Análise de Algoritmos e Estruturas de Dados
Centro de Matemática, Computação e Cognição
Universidade Federal do ABC
Profa. Carla Negri Lintzmayer

**Lista 1:** Tempo de execução, notação assintótica e corretude de algoritmos iterativos

## Instruções importantes

(1) Em qualquer exercício que peça para você fornecer um algoritmo/solução para um problema, a menos que explicitamente dito o contrário, você deve:

- descrever em palavras qual é a ideia do seu algoritmo (**obrigatório**);
- escrever um pseudocódigo (opcional, se a descrição do item anterior ficou clara o suficiente);
- provar ou fornecer um argumento intuitivo para sua corretude (**obrigatório**). É claro que, se o exercício explicitamente pede uma prova, então um argumento intuitivo não será o suficiente;
- analisar o tempo do seu algoritmo (**obrigatório**).

O não cumprimento dos itens acima implica que o exercício está incorreto e o mesmo será desconsiderado.

(2) Você pode utilizar qualquer algoritmo visto em aula sem reescrevê-lo ou provar sua corretude novamente, mesmo que o algoritmo necessite de alguma pequena alteração.

- Descrever clara e sucintamente o que o algoritmo recebe, o que ele devolve e qual o seu consumo de tempo.
- Se houver alterações, descreva-as, indicando, por exemplo, quais linhas estão sendo alteradas/acrescentadas.

1. Ordene a lista de funções a seguir por ordem crescente de taxa de crescimento:

$$f_1(n) = n^{2.5},\ f_2(n) = \sqrt{2n},\ f_3(n) = 10^n,\ f_4(n) = 100^n,\ f_5(n) = n^2 \log_2 n$$

$$f_6(n) = 34n^0,\ f_7(n) = 18902n^2,\ f_8(n) = 3\frac{n^{127}}{n^{34}},\ f_9(n) = 456^{8478},\ f_{10}(n) = \frac{n(n+1)}{5}$$

2. Para cada um dos Algoritmos 1, 2 e 3, forneça duas expressões, em notação assintótica, de descrição de tempo de execução, sendo uma para o melhor caso e outra para o pior caso. Nos dois primeiros algoritmos, ambos os valores $n$ e $m$ descrevem o tamanho da entrada, por isso suas expressões devem ser uma função sobre ambos. Use a notação $\Theta$ sempre que possível. Justifique sua resposta.

**Algorithm 1** Busca em dois vetores.

```
1: Função BuscaVetores(A, n, B, m, k)
2:     Para i = 1 até n faça
3:         Se A[i] == k então
4:             Devolve verdadeiro
5:     Para i = 1 até m faça
6:         Se B[i] == k então
7:             Devolve verdadeiro
8:     Devolve falso
```

**Algorithm 2** Busca por elemento em comum.

```
1: Função BuscaIntersecao(A, n, B, m)
2:     Para i = 1 até n faça
3:         Para j = 1 até m faça
4:             Se A[i] == B[j] então
5:                 Devolve verdadeiro
6:     Devolve falso
```

**Algorithm 3** Busca por elementos duplicados no mesmo vetor.

```
1: Função BuscaDuplicados(A, n)
2:     Para i = 1 até n faça
3:         Para j = i + 1 até n faça
4:             Se A[i] == A[j] então
5:                 Devolve verdadeiro
6:     Devolve falso
```

3. O algoritmo *Bubble Sort* é um clássico da computação. Ele recebe um vetor $A[1..n]$ e promete ordená-lo. Sua ideia é percorrer o vetor várias vezes, trocando a ordem de pares de elementos adjacentes que estejam fora de ordem. Seu pseudocódigo é dado a seguir:

```
1: Função BubbleSort(A, n)
2:     Para i = n até 2, decrementando faça
3:         Para j = 1 até i − 1, incrementando faça
4:             Se A[j] > A[j + 1] então
5:                 troque A[j] com A[j + 1]
```

Responda:

(a) Prove que a frase $P(x)$ = "Antes da iteração em que $j = x$ começar, $A[j]$ maior do que os elementos de $A[1..j-1]$." é uma invariante do laço interno.

(b) Prove que a frase $R(y)$ = "Antes da iteração em que $i = y$ começar, o subvetor $A[i+1..n]$ está ordenado e contém os maiores elementos de $A$." é uma invariante do laço externo.

