Atividade 1 - Projeto e Analise de Algoritmos 2024.2

Aluno: Davi de Paula Coelho Raia dos Santos
Professores: Leonardo Sampaio, Junior Ferro
Entrega: 10/01/2025
[Link do Github com os códigos usados]

Início Atividade 1

Questão 1)

Item a)

$f(n) = 3n^2 + 2$
$g(n) = n^2$
$Prove:$
$f(n) = O(g(n))$
$Resposta:$ Verdadeiro
$f(n) \leq c * g(n) \; \forall n \geq n_0$
$f(n) = 3n^2 + 2 \leq 3n^2 + 2n^2 = 5n^2$ se $n \geq 1$
Constantes: $c = 5$
$n_0 = 1$

Item b)

$f(n) = 2^n$
$g(n) = 2^{n+1}$
$Prove:$
$f(n) = \Omega(g(n))$
$Resposta:$ Verdadeiro
$f(n) \geq c * g(n) \; \forall n \geq n_0$
$f(n) = 2^n \geq (1/2) * 2^{n-1} = 2^n$ se $n \geq 1$
Constantes:
$c = (1/2)$
$n_0 = 1$

Item c)

$f(n) = 3n^3 + n^2$
$g(n) = n^3$
$Prove:$
$f(n) = \Theta(g(n))$
$Resposta:$ Verdadeiro

$c_1 * g(n) \leq f(n) \leq c_2 * g(n) \, \forall n \geq n_0$
$1 * n^3 \leq 3n^3 + n^2 \leq 4 * n^3$ se $n \geq 1$
Constantes:
$c_1 = 1$
$c_2 = 4$
$n_0 = 1$

Questão 3)

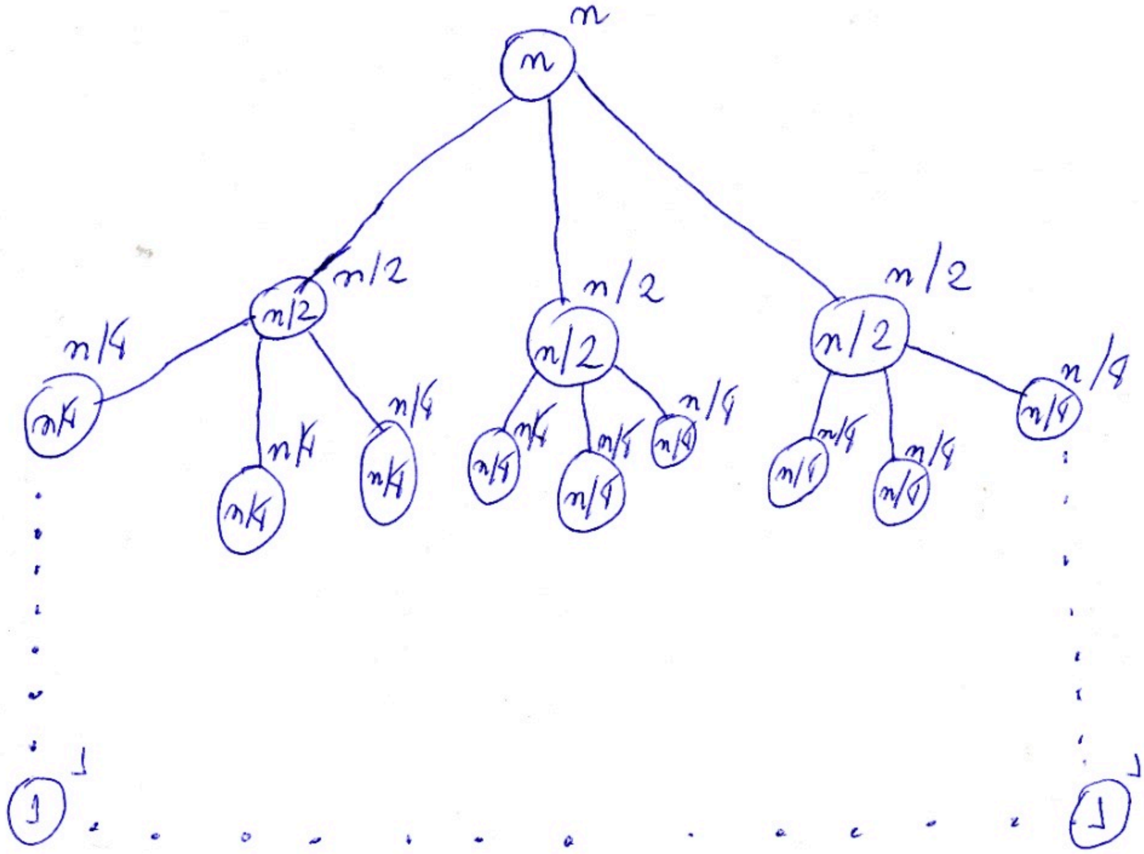
```
RemovePontos(Pontos) -> (Passos, Formas):
  index = 0
  maxIndex = len(Pontos)
  contaPassos = 0
  contaInicialização = 0

  enquanto index < maxIndex - 2:
    se eColinear(Pontos[index], Pontos[index+1], Pontos[index+2]):
      removePontos(Pontos[index], Pontos[index+1], Pontos[index+2])
      contaPassos = contaPassos + 1

    contaInicialização = contaInicialização + 1
    index = index + 2

  retorna contaPassos, contaInicialização
```

Fazendo uma analise, nesse caso a complexidade não é uma preocupação.

Questão 4)

Item a)

$$T(n) = 3T\left(\frac{n}{2}\right) + n$$



Nível	Tamanho	Num Nós	Tempo Por nó
0	n	1	n

Nível	Tamanho	Num Nós	Tempo Por nó
1	$n/2$	3	$n/2$
2	$n/4$	9	$n/4$
3	$n/8$	27	$n/8$
i	$n/2^i$	3^i	$n/2^i$
$log_2 n$	1	$n^{log_2 3}$	1

Após verificado a tabela e árvore podemos chegar no custo total:

$$T(n) = \sum_{i=0}^{log_2 n} ((n/2^i) * 3^i)$$
$$T(n) = n * \sum_{i=0}^{log_2 n} (3/2)^i$$
$$T(n) = n * \sum_{i=0}^{log_2 n} (3/2)^i = n * \frac{((\frac{3}{2})^{log_2 n+1} - 1)}{(3/2) - 1}$$
$$T(n) = n * \frac{((\frac{3}{2})^{log_2 n} * \frac{3}{2} - 1)}{(1/2)}$$
$$T(n) = n * 2 * ((\frac{3}{2})^{log_2 n} * \frac{3}{2} - 1)$$
$$T(n) = n * ((n)^{log_2 \frac{3}{2}} * 3 - 2)$$
$$T(n) = 3n^{1+log_2 \frac{3}{2}} - 2n$$
$$T(n) = O(n^{1+log_2 \frac{3}{2}})$$
$$T(n) = O(n^{log_2 3})$$

Demonstração:

$$T(n) \leq c * n^{log_2 3}$$

Fazendo n = (n / 2)

$$T(n/2) \leq c * (n/2)^{log_2 3}$$

Substituindo a equação original

$$T(n) \leq 3 * c * (n/2)^{log_2 3} + n$$
$$T(n) \leq 3 * c * (n^{log_2 3}/2^{log_2 3}) + n$$
$$T(n) \leq 3 * c * (n^{log_2 3}/3^{log_2 2}) + n$$
$$T(n) \leq 3 * c * (n^{log_2 3}/3) + n$$
$$T(n) \leq c * (n^{log_2 3}) + n$$

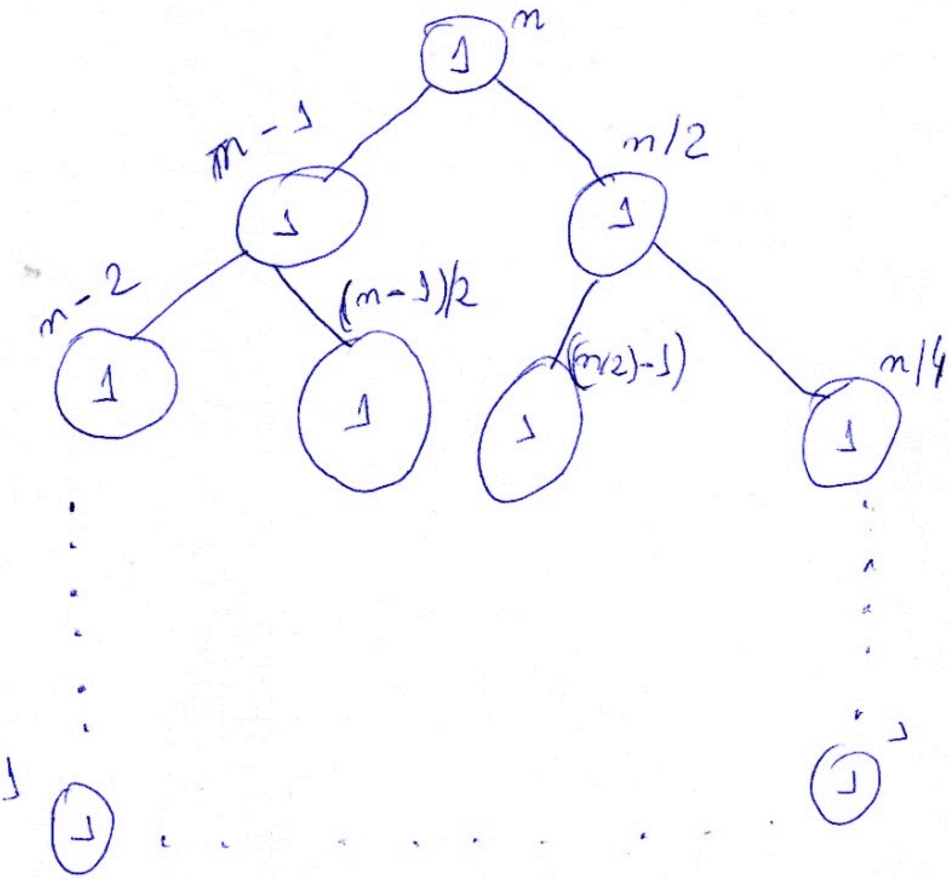
como $(n^{log_2 3}) \geq 1$ o termo $(n^{log_2 3})$ é o dominante

Ou seja, é valido:

$$T(n) = O(g(n))$$

Item b)

$$T(n) = T(n-1) + T(n/2)$$



Nível	Tamanho	Num Nós	Tempo Por nó
0	n	1	1

Nível	Tamanho	Num Nós	Tempo Por nó
1	$n-1$, $n/2$	2	1
2	$n-2$, $n/4$	4	1
i	$n-i, n$ $/2^i$	2^i	1
$n-1$	1, 1	2^{n-1}	1

$$T(n) = \sum_{i=0}^{n-1}(2^{n-1})$$

$$T(n) = O(2^{n-1})$$

Demonstração:

$$T(n-1) \leq c * 2^{(n-1)-1}$$

$$T(n/2) \leq c * 2^{(n/2)-1}$$

$$T(n) \leq c * 2^{n-2} + c * 2^{(n/2)-1}$$

Usando o comportamento assintótico:

$$T(n) \leq c * 2^{n-2} * 2$$

chegamos em:

$$T(n) \leq c * 2^{n-1}$$

Ou seja, é valido:

$$T(n) = O(g(n))$$

Questão 5)

Item a)

```
CriarSubMatrizes(Mat):
    fronteira = ArredondaBaixo(Tamanho(Mat) / 2)



    indicesMat1 = ParaTodos(i <= fronteira)
     e ParaTodos(j <= fronteira)
    indicesMat2 = ParaTodos(i <= fronteira)
     e ParaTodos(j > fronteira e j <= Tamanho(Mat))
    indicesMat3 = ParaTodos(i > fronteira e i < Tamanho(Mat))
     e ParaTodos(j <= fronteira)
    indicesMat4 = ParaTodos(i > fronteira e i < Tamanho(Mat))
     e ParaTodos(j > fronteira e j <= Tamanho(Mat))
```

```
    Mat1 = Mat[indicesMat1]
    Mat2 = Mat[indicesMat2]
    Mat3 = Mat[indicesMat3]
    Mat4 = Mat[indicesMat4]

    retorna Mat1, Mat2, Mat3, Mat4

CriaMatrizFinal(Mat1, Mat2, Mat3, Mat4):

    Retorna Mat[[Mat1, Mat2], [Mat3, Mat4]]

Strassen(Mat_A, Mat_B):
    se Tamanho(Mat_A) = 1 e Tamanho(Mat_B) = 1:
        Retorna Mat_A * Mat_B

    Mat_A1, Mat_A2, Mat_A3, Mat_A4 = CriarSubMatrizes(Mat_A)
    Mat_B1, Mat_B2, Mat_B3, Mat_B4 = CriarSubMatrizes(Mat_B)

    Mult1 = Strassen(Mat_A1 + Mat_A4, Mat_B1 + Mat_B4)
    Mult2 = Strassen(Mat_A3 + Mat_A4, Mat_B1)
    Mult3 = Strassen(Mat_A1, Mat_B2 - Mat_B4)
    Mult4 = Strassen(Mat_A4, Mat_B3 - Mat_B1)
    Mult5 = Strassen(Mat_A1 + Mat_A2, Mat_B4)
    Mult6 = Strassen(Mat_A3 - Mat_A1, Mat_B1 + Mat_B2)
    Mult7 = Strassen(Mat_A2 - Mat_A4, Mat_B3 + Mat_B4)

    Mat_C1 = Mult1 + Mult4 - Mult5 + Mult7
    Mat_C2 = Mult3 + Mult5
    Mat_C3 = Mult2 + Mult4
    Mat_C4 = Mult1 - Mult2 + Mult3 + Mult6

    Mat_C = CriaMatrizFinal(Mat_C1, Mat_C2, Mat_C3, Mat_C4)

    retorna Mat_C
```

Item B)

$$A = \begin{pmatrix} 1 & 3 \\ 7 & 5 \end{pmatrix}, B = \begin{pmatrix} 6 & 8 \\ 4 & 2 \end{pmatrix}$$

Seguindo o algoritmo precisamos efetuar a criação das sub-matrizes:

$$A1 = \begin{pmatrix} 1 \end{pmatrix}, A2 = \begin{pmatrix} 3 \end{pmatrix}, A3 = \begin{pmatrix} 7 \end{pmatrix}, A4 = \begin{pmatrix} 5 \end{pmatrix}$$

$$B1 = \begin{pmatrix} 6 \end{pmatrix}, B2 = \begin{pmatrix} 8 \end{pmatrix}, B3 = \begin{pmatrix} 4 \end{pmatrix}, B4 = \begin{pmatrix} 2 \end{pmatrix}$$

Agora, vamos "Chamar" novamente o método de Strassen, porém irá cair no caso base.

$$
\begin{gathered}
Mult1 = ((6) * (8)),\\
Mult2 = ((12) * (6)),\\
Mult3 = ((1) * (6)),\\
Mult4 = ((5) * (-2)),\\
Mult5 = ((4) * (2)),\\
Mult6 = ((6) * (14)),\\
Mult7 = ((-2) * (6))
\end{gathered}
$$

Calcular as Parciais da matriz C

$$
\begin{gathered}
C1 = 48 + (-10) - (8) + (-12) = 18\\
C2 = 6 + 8 = 14\\
C3 = 72 - 10 = 62\\
C4 = 48 - 72 + 6 + 84 = 66
\end{gathered}
$$

Agora vamos criar a matriz final:

Nesse Caso:

$$
C = \begin{pmatrix} C1 & C2 \\ C3 & C4 \end{pmatrix}
$$

$$
C = \begin{pmatrix} 18 & 14 \\ 62 & 66 \end{pmatrix}
$$

Item C)

$$
T(n) = 7T(n/2) + n
$$

Questão 6)

Item A) QuickSort Implementado

```
"""
// @DOCSTART
// ## quicksort.py  @NL
// Escrito Por: Davi Coelho  @NL
// Data: 09/01/2025  @NL
//  @NL
// Implementação QuickSort com diversas funções de Partições.  @NL
// @DOCEND
"""



"""
// @DOCSTART
// ### Funções de Partição:  @NL
// initPartition  @NL
// endPartition (Default)  @NL
```

```
// randomPartition  @NL
// certerPartition  @NL
// ### Função Principal:  @NL
// quickSort (Principal do arquivo)  @NL
// @DOCEND
"""




from math import ceil
from random import randint
import sys
def initPartition(arr, left, right):
    pivot = arr[left]

    changeIndexLeft = left + 1  # Pq o Left já é o Pivô
    changeIndexRight = right  # Pq vai ser necessário fazer uma 'comparação dupla'

    while True:
        while changeIndexLeft <= changeIndexRight and arr[changeIndexRight] >= pivot:
            changeIndexRight = changeIndexRight - 1

        while changeIndexLeft <= changeIndexRight and arr[changeIndexLeft] <= pivot:
            changeIndexLeft = changeIndexLeft + 1

        if changeIndexLeft <= changeIndexRight:
            _changePosHelper(arr=arr, goal=changