MATA52 - Exercícios da Semana 01

- Grupo: Paládio
- Autores:
    - Bruno de Lucas Barbosa (Responsável)
        - Convidei os membros para uma primeira tentativa de realizar a lista na quarta-feira(09/03) e todos compareceram. Conseguimos pensar bastante com ajuda de todos e resolvemos 3 das 4 questões com uma certa dificuldade e marcamos outro encontro no dia 11/03 onde todos compareceram para tentarmos realizar a questão que faltou
    - Elis Marcela de Souza Alcantara
        - Ajudou ativamente em TODAS as questões buscando formas de resolver e conteúdos para que todos os membros pudessem acompanhar e contribuir com a atividade e compareceu em todos os 2 encontros.
    - Lucas dos Santos Lima
        - Ajudou ativamente em TODAS as questões, trazendo um conhecimento matemático exemplar junto de Elis e sintático para que pudessemos realizar a atividade.
    - Monique Silva
        - Esteve presente em todos o 2 encontros, além de ajudar ativamente em TODAS as questões buscando formas de resolver e buscando conteúdos relevantes para todos os membros acompanharem.

Instruções (não apagar)

1. Após criar este notebook, altere o nome do notebook/documentação incluindo o nome do seu grupo. Por exemplo, se você é do grupo Ouro, altere o nome do notebook para "MATA53-Semana01-Ouro.ipynb"
2. Compartilhe este documento com todos os membros do grupo (para participarem da elaboração deste documento). É importante que o notebook utilizado seja o mesmo compartilhado para que os registros de participação e colaboração fiquem salvos no histórico.
3. Incluia o nome completo dos autores na lista de autores que auxiliaram na elaboração deste notebook. Destaque o responsável como sendo o(a) primeiro(a) autor(a). Relatar brevemente

a contribuição de cada membro do grupo.

4. Utilize os recursos de blocos de texto e de código para adicionar as suas respostas, sem alterar os blocos de texto e código existente. Não economize, esses blocos são de graça.
5. Ao concluir a atividade, compartilhe o notebook com januario.ufba@gmail.com (dando permisão para edição) e deixando o aviso de notificação marcado, para que eu receba o seu e-mail. Identificar o nome do grupo na mensagem de compartilhamento.

Exercícios

1. Ordene as seguintes funções por ordem de crescimento:

- f(n) = n^2
- f(n) = $n \log n$
- f(n) = n
- f(n) = $\log n$
- f(n) = $\sqrt{n}$

  Utilize o código abaixo para facilitar a sua análise

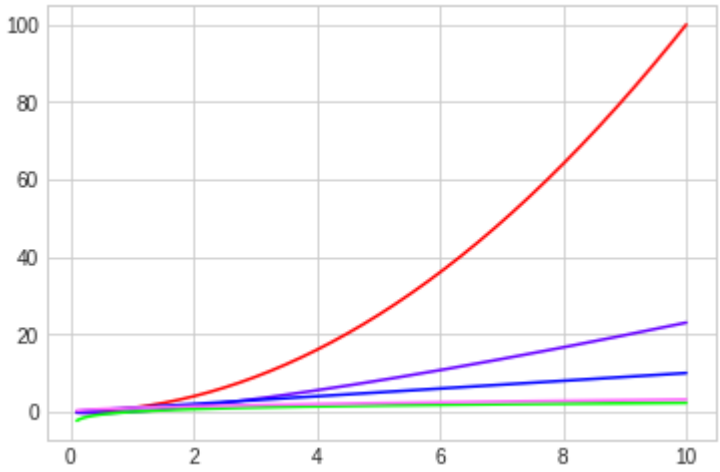
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np

n = np.linspace(0.1, 10, 1000)
# plt.plot(n, 2*n, linestyle='solid')
# plt.plot(n, n*np.log(np.power(n,2)), linestyle='dashed')

plt.plot(n, np.power(n, 2), linestyle='solid', color = '#ff0000') # red
plt.plot(n, n*np.log(n), linestyle='solid', color = '#6600ff') # purple
plt.plot(n, n, linestyle='solid', color = '#0000ff') # blue
plt.plot(n, np.log(n), linestyle='solid', color = '#00ff00') # green
plt.plot(n, np.sqrt(n), linestyle='solid', color = '#ff66ff') # pink
```

1. Resposta:

De acordo com a análise feita com base no trecho de código:

```
plt.plot(n, np.power(n, 2), linestyle='solid', color = '#ff0000') # red
plt.plot(n, n*np.log(n), linestyle='solid', color = '#6600ff') # purple
plt.plot(n, n, linestyle='solid', color = '#0000ff') # blue
plt.plot(n, np.log(n), linestyle='solid', color = '#00ff00') # green
plt.plot(n, np.sqrt(n), linestyle='solid', color = '#ff66ff') # pink
```

Podemos obter o gráfico que auxiliou a identificação de uma superioridade muito grande da função * f(n) = $n \log n$ em comparação com a * f(n) = n^2



Identificamos a seguinte ordem de cresimento

1. ◦ f(n) = $\log n$
2. ◦ f(n) = $\sqrt{n}$
3. ◦ f(n) = n
4. ◦ f(n) = $n \log n$
5. ◦ f(n) = n^2

2. Suponha que nós estamos comparando duas implementações de algoritmos de ordenação. O algoritmo A executa em $2n^2$ passos e o algoritmo B executa em 64 $n \log n$ passos. Para quais intervalos de n você recomenda o uso do algoritmo A ao invés do algoritmo B. Apresente uma solução gráfica e uma prova matemática.

2. Resposta:

Sabemos que a constante do algoritmo A é 2 e a constante do algoritmo B é 64, portanto temos que analisar um caso em que n é grande o suficiente para que o algoritmo A seja mais proveitoso que o algoritmo B.

Usando n = 2

$2n^2$

$2 * (2)^2$

8

$64 * n \log n$

$64 * 2 \log 2$

38.5

A partir do gráfico é possível notar que o comportamento do algoritmo A até $n \leq 200$ é mais proveitoso que o algoritmo B, porém, com mais entradas o algoritmo B se torna mais eficiente.

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np

n = np.linspace(0.1, 1000, 1000)

plt.plot(n, 2 * np.power(n, 2), linestyle='solid', color = '#ff0000') # red
plt.plot(n, 64 * n * np.log(n), linestyle='dashed', color = '#0000ff') # blue
```

3. Escreva um algorimo em Python que implementa o método de ordenação por inserção em ordem decrescente. Considere como entrada um vetor de tamanho genérico.

3. Resposta:

O número de comparações que ocorrem durante a ordenação por inserção depende de como a lista está inicialmente ordenada. Se a lista estiver em ordem, o número de comparações será n-1. No pior caso, o algoritmo de ordenação por inserção possui complexidade quadrática (O(n2)). Portanto, para o pior caso, a ordenação por inserção é tão ruim quanto a ordenação bolha.

O código abaixo ilustra e detalha como ocorre a ordenação por inserção em ordem decrescente de uma array com 10 elementos.

```
array = [1,8,2,6,7,9,0,4,5,3];
# inserir a sua resposta aqui...;
def decInsertionSort(array):
    # Primeiro partimos de um laço exterior que ;
    # parte do segundo número no array, visto que;
    # se houver somente 1, o array está ordenado.;
    for step in range(1, len(array)):
        # Utilizamos esse número como chave/key;
        key = array[step];
        # Temos um número j que sempre terá índices ;
        # anteriores ao do valor key ;
        j = step - 1;
        # Garantimos nunca sair dos limites de índice do array;
        # e que só percorremos números anteriores à chave atual.;
        while j >= 0 and key > array[j]:
            # Ao encontrar um valor maior que a chave, o ;
            # movemos para a posição seguinte ;
            # e assim sucessivamente com números anteriores;
            array[j + 1] = array[j];
            j = j - 1;
        # Por último enviamos o número chave para a posição de ;
        # menor número na lista.;
        array[j + 1] = key;
    return array;

print(decInsertionSort(array));
```

Muito embora o número de comparações possa ser razoalvelmente baixo, para certos conjuntos de dados, como a que foi utilizada na questão, a array precisa ser deslocada cada vez que um elemento é colocado na sua posição correta. Como resultado, o número de movimentações pode ser significativo.

4. Expresse a função $\frac{n^3}{100} + 10n^2 - n + 3$ em notação assintótica. Apresente o passo a passo da sua simplificação.

4. Resposta:

$\frac{n^3}{100} + 10n^2 - n + 3$

Regra do produto: Se o Big O for o produto de vários termos, podemos descartar os termos constantes.

$n^3 * \frac{1}{100} + 10 * (n^2) - n + 3$

$n^3 + n^2 - n$

Regra da soma: Se o Big O for a soma de vários termos, mantemos apenas o maior termo e descartamos o resto.

---

n^3

Logo, chegamos ao **resultado**: $O(n^3)$

Provando o resultado:

$\frac{n^3}{100} + 10n^2 - n + 3$ = $O(n^3)$

Se eu escolher c = 100 quando n > 1:

$\frac{n^3}{100} + 10n^2 - n + 3 \leq 100 * n^3$, utilizando n = 5

$\frac{5^3}{100} + (10 * 5)^2 - 5 + 3 \leq 100 * 5^3$

$2499.25 \leq 12500$

Conclui-se que, existe uma constante c tal que para todo n suficientemente grande $f(n) \leq c * g(n)$.