Lista 1 - Estrutura de Dados 1 – 2023.1

Prof. Ana Luiza Bessa de Paula Barros
Ciência da Computação – **UECE**

1) Para cada um dos trechos de código abaixo, analise o tempo estimado de execução no melhor e no pior caso. Considere que as variáveis n, m e vetor sejam dados de entrada.

a)

```
int soma = 0; ----------------------------------- 1
for (int i=0; i<n; i++){
      if ( vetor[i] % 2 == 0) //se for par ------ 1*n
          soma = soma + vetor[i]; ------------- 1*n
}
```

**Melhor caso*: $f(n) = 1 + n \rightarrow \Theta(n)$, se todos os elementos do vetor forem ímpares;
**Pior caso*: $f(n) = 1 + 2n \rightarrow \Theta(n)$, se todos os elementos do vetor forem pares;

b)

```
int soma1 = 0; --------------- 1
for (int i=0; i<n; i++){
     soma1 = soma1 + 1; --- 1*n
}
for (int j=0; j<n;j++){
      soma1 = soma1 + j; --- 1*n
}
```

- Em ambos os casos, ambos os loops executarão n vezes, conforme o tamanho da entrada. Portanto, sua complexidade seria calculada por $f(n) = 1 + n + n = 1 + 2n$, o que implica em $\Theta(n)$.

**Melhor caso*: $\Theta(n)$;
**Pior caso*: $\Theta(n)$;

c)

```
int soma = 0;-------------------------- 1
for (int i=0; i<n; i++){
     for (int j=0; j< n; j++){---------  1*n*n
           soma = soma + 1;
     }
}
```

- Em ambos os casos, ambos os loops externo e interno executarão n vezes, conforme o tamanho da entrada. Portanto, sua complexidade seria calculada por $f(n) = 1 + n * n = 1 + n^2$, o que implica em $\Theta(n^2)$.

**Melhor caso*: $\Theta(n^2)$;
**Pior caso:* $\Theta(n^2)$;

d)

```
int menor = MAIOR-INTEIRO; ----------- 1
for (int i=0; i<n; i++){
     if (vetor[i] < menor) ---------------------- 1*n
         menor = vetor[i];  ---------------------- 1* n
}
if (menor < 0){ -------------------------------- 1
     for (int i=0; i<n; i++){
          menor = menor * (i+1); --------------- 1*n
       }
}
```

***Melhor caso**: $f(n) = 3 + n \rightarrow \Theta(n)$, se o vetor estiver ordenado em ordem crescente, onde o menor elemento é maior ou igual a 0;

***Pior caso**: $f(n) = 2 + 3n \rightarrow \Theta(n)$, se o vetor estiver ordenado em ordem crescente, onde o menor elemento é negativo;

e)

```
int menor = MAIOR-INTEIRO; ----------- 1
for (int i=0; i<n; i++){
      if (vetor[i] < menor) --------------------- 1*n
           menor = vetor[i]; --------------------- 1*n
}
if (menor < 0){ -------------------------------- 1
    for (int i=0; i<n; i++){
          menor = menor * (i+1);
    }

}else if (menor > 0){ -------------------------- 1
       for (int i=0; i<n*n; i++)
            printf("%d\n", menor);--------------- 1*n*n

}else{
      printf("%d\n", menor); -------------------- 1
}
```

***Melhor caso**: $f(n) = 3 + 2 + n^2 \rightarrow \Theta(n^2)$, se o vetor for de ordem crescente e o menor elemento for zero;

***Pior caso**: $f(n) = 5 + n \rightarrow \Theta(n)$, se o vetor for de ordem decrescente e s o menor valor do vetor for positivo;

2) Mostre as seguintes propriedades

a) <u>Transitividade</u>

1. Se $f(n) = \Theta(g(n))$ e $g(n) = \Theta(h(n))$ então $f(n) = \Theta(h(n))$

- Isso significa que f(n)e g(n) têm a mesma ordem de crescimento, e g(n) e h(n) também têm a mesma a ordem de crescimento, então f(n) e h(n) têm a mesma ordem de crescimento.

2. Se $f(n) = O(g(n))$ e $g(n) = O(h(n))$ então $f(n) = O(h(n))$

- Essa propriedade significa que se f(n) é assintoticamente menor ou igual a g(n), e g(n) é assintoticamente menor ou igual a h(n), então f(n) também é assintoticamente menor ou igual a h(n).

3. Se $f(n) = \Omega(g(n))$ e $g(n) = \Omega(h(n))$ então $f(n) = \Omega(h(n))$
- Isso mostra que se f(n) é assintoticamente maior ou igual g(n), e g(n) é assintoticamente maior ou igual a h(n), então f(n) também é assintoticamente maior ou igual a h(n).

b) Reflexividade
1. $f(n) = \Theta(f(n))$
- Esta propriedade afirma que uma função de complexidade é assintoticamente equivalente a si mesma. Isso significa que a função cresce na mesma taxa para entradas de tamanho crescente.

2. $f(n) = O(f(n))$
- Esta propriedade estabelece que a complexidade de um algoritmo nunca excede sua própria complexidade. Em outras palavras, uma função que descreve a complexidade do algoritmo não pode crescer mais rápido do que ela mesma.

3. $f(n) = \Omega(f(n))$
- Esta propriedade afirma que a complexidade de um algoritmo nunca é menor do que sua própria complexidade. Ou seja, uma função que descreve a complexidade de um algoritmo não pode crescer mais devagar do que ela mesma.

c) Simetria
1. $f(n) = \Theta(g(n))$ se somente se $g(n) = \Theta(f(n))$
- Tal propriedade afirma que se duas funções tem a mesma ordem de crescimento, elas são equivalentes em termos de complexidade assintótica e que ambos os limites superior e inferior para f(n) e g(n) são iguais.

3) Classifique as seguintes ordens de complexidade de forma crescente
1. $O(n^2)$
2. $O(\log(n))$
3. $O(n^3)$
4. $O(2^n)$
5. $O(n\log(n))$
6. $O(n)$;

***Resposta:**
- Em ordem crescente:
$O(\log n) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(2^n)$

4) Verifique se cada questão abaixo é verdadeira ou falsa
(a) $10^{56}n^2 \in O(n^2)$ - (V)
$10^{56}n^2 \leq cn^2$; $c = 10^{56}$
$10^{56}n^2 = 10^{56}n^2$; $c = 10^{56}$

(b) $10^{56}n^2 \in O(n^3)$ - (V)
$10^{56}n^2 \leq cn^3$; $c=1$
$10^{56}n^2 \leq 1n^3$
$10^{56}n^2 / n^3 \leq 1$
$10^{56}/n \leq 1$

(c) $10^{56}n^2 \in O(n)$ – (F)
$10^{56}n^2 \leq cn$
$10^{56}n^2/n \leq c$
$10^{56}n \leq c$

(d) $2^{n+1} \in O(2^n)$ – (V)

$2^{n+1} \leq c2^n$

$2^n 2^1 \leq c2^n$; c = 2

$2^n 2^1 \leq 2^{2n}$; c = 2

(e) $2^n \in O(2^n)$ – (F)

$2^{2n} \leq c^{2n}$

$2^n 2^n \leq c^{2n}$

$2^n 2^n / 2^n \leq c$

$2^n \leq c$

(f) $n \in O(n^3)$ - (V)

$n \leq cn^3$

$n/n^3 \leq c$

$1/n^2 \leq c$

$n \leq cn^3$; c = 1

$n \leq n^3$; c = 1 ; a = 1

5) Dados dois vetores ordenados A e B, contendo cada um deles N números inteiros. Os vetores A e B precisam ser unidos em outro vetor C que terá 2 números, também ordenados. QuaÍ é a complexidade de tempo (big O notation) do processo de união dos dois vetores A e B?

(A) O(1), pois se precisa fazer apenas uma coisa simples de cada um dos elementos originais.
(B) O (log n), pois se usa a busca binária para determinar qual será o próximo elemento copiado para o vetor de destino.
(C) O(n), pois se precisa fazer uma cópia de cada um dos elementos originais, o que implica uma varredura de cada vetor de origem.
(D) O(n log n), pois se precisa fazer uma busca de cada elemento, para depois inseri-lo no vetor destino.
(E) $O(n^2)$ pois, como há dois vetores, precisa-se fazer dois laços de forma aninhada (um dentro do outro), gerando uma multiplicação das quantidades de elementos.

6) Sendo n o tamanho da entrada de um algoritmo para um problema P. Cada uma das alternativas abaixo corresponde a um algoritmo distinto, representando o número de operações necessárias para resolver P. Considerando-se a análise assintótica (Big O notation), qual algoritmo possui menor complexidade?

(A) 2 + 10 log n
(B) $3n^2 + n$
(C) $1000 + 2n^3$
(D) 5n + 128
(E) 4^n