(c) Prove que o *Bubble Sort* faz o que promete.

(d) Qual é o tempo de execução do *Bubble Sort*?

4. Em cada situação a seguir, prove se $f(n) = O(g(n))$ ou $f(n) \neq O(g(n))$, e se $f(n) = \Omega(g(n))$ ou $f(n) \neq \Omega(g(n))$. Comente quando $f(n) = \Theta(g(n))$. Considere que $a$ e $b$ são constantes positivas:

(a) $f(n) = n^2 + 10n + 20$ e $g(n) = n^2$

(b) $f(n) = n^{1/2}$ e $g(n) = n^{2/3}$

(c) $f(n) = \frac{n(n+1)(n+2)}{2}$ e $g(n) = n^2 \log n$

(d) $f(n) = \frac{n}{1000}$ e $g(n) = 50^{100}$

(e) $f(n) = \log_a n$ e $g(n) = \log_b n$ (o que esse resultado significa?)

(f) $f(n) = 100^{n+a}$ e $g(n) = 100^n$

(g) $f(n) = 100^{an}$ e $g(n) = 100^n$

(h) $f(n) = 99^{n+a}$ e $g(n) = 100^n$

(i) $f(n) = n^{1.01}$ e $g(n) = n \log^2 n$

(j) $f(n) = \log \sqrt{n}$ e $g(n) = \log(100n)$

(k) $f(n) = 2^{n+1}$ e $g(n) = 2^n$

(l) $f(n) = 2^{2n}$ e $g(n) = 2^n$

(m) $f(n) = (n + a)^b$ e $g(n) = \Theta(n^b)$ com $b > 0$

(n) $f(n) = n \log n$ e $g(n) = 10n \log 10n$

(o) $f(n) = 10 \log n$ e $g(n) = \log(n^2)$

(p) $f(n) = n!$ e $g(n) = n \log n$ (dica: $n^n$ e $(n/2)^{n/2}$ podem ser úteis)

5. Prove que

(a) $\sum_{i=1}^{n} i^k$ é $\Theta(n^{k+1})$

(b) $\sum_{i=1}^{n} \frac{i}{2^i}$ é $O(1)$

6. Sejam $f(n)$ e $g(n)$ funções crescentes e maiores do que 1 tais que $f(n)$ é $O(g(n))$. Isto é, existem constantes $d$ e $n_0$ tais que $f(n) \leq dg(n)$ sempre que $n \geq n_0$. Note que se tomarmos $c = \max\{1, d\}$, também vale que $f(n) \leq cg(n)$. Para cada item a seguir, decida se o mesmo é verdadeiro ou falso e dê uma prova ou contraexemplo:

(a) se $f(n)$ é $O(g(n))$, então $\log f(n)$ é $O(\log g(n))$

(b) $2^{f(n)}$ é $O(2^{g(n)})$

(c) $f(n)^2$ é $O(g(n)^2)$

(d) $\log \sqrt{n} = O(\log n)$

(e) se $f(n) = O(g(n))$ e $g(n) = O(h(n))$, então $f(n) = O(h(n))$

(f) se $f(n) = O(g(n))$ e $g(n) = \Theta(h(n))$, então $f(n) = \Theta(h(n))$

7. Considere um polinômio $P(n)$ de grau $k$, isto é, $P(n) = \sum_{i=0}^{k} a_i n^i$, onde cada $a_i$ é uma constante e $a_k > 0$. Seja $t$ uma constante. Prove que

   (a) se $t \geq k$, então $P(n)$ é $O(n^t)$.

   (b) se $t \leq k$, então $P(n)$ é $\Omega(n^t)$.

   (c) se $t = k$, então $P(n)$ é $\Theta(n^t)$.

8. Sejam $f(n)$ e $g(n)$ funções assintoticamente não negativas. Usando a definição da notação $\Theta$, prove que $\max\{f(n), g(n)\} = \Theta(f(n) + g(n))$. O que esse resultado significa?

9. Você provou que seu algoritmo tem tempo de execução $\Omega(n^2)$ no pior caso. O que isso diz sobre o tempo de execução do algoritmo sobre qualquer entrada? Você acredita que o tempo de execução no melhor caso é $\Omega(n^3)$. Isso é uma contradição com o resultado anterior?

