INSTITUTO FEDERAL Ceará Campus Cedro	**INSTITUTO FEDERAL DO CEARÁ** **Campus Cedro**
	Disciplina: Construção e Análise de Algoritmos
	Professor(a): Prof. Me. Antonio Denilsno de Souza Oliveira
	Discente: **Nota:** 2.5 (Bonus 0.8 a 1.0)
	Curso: Sistemas de informação **Semestre:** S04
	Lista 2: Ordenação, Construção e Análise de Algoritmos

1. Elabore um algoritmo $O(n)$ de decomposição de um vetor S em três subvetores. Esse algoritmo recebe como entrada, além do vetor S, um valor piv pertencente a S, e os índices p e r, $1 \leq p \leq r$. O algoritmo deve rearrumar os elementos em $S[p \ldots r]$ e retornar dois índices q_1 e q_2 satisfazendo as seguintes propriedades:

   (a) se $p \leq k \leq q_1$, então $S[k] < piv$;

   (b) se $q_1 < k \leq q_2$, então $S[k] = piv$;

   (c) se $q_2 < k \leq r$, então $S[k] > piv$.

2. Faça um algoritmo de divisão e conquista para multiplicar duas matrizes quadradas (ou seja, o número de linhas é igual ao número de colunas), dividindo cada matriz em 9 submatrizes quadradas. Calcule a complexidade de tempo em notação assintótica.

3. Dado um um vetor de números inteiros e um inteiro X, retorne os índices dos dois números de forma que somados é igual a X. Você pode assumir que cada entrada teria exatamente uma solução e não pode usar o mesmo elemento duas vezes. Você pode retornar a resposta em qualquer ordem.
   Solução simples: Complexidade $O(n^2)$
   Solução melhorada: Complexidade abaixo de $O(n^2)$ (bonus 0.1)
   Solução melhor caso: Complexidade $O(n)$ (bonus 0.2)

   - Example 1:
     Entrada: nums = [2,7,11,15], $X = 9$
     Saida: [0,1]
     Explicação: por causa que nums[0] + nums[1] == 9,
     retornamos [0, 1].
   - Example 2:
     Entrada: nums = [3,2,4], $X = 6$
     Saida: [1,2]

4. Dado um array nums de tamanho n, retorne o elemento majoritário. O elemento majoritário é aquele que aparece mais de [n/2] vezes. Você pode assumir que o elemento majoritário sempre existe no array.

   Solução melhor caso: Tempo linear ($O(n)$) e $O(1)$ em espaço.

5. Analise os algoritmos abaixo de maneira mais justa possível. Definir o custo dos laços em termos de n.

   1-

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

2-

```
        sum = 0;
        for (int i=1; i<n;i*=2){
            for (int j=1; j <=n; j++){
                sum++;
            }
        }
```

3-

```
        sum = 0;
        for (int i=0; i<n;i*=2){
            for (int j=1; j <=n; j+=i){
                sum++;
            }
        }
```

6. Uma subsequência é palíndroma se ela é igual lendo da direita para esquerda ou lendo da esquerda para direita. Por exemplo, a sequência $(ACGTGTCAAAATCG)$ possui muitas subsequências palíndromas, como $(ACGCA)$ e $(AGTGA)$. Mas a subsequência (ACT) não é palíndroma. Escreva um algoritmo $O(n^2)$ que recebe uma sequência $S[1 \ldots n]$ e retorna a subsequência palíndroma de tamanho máximo.

7. Dado um array inteiro nums classificado em ordem não decrescente, retorne um array dos quadrados de cada número classificado em ordem não decrescente.

   Example 1:

   Entrada: nums $= [-4, -1, 0, 3, 10]$

   Saida: [0,1,9,16,100]

   Explicação: Após a quadratura, a matriz se torna [16,1,0,9,100].
   Após a classificação, torna-se [0,1,9,16,100].

   - O quadrado de cada elemento e classificar a nova matriz é muito trivial. Solução simples: $O(n \log n)$
   - Você poderia encontrar uma solução $O(n)$ usando uma abordagem diferente? Bonus 0.2

8. (Algoritmos não-recursivos) Determine a função de complexidade (no pior e melhor caso e no caso médio), das funções implementadas em Python, apresentadas abaixo, fazendo as considerações pertinentes.

```
#primeiro
def bubble_sort(A, n):
    for j in range(n):
        for i in range(n - 1):
            if A[i] > A[i + 1]:
                aux = A[i]
                A[i] = A[i + 1]
                A[i + 1] = aux
```

```python
#segundo
def bubble_sort2(A, n):
    troca = True
    while troca:
        troca = False
        for i in range(n - 1):
            if A[i] > A[i + 1]:
                aux = A[i]
                A[i] = A[i + 1]
                A[i + 1] = aux
                troca = True
```

```python
def AlgumaCoisa(n):
    x = 0
    for i in range(1, n):
        for j in range(i + 1, n + 1):
            for k in range(1, j + 1):
                x = x + 1
```

```python
def AlgumaCoisa2(n):
    x = 0
    for i in range(1, n + 1):
        for j in range(i + 1, n):
            for k in range(1, j + 1):
                x = x + 1
```

9. (Algoritmos recursivos) Determine a função de complexidade, das funções recursivas apresentadas abaixo, fazendo as considerações que considerar pertinente.

```python
# Primeira
def Pesquisa1(A, n):
    if n > 1:
        InspecioneNElementos = n * n * n  # custo n^3
        Pesquisa1(A, n // 3)
# Segunda
def Pesquisa2(A, n):
    if n <= 1:
        return
    else:
        # obtenha o maior elemento entre os elementos
        # de alguma forma isso permite descartar 2/5 dos elementos e fazer uma chamada recursiva no resto
        Pesquisa2(A, 3 * n // 5)
# Terceira
def Pesquisa3(A, n):
    if n <= 1:
        return
    else:
        # ordena os elementos
        # de alguma forma isso permite descartar 1/3 dos elementos e fazer uma chamada recursiva no resto
        Pesquisa3(A, 2 * n // 3)
#Magica!!
class Item:
    def __init__(self, Chave):
        self.Chave = Chave

def Enigma2(A, m, n, i, j):
    x = A[(i + j) // 2]
    while True:
        while x.Chave > A[i].Chave:
```

```
            i += 1
        while x.Chave < A[j].Chave:
            j -= 1
        if i <= j:
            A[i], A[j] = A[j], A[i]
            i += 1
            j -= 1
        if i > j:
            break

def Enigma1(A, m, n):
    i, j = 0, 0
    Enigma2(A, m, n, i, j)
    if m < j:
        Enigma1(A, m, j)
    if i < n:
        Enigma1(A, i, n)
```

10. Dado um array ordenado de inteiros distintos e um valor alvo, retorne o índice se o alvo for encontrado. Caso contrário, retorne o índice onde estaria se fosse inserido na ordem.

    Você deve escrever um algoritmo com complexidade de tempo de execução $O(logn)$.

```
#Exemplo 1:
Entrada: nums = [1,3,5,6], alvo = 5
Saida: 2
#Exemplo 2:
Entrada: nums = [1,3,5,6], alvo = 2
Saida: 1
```

11. (Bonus 0.6) Você recebe dois arrays de inteiros nums1 e nums2, classificadas em **ordem não decrescente**, e dois inteiros m e n, representando o número de elementos em nums1 e nums2 respectivamente.

    Mesclar nums1 e nums2 em uma única matriz classificada em **ordem não decrescente**.

    O array classificado final não deve ser retornado pela função, mas sim armazenado dentro do array $nums1$. Para acomodar isso, $nums1$ tem um comprimento de $m+n$, onde os primeiros m elementos denotam os elementos que devem ser mesclados e os últimos n elementos são definidos como 0 e devem ser ignorados. $nums2$ tem um comprimento de n.

    Exemplo 1:

    Entrada:
    nums1 = [1,2,3,0,0,0], $m = 3$,
    nums2 = [2,5,6], $n = 3$
    Saída: [1,2,2,3,5,6]

    Explicação: Os arrays que estamos mesclando são [1,2,3] e [2,5,6]. O resultado da mesclagem é [1,2,2,3,5,6] com os elementos sublinhados vindos de nums1.

    * Caso simples: $O((m+n)log(m+n))$
    * Você consegue criar um algoritmo que execute em tempo $O(m+n)$?