Pontifícia Universidade Católica do Paraná
Escola Politécnica / Bacharelado em Ciência da Computação
Disciplina de Complexidade de Algoritmos / Prof. Edson Emilio Scalabrin
Data: 04/03/2021

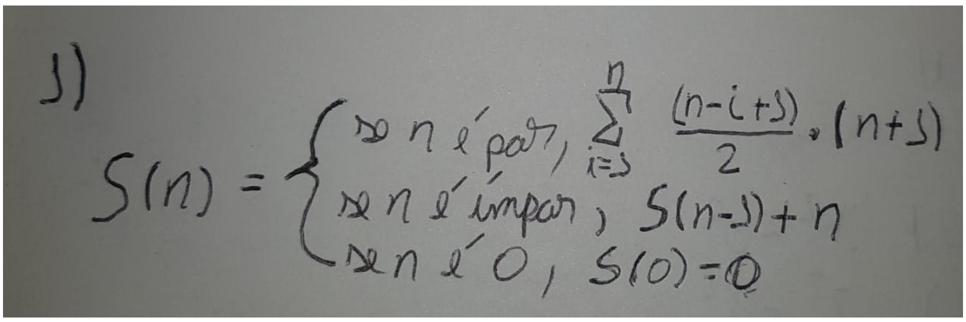
AVALIAÇÃO DIAGNÓSTICA

Estudante: Gustavo Hammerschmidt.

Regras: Como se trata de uma avaliação diagnóstica, ela não gera nenhuma pontuação efetiva para a disciplina. Entretanto, ela é importante para indicar o ponto de início da disciplina e também servir de referência no final da disciplina para sabermos o avanço realizado. Sendo assim, convido cada estudante tentar fazer individualmente e sem consultar nenhum material (livro, caderno, internet, colega) para responder as perguntas desta avaliação. Deve-se ter em mente que os assuntos constante neste avaliação serão abordados durante a disciplina independente do resultado que cada aluno(a) alcance. O que poderá mudar é o tempo dedicado em cada tema, que poderá ser maior ou menor.

A avaliação pode ser preenchida a mão em folha de papel, fotografada, as fotos inseridas em um arquivo Word, gerar um PDF e postá-lo no ***Blackboard***. Pode-se também fazer cópia em scanner da resolução e postá-lo no ***Blackboard*** em apenas um arquivo PDF. Outra forma, de preencher a avaliação é usando um editor de texto e postar o PDF correspondente no ***Blackboard***.

ITEM 01:
Escreva a fórmula fechada para a soma dos números inteiros positivos, onde S = 1 + 2 + 3 + ... + n.



ITEM 02:
Prove por indução matemática que a fórmula para a soma dos números inteiros positivos do ITEM 01 é verdadeira para qualquer valor de n.

2)

$\vdash S(n) \rightarrow True$

$\cancel{\vdash S(n+1)}$ $^{*}\vdash S(n+1) \rightarrow True$

$n = 1 \mapsto S(0) + 1 \rightarrow 1$ ✓

$n = 2 \vdash \frac{(2-1+1)}{2} \cdot (2+1) \vdash 1 \cdot 3 \vdash 3$ ✓

$n = n+1 \vdash \frac{(n+1-1+1)}{2} \cdot (n+1+1) \vdash \frac{(n+1)\cdot(n+2)}{2}$ ✓

$\sum n-1 = n \rightarrow \frac{(n)\cdot(n+1)}{2}$ c.q.d.

Teorema de Gauss.

ITEM 03:
Escreva duas funções recursivas:
a) uma função para o cálculo do fatorial; e

```
function factorial(n)

    if n == 0

        return 1;

    else

        return n * factorial(n-1);

    end if

end
```

b) uma função para a busca binária de uma chave sobre um arranjo ordenado de chaves.

```
function bin_search(int[] chaves, int chave, int begin, int end)
    int begin = begin, end = end; // begin <- 0 | end <- chaves.length()
    int mid = begin + (end-begin) / 2;
    if chaves[mid] == chave
        return mid;
    else if chaves[mid] > chave
        return bin_search(chaves, chave, 0, mid);
    else
        return bin_search(chaves, chave, mid, end);
    end if
end
```

ITEM 04: Para a contagem de tempo ou de instruções, escreva as equações de recorrência para as seguintes funções:

a) cálculo do fatorial; e

```
Obs.: Entendo por equação de recorrência uma instrução que possui uma cadeia de eventos ou uma função com um
stack de chamadas.

Sendo assim:

    a) cálculo do fatorial:

        f(n) [suponha n = 3;
                p, uma pilha de chamadas;
                r, return]

        p[0]    f(3){

        p[1]        r 3 * f(2){

        p[2]            r 2 * f(1){

        p[3]                r 1 * f(0){

                                return 1;

        p[3]                }

        p[2]            }

        p[1]        }

        p[0]    }

        Quatro tempos de stack e quatro de unstack.
        Quatro multiplicações.
        A partir de f(3), quatro instruções foram executadas.

        Tempo de execução: T_stack(n+1) + T_unstack(n+1)
                                        + T_op_mult(n+1)
                                        + T_instrução_de_retorno(n+1)

        O(n!).

    =================
```

b) busca binária.

```
================

b) busca binária:

    arr = [ 0, 1, 2, 3, 4, 5, 6, 7, 8]

    bin_search(arr, 3, 0, arr.length()){                        |
                                                                |
        int begin = 0, end = 9, mid = 4;                        |      <- A
                                                                |
        if mid == 3 {}; (FALSE)                                 |      <- B
                                                                |
        else if mid > 3 {}; (TRUE)                              |      <- B
        |                                                       |
        |-> return bin_search(arr, 3, 0, 4){                    |      <- C
                                                                |
                int begin = 0, end = 4, mid = 2;                |      <- A
                                                                |
                if mid == 2 {}; (FALSE)                         |      <- B
                                                                |
                else if mid > 3 {}; (FALSE)                     |      <- B
                |                                               |
                |-> return bin_search(arr, 3, 2, 4){            |      <- C
                                                                |
                        int begin = 2, end = 4, mid = 3;        |      <- A
                                                                |
                        if mid == 3 {}; (TRUE)                  |      <- B
                        |                                       |
                        |-> return mid;                         |      <- C
                                                                |
                }                                               |
                                                                |
            }                                                   |
                                                                |
    }                                                           |


        | A -> instrução de armazenamento;
        | B -> instrução de decisão;
        | C -> Chamada de retorno.

    O( lg(n) ). 9 valores -> máximo de 3 chamadas para encontrar todos os valores.
```

ITEM 05:

Resolva a equação de recorrência para busca binária do ITEM 3b por substituição ou árvore de resolução.

```
item 5

        arr = [ 0, 1, 2, 3, 4, 5, 6, 7, 8]

        valor = 7

                        bin_search
                            |
                            |
                           arr
                           /|\
                          / | \
                         0  4  8
                        /   |   \
                      4>7   |   4<7
                      /    4==7   \
                     *      |     arr
                            *     /|\
                                 4 6 8
                                /  |  \
                              4>7  |   8>7
                              /  6==7   \
                             *     |     arr
                                   *     /|\
                                        6 7 8
                                       /  |  \
                                    6>7  |   8>7
                                    /  7==7   \
                                   *      |     *
                                         end

            left-comparison-value é o index do array.
            right-comparison-value é o valor.
            mid-comparison é executada primeiro.
            begin- e end-comparison são testadas em if.
```

ITEM 06:

Revolva os somatórios a seguir indicando para cada um deles a sua fórmula fechada:

a)	$\sum_{i=0}^{n}(2i-1)$ |- $(0-1)+(2-1)+(6-1)+\ldots\| - 2*(0+1+2+3) - 1*n\| - 2*(teorema\ de\ gauss) - n$ $$\sum_{i=0}^{n}(2i-1) = 2*\frac{n*(n+1)}{2} - n - 1 = n*(n+1) - n - 1$$ $$= n*(n+1) - 1*(n+1) = (n+1)*(n-1)$$ - 1 surge da primeira interação, quando n = 0, somatório é igual a -1.
b)	$\sum_{i=0}^{n-1}[(i+1)^2 - i^2] = (1^2-0^2)+(2^2-1^2)+(3^2-2^2)+\cdots+((n-2)^2-(n-3)^2)+((n-1)^2-(n-2)^2)+(n^2-(n-1)^2) = n^2$
c)	$\sum_{i=1}^{\infty}\left(\frac{5}{i^2}-\frac{5}{(i+1)^2}\right) = \frac{5}{1^2} - \cancel{\frac{5}{2^2}} + \cancel{\frac{5}{2^2}} - \cancel{\frac{5}{3^2}} + \ . \ . \ .$ $\therefore \lim_{k\to\infty}\left(\sum_{i=1}^{k}\frac{5}{i^2} - \frac{5}{(i+1)^2}\right) = \frac{5}{i^2} = 5$ $c.q.d.$

ITEM 07: Dado o programa **P1** abaixo escrito em Python:

- **a)** indique (para cada linha) ao lado de cada constante Ci = 0, 1, 2, 3, ..., 12 o número de vezes que cada instrução é executada.
- **b)** A partir do que foi feito em item (a), expresse o custo total na forma de um somatório.
- **c)** Reescreva o somatório do item (b), assumindo que o custo de cada instrução/comando é 1.
- **d)** Revolva o somatório reescrito no item (c).

Item a)

```python
def P1( A ):                                  # custo
    # N é o tamanho do problema               # C0 . 0 = 0
    N = len( A )                              # C1
    for j in range(1, N, 1):                  # C2
        chave = A[ j ]                        # C3
        i = j - 1                             # C4
        while (i > -1) and (A[ i ] > chave):  # C5
            A[ i + 1 ] = A[ i ]               # C6
            i = i - 1                         # C7
        A[ i + 1] = chave                     # C8
        p = 0                                 # C9
        while (p < N):                        # C10
            print(A[ p ])                     # C11
            p = p + 1                         # C12
```

C1 = 1
C2 = 1
C3 = S1
C4 = S1
C5 = S1
C6 = S1 * S2
C7 = S1 * S2
C8 = S1
C9 = S1
C10 = S1
C11 = S1 * N
C12 = S1 * N

S1 = (N-1)/1 = N-1
S2 = no pior caso, o total de loops será N^2. O algoritmo segue a lógica de um bubblesort, ou seja, em cada interação, pode executar a mesma instrução de 1 a N vezes.

Item b)

Custo(P1(A))= $C1 + C2 + \sum_{j=1}^{n} (C3 + C4 + C5 + \sum_{i=j-1}^{k=-1}(C6 + C7) + C8 + C9 + C10 + \sum_{p=0}^{n}(C11 + C12)$), n = len(A)

Item c)

Custo(P1(A)) = $2 + \sum_{j=1}^{n} (6 + \sum_{i=j-1}^{k=-1}(2 * i) + 2 * n$), n = len(A)

Item d)

Custo(P1(A)) = $2 + \sum_{j=1}^{n} (6 + \sum_{i=j-1}^{k=-1}(2 * i) + 2 * n$), n = len(A)

$= 2 + (n-1) * (6 + 2 * n) + \sum_{j=1}^{n} (\sum_{i=j-1}^{k=-1}(2 * i))$, n = len(A)

Instruções de comando $= 2 + (n-1) * (6 + 2 * n) = -4 + 4n + 2n^2$, n = len(A)

| Considere que o tempo de execução de instruções de comando tende a 0 ms.

$$\text{Custo}(\text{P1}(\text{A})) = \sum_{j=1}^{n} (\sum_{i = j-1}^{k=-1}(2 * i)), \text{n} = \text{len}(\text{A})$$

Note que o primeiro somatório itera de j=1 até n, e o segundo somatório nos limites de 0 até n-1, sendo assim, temos que o custo de P1 – em termos de A – é de n valores para o primeiro somatório vezes n valores (no pior cenário) para o segundo somatório vezes duas instruções. Portanto, o custo é equivalente a n * n * 2, desconsidere a magnitude das duas instruções, e teremos, logo, que o custo é de n^2 no pior cenário em termos de A para a função P1.

Custo(P1(A)) = [*Worst-Case*] O(n^2), [*Best-Case*] O(n), n = len(A)

c.q.d.

ITEM 08: Qual é a importância da passagem de parâmetros por referência com vista a complexidade de um programa? Ilustre a situação usando textualmente e/ou graficamente.

A importância se dá quando um programa utiliza um conjunto de dados grande em memória ram, se não passados por referência, haverá uma duplicação desses elementos em memória ram – aumentando, subsequentemente, o uso de memória pelo programa e o tempo de leitura e escrita dos dados: o que pode intensificar uma piora no tempo de execução -; não somente isto, mas, se o programa utilizar recursividade, então, haverá possivelmente um uso de memória ram não só para armazenar o stack de chamadas de função mas, também, a escrita em memória dos dados de modo cascata: a cada chamada uma nova escrita dos mesmos dados é feita em memória – o que, logicamente, impacta significativamente na performance de um programa – logo, o impacto espacial será alto.