10. É possível que um algoritmo tenha tempo de execução $\Omega(n^3)$ e $O(n^2 \log n)$ no pior caso? Justifique.

11. Se um algoritmo tem tempo de execução $O(n^2)$ sobre todas as entradas de tamanho $n$, ele pode ter tempo de execução $\Omega(n \log n)$ no pior caso?

12. Se um algoritmo tem tempo de execução $\Theta(n^2)$ sobre todas as entradas de tamanho $n$, ele pode ter tempo de execução $\Omega(n \log n)$ no pior caso?

13. Explique por que a declaração "O tempo de execução do algoritmo A é no mínimo $O(n^2)$" não faz sentido.

14. Suponha que estamos estudando o desempenho de um algoritmo em função do tamanho, $n$, das instâncias de um problema. Considere as seguintes afirmações:

    (a) "o tempo do algoritmo é $O(n^2)$ no pior caso",

    (b) "o tempo do algoritmo é $O(n^2)$ para toda instância do problema",

    (c) "o tempo do algoritmo é $O(n^2)$ no melhor caso" e

    (d) "o tempo do algoritmo é $O(n^2)$ para alguma instância de tamanho $n$".

    Qual a diferença entre as afirmações 14a e 14b? Qual a diferença entre as afirmações 14c e 14d?

15. Nesse exercício, um número inteiro $x$ com $n$ dígitos será representado em um vetor com $n$ posições, uma para cada dígito, de forma invertida. Assim, por exemplo, o número 38549 é representado pelo vetor $A = (9, 4, 5, 8, 3)$.

    Considere o problema de adicionar dois inteiros de $n$ dígitos cada que estão representados em dois vetores $A$ e $B$, de tamanho, portanto, $n$. A soma dos dois inteiros deve ser representada em um vetor $C$ de tamanho $n + 1$.

    Escreva um algoritmo que resolva esse problema e escreva uma expressão para seu tempo de execução no pior e no melhor caso, usando notação assintótica.

16. Seja $A[1..n]$ um vetor ordenado com elementos distintos. Mostre um algoritmo que decide se existe índice $i$, com $1 \leq i \leq n$, tal que $A[i] = i$ em tempo $O(\log n)$.

17. Sejam $X[1..n]$ e $Y[1..n]$ dois vetores, cada um contendo $n$ números ordenados. Dê um algoritmo $O(\log n)$ para encontrar a mediana de todos os $2n$ elementos dos vetores $X$ e $Y$.

18. Escreva um algoritmo que rearranje um vetor $A[ini..fim]$ de inteiros de modo que tenhamos $A[ini..j-1] \leq 0$ e $A[j..fim] > 0$ para algum $j$ em $ini..fim+1$. Faz sentido exigir que $j$ esteja em $ini..fim$? Procure fazer um algoritmo rápido que não use vetor auxiliar.

19. Neste problema, considere que fotos e imagens são representadas por matrizes binárias (matrizes de 0's e 1's). Dada a imagem de um elemento $X$ que está sendo procurado, que é uma matriz binária de tamanho $p \times q$, deseja-se procurá-la em uma foto $F$ de tamanho $m \times n$. Cada posição de uma matrizes é também chamada de pixel.

    (a) Escreva em pseudocódigo uma função de nome PROCURA-SE, especificada de forma a receber parâmetros $X$, $p$, $q$, $F$, $m$ e $n$ como descritos acima. A função deve devolver `Verdadeiro` se existir uma submatriz de $F$ delimitada por $p$ linhas consecutivas e $q$ colunas consecutivas que seja idêntica à matriz $X$, e `Falso` caso contrário. Você não precisa provar a corretude desse pseudocódigo.

    (b) Em função dos parâmetros fornecidos à função PROCURA-SE, calcule o número de comparações de pixels feitas, no pior caso.

    (c) Suponha que haja $N = 10$ milhões de fotos em um certo banco de dados. Adote os valores $p = q = 100$, $m = n = 1000$, e suponha que cada comparação de pixels (elementos das matrizes) gasta ao menos $10^{-13}$ dias (8.64 nanosegundos). Calcule, em dias, o tempo que um programa que chame a sua função PROCURA-SE $N$ vezes levará, no pior caso, para executar todas as comparações de pixels necessárias na procura de uma imagem $X$ nas $N$ fotos desse banco de dados.

20. Dado um vetor $A$ com $n$ elementos ordenados de forma crescente e um valor $k$, escreva um algoritmo que encontre a quantidade de ocorrências de $k$ em $A$ em tempo $O(\log n)$. Por exemplo, se $A = (50, 50, 68, 68, 68, 68, 76, 79, 90)$ e $k = 68$, o algoritmo deve devolver 4. Você pode considerar que $k$ ocorre em $A$ ao menos uma vez.

# 1 Questões retiradas do Enade e Poscomp

**QUESTÃO 12**

Analise o custo computacional dos algoritmos a seguir, que calculam o valor de um polinômio de grau *n*, da forma: $P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$, onde os coeficientes são números de ponto flutuante armazenados no vetor *a[0..n]*, e o valor de *n* é maior que zero. Todos os coeficientes podem assumir qualquer valor, exceto o coeficiente $a_n$ que é diferente de zero.

**Algoritmo 1:**

```
soma = a[0]
Repita para i = 1 até n
   Se  a[i] ≠ 0.0 então
      potência = x
      Repita para j = 2 até i
         potência = potência * x
      Fim repita
      soma = soma + a[i] * potencia
   Fim se
Fim repita
Imprima(soma)
```

**Algoritmo 2:**

```
soma = a[n]
Repita para i = n-1 até 0 passo -1
      soma = soma * x + a[i]
Fim repita
Imprima(soma)
```

Com base nos algoritmos 1 e 2, avalie as asserções a seguir e a relação proposta entre elas.

I. Os algoritmos possuem a mesma complexidade assintótica.

**PORQUE**

II. Para o melhor caso, ambos os algoritmos possuem complexidade *O(n)*.

A respeito dessas asserções, assinale a opção correta.

**A** As asserções I e II são proposições verdadeiras, e a II é uma justificativa correta da I.

**B** As asserções I e II são proposições verdadeiras, mas a II não é uma justificativa correta da I.

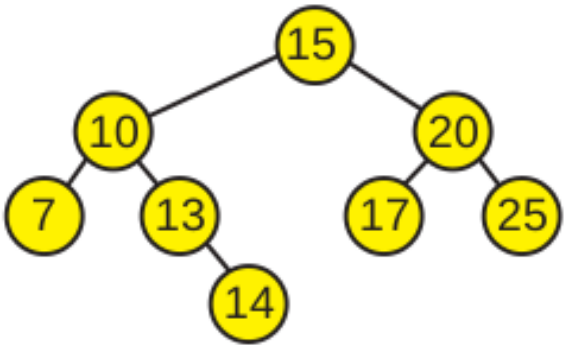
**C** A asserção I é uma proposição verdadeira, e a II é uma proposição falsa.

**D** A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.

**E** As asserções I e II são proposições falsas.

**QUESTÃO 13**

A figura a seguir apresenta uma árvore binária de pesquisa, que mantém a seguinte propriedade fundamental: o valor associado à raiz é sempre menor do que o valor de todos os nós da subárvore à direita e sempre maior do que o valor de todos os nós da subárvore à esquerda.




Em relação à árvore apresentada na figura, avalie as afirmações a seguir.

I. A árvore possui a vantagem de realizar a busca de elementos de forma eficiente, como a busca binária em um vetor.

II. A árvore está desbalanceada, pois a subárvore da esquerda possui um número de nós maior do que a subárvore da direita.

III. Quando a árvore é percorrida utilizando o método de caminhamento pós-ordem, os valores são encontrados em ordem decrescente.

IV. O número de comparações realizadas em função do número n de elementos na árvore em uma busca binária realizada com sucesso é O(log n).

É correto apenas o que se afirma em

**A** I e III.
**B** I e IV.
**C** II e III.
**D** I, II e IV.
**E** II, III e IV.

**QUESTÃO 16**

Uma pilha é uma estrutura de dados que armazena uma coleção de itens de dados relacionados e que garante o seguinte funcionamento: o último elemento a ser inserido é o primeiro a ser removido. É comum na literatura utilizar os nomes *push* e *pop* para as operações de inserção e remoção de um elemento em uma pilha, respectivamente. O seguinte trecho de código em linguagem C define uma estrutura de dados pilha utilizando um vetor de inteiros, bem como algumas funções para sua manipulação.

```
#include <stdlib.h>
#include <stdio.h>
typedef struct {
        int elementos[100];
        int topo;
}pilha;

pilha * cria_pilha() {
        pilha * p =malloc(sizeof(pilha));
        p->topo = -1;
        return pilha;
}

void push(pilha *p, int elemento) {
        if (p->topo >= 99)
                return;
        p->elementos[++p->topo] = elemento;
}
int pop(pilha *p) {
        int a = p->elementos[p->topo];
        p->topo--;
        return a;
}
```

O programa a seguir utiliza uma pilha.

```
int main() {
        pilha * p = cria_pilha();
        push(p, 2);
        push(p, 3);
        push(p, 4);
        pop(p);
        push(p, 2);
        int a = pop(p) + pop(p);
        push(p, a);
        a += pop(p);
        printf("%d", a);
        return 0;
}
```

A esse respeito, avalie as afirmações a seguir.

I. A complexidade computacional de ambas funções `push` e `pop` é O(1).

II. O valor exibido pelo programa seria o mesmo caso a instrução `a += pop(p);` fosse trocada por `a += a;`

III. Em relação ao vazamento de memória (*memory leak*), é opcional chamar a função `free(p)`, pois o vetor usado pela pilha é alocado estaticamente.

É correto o que se afirma em

**A** I, apenas.

**B** III, apenas.

**C** I e II, apenas.

**D** II e III, apenas.

**E** I, II e III.

**QUESTÃO 25 –** A análise de algoritmos que estabelece um limite superior para o tempo de execução de qualquer entrada é denominada análise

A) do melhor caso.
B) do caso médio.
C) do pior caso.
D) da ordem de crescimento.
E) do tamanho da entrada.

**QUESTÃO 22 –** Dado o trecho de código

```
int i, j, c;
c = 1;
for (i =1; i < n; i = i*2){
        for (j =1; j <= n; j++){
                c=c+1;
        }
}
```

Assumindo que a instrução c=c+1 é $O(1)$, a expressão que melhor define a ordem de complexidade desse trecho é:

A) $O(n \log n)$
B) $O(\log n)$
C) $O(n)$
D) $O(n^2)$
E) $O(\sqrt{n})$

**QUESTÃO 25 –** Para medir o custo de execução de um algoritmo, é comum definir uma função de complexidade *f*, em que *f(n)* é a medida de tempo necessário para executar um algoritmo para um problema de tamanho *n*. Considere as afirmações abaixo sobre funções de complexidade:

I. Se *f(n)* é uma medida de quantidade de tempo necessário para executar um algoritmo em um problema de tamanho *n*, então *f* é chamada função de complexidade de tempo.
II. Se *f(n)* é uma medida de quantidade de memória necessária para executar um algoritmo de tamanho *n*, então *f* é chamada função de complexidade de espaço.
III. A complexidade de tempo não representa o tempo diretamente, mas é estimada pelo número de vezes que determinada operação relevante é executada.

Quais estão corretas?

A) Apenas I.
B) Apenas II.
C) Apenas III.
D) Apenas I e II.
E) I, II e III.

**QUESTÃO 22 –** Considere as seguintes funções:

$$f(n) = 2^n$$

$$g(n) = n!$$

$$h(n) = n^{log n}$$

Assinale a alternativa correta a respeito do comportamento assintótico de f(n), g(n) e h(n).

A) $f(n) = O(g(n)); g(n) = O(h(n))$.
B) $f(n) = \Omega(g(n)); g(n) = O(h(n))$.
C) $g(n) = O(f(n)); h(n) = O(f(n))$.
D) $h(n) = O(f(n)); g(n) = \Omega(f(n))$.
E) Nenhuma das anteriores.

**QUESTÃO 25 –** Considere a seguinte função em C:

```
void funcao(int n){
   int i,j;
   for (i=1; i<=n; i++)
      for(j=1; j<log(i); j++)
         printf("%d",i+j)
}
```

A complexidade dessa função é:

A) $\theta(n)$
B) $\theta(n\ log\ n)$
C) $\theta(log\ n)$
D) $\theta(n^2)$
E) $\theta(n^2 log\ n)$