buscado não ocorre no vetor.

```
int busca_sequencial(int vet[], int n, int alvo) {
   for (int i = 0; i < n; i++) {
      if (vet[i] == alvo) return i; // encontrou
   }
   return -1; // não encontrou
}
```

Melhor caso

- Número de comparações = 1 (quando i = 0).
- Outras operações são constantes.
- Portanto o tempo é uma constante: **$T(n) = c \Rightarrow \Theta(1)$**.

Pior caso

- Número de comparações = n (testa todos os elementos).
- Mais algumas operações constantes (controle do loop, retorno).
- Portanto $T(n) = a \cdot n + b \Rightarrow \mathbf{\Theta(n)}$.

4. Considerando uma função chamada **`outraF(n)`** de complexidade de tempo $\Theta(n^1)$, qual será a complexidade de tempo (do melhor e do por caso) do código abaixo? Calcule detalhadamente. (Note que a linha do for foi quebrada em 3, pois cada parte tem uma quantidade de execuções diferente das demais.)

```
for (int i = 0;
                 i < 2*n;
                          i++)
   if (arr[i] < k) // k e constante
      outraF(n);
```

Linha de código	Quantidade de execuções	Tempo de 1 execução	Tempo total	Justificativa
int i = 0;	1	O(1)	O(1)	Inicialização do loop ocorre apenas uma vez
i < 2*n	(2n + 1) vezes	O(1)	O(n)	Teste de condição é feito a cada iteração + 1 vez no fim
i++	2n vezes	O(1)	O(n)	Executado a cada iteração do laço
if (arr[i] < k)	2n vezes	O(1)	O(n)	Comparação constante, feita em todas as iterações
outraF(n)	Depende do caso	$O(n^1) = O(n)$	Depende do caso	Executada somente se arr[i] < k for verdadeiro
Total	—	—	**O(n)**	—

Melhor caso:

- Nenhum elemento satisfaz arr[i] < k.
  ⇒ outraF(n) **nunca é executada**.
  - Tempo total = **Θ(n)** (das outras partes do laço).

Pior caso:

- Todos os elementos satisfazem arr[i] < k.
  ⇒ outraF(n) é executada **2n vezes**.
  - Cada chamada custa Θ(n).
  - Total = 2n × Θ(n) = **Θ(n²).**

5. Considerando uma função chamada **`outraF(n)`** de complexidade de tempo $\Theta(n^2)$, qual será a complexidade de tempo (do melhor e do por caso) do código abaixo? Calcule detalhadamente.

```
for (int i = 0;
                i < 100;
                       i++){
      outraF(n);
      outraF(n);
      outraF(n);
}
```

Linha / trecho	Qtde de execuções	Tempo de 1 execução	Tempo total (estim.)	Justificativa
int i = 0;	1	O(1)	O(1)	Inicialização do laço, executa 1 vez
i < 100	101	O(1)	O(1)	Teste de condição feito 100 iterações + 1 teste final → constante
i++	100	O(1)	O(1)	Incremento executado a cada iteração → constante
outraF(n) (1ª)	100	$O(n^2)$	$100 \cdot O(n^2) = O(n^2)$	Chamada incondicional por iteração
outraF(n) (2ª)	100	$O(n^2)$	$100 \cdot O(n^2) = O(n^2)$	Segunda chamada incondicional
outraF(n) (3ª)	100	$O(n^2)$	$100 \cdot O(n^2) = O(n^2)$	Terceira chamada incondicional
Total	—	—	$O(1) + O(1) + O(1) + 3 \cdot (100 \cdot O(n^2)) =$ **$O(n^2)$**	—

Como as chamadas a outraF(n) ocorrem **sempre** (são incondicionais dentro do laço de 100 iterações), **melhor caso = pior caso**:

- **Melhor caso:** $O(n^2)$
- **Pior caso:** $O(n^2)$