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**, na seguinte ordem:

- descrever em palavras qual é a ideia do seu algoritmo (**obrigatório**);
- escrever um pseudocódigo (**obrigatório**);
- provar ou fornecer um argumento intuitivo para sua corretude (**obrigatório**), a depender do que o exercício pede;
- analisar o tempo de execução de pior caso do seu algoritmo (**obrigatório**).

O não cumprimento dos itens acima implica que o exercício está incorreto e portanto será desconsiderado.
Na dúvida, procure atendimento!

(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:

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

$$f_5(n) = n^2 \log_2 n, \quad f_6(n) = 34n^0, \quad f_7(n) = 18902n^2,$$

$$f_8(n) = 3\frac{n^{127}}{n^{34}}, \quad f_9(n) = 456^{8478}, \quad f_{10}(n) = \frac{n(n+1)}{5}$$

   Escolha duas funções quaisquer e justifique por que você decidiu colocar uma antes da outra na sua lista.

2. 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:

   Função BUBBLESORT(A, n)
       Para $i = n$ até 2, decrementando **faça**
           Para $j = 1$ até $i - 1$, incrementando **faça**
               Se $A[j] > A[j+1]$ **então**
                   troque $A[j]$ com $A[j+1]$

   Responda:

   (a) Prove que a frase $P(x)$ = "Antes da x-ésima iteração, vale que $j = x$ e que $A[j]$ é 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 y-ésima iteração, vale que $i = n - y + 1$, 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*?

3. Para cada um dos Algoritmos 1, 2 e 3, forneça duas expressões, apenas em notação assintótica, de descrição de tempo de execução, sendo uma para o melhor caso e outra para o pior caso. Observe que 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 Θ sempre que possível. Justifique sua resposta com uma única frase.

Algorithm 1 Busca em dois vetores.

Função BUSCAVETORES(A, n, B, m, k)
    Para $i = 1$ até n **faça**
        Se $A[i] == k$ **então**
            Devolve *verdadeiro*
    Para $i = 1$ até m **faça**
        Se $B[i] == k$ **então**
            Devolve *verdadeiro*
    Devolve *falso*

Algorithm 2 Busca por elemento em comum.

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

Algorithm 3 Busca por elementos duplicados no mesmo vetor.

```
Função BuscaDuplicados(A, n)
    Para i = 1 até n faça
        Para j = i + 1 até n faça
            Se A[i] == A[j] então
                Devolve verdadeiro
    Devolve falso
```

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) $f(n) = 5n^4 - 45n^3 + 3n^2 - 21n$ e $g(n) = n^4$

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

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

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

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

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

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

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

   (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 $b > 0$ e inteiro

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

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

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

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

5. Prove que

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

   (b) $\displaystyle\sum_{i=1}^{n} \frac{i}{2^i}$ 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) se $f(n) = O(g(n))$ e $g(n) = O(h(n))$, então $f(n) = O(h(n))$
   (e) 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 Θ, 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, ao mesmo tempo, 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 qualquer entrada 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 qualquer entrada 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 $O(n^2)$",
   (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 o significado de cada afirmação? Qual a diferença entre as afirmações 14a e 14b? Qual a diferença entre as afirmações 14c e 14d?

15. Reescreva o algoritmo da busca binária, alterando-o para que, quando não há elemento com chave k presente no vetor, ele devolva um índice i tal que $A[i-1] < k$, se $i \geq 2$, e $A[i+1] > k$, se $i \leq n-1$. Em outras palavras, o algoritmo sempre devolve algum índice válido i do vetor: se o elemento existe, está armazenado no índice i, e se o elemento não existe, ele estaria armazenado no índice i caso existisse no vetor. Prove que o seu algoritmo está correto por meio de uma invariante de laço.

16. 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. Perceba que o número resultante da soma pode ter $n + 1$ dígitos.

    Faça um algoritmo que recebe um inteiro n, dois vetores A e B que representam dois inteiros com n dígitos cada e que devolve um vetor C que representa a soma dos inteiros recebidos.

17. 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 $O(\log n)$.

18. Escreva um algoritmo que rearranje um vetor $A[ini..fim]$ de inteiros de modo que, ao fim do processo, exista um índice j, com $ini \leq j \leq fim + 1$, tal que todos os elementos de $A[ini..j-1]$ sejam menores ou iguais a 0 e todos os elementos de $A[j..fim]$ sejam maiores do que 0. Faz sentido exigir que j esteja no conjunto $[ini..fim]$ ao invés de $[ini..fim + 1]$? Procure fazer um algoritmo rápido que não use vetor auxiliar. Atenção: escreva seu próprio algoritmo para resolver diretamente esse problema, isto é, não use, por exemplo, algoritmos de ordenação.

19. 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.

    Para mostrar a corretude deste algoritmo, você pode usar algum argumento simples (não há necessidade de formalizar uma invariante de laço).

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.

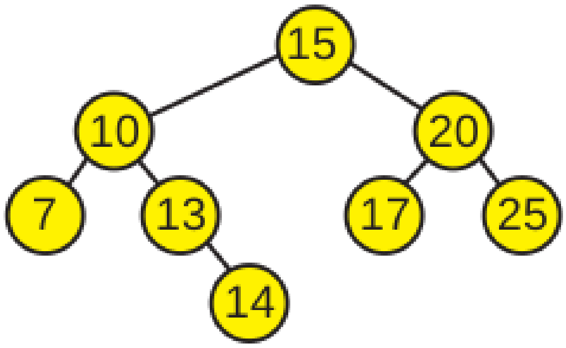
C A asserção I é uma proposição verdadeira, e a II é uma proposição falsa.

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:

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

$$g(n) = n!$$

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

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) $\theta(n^2 log\ n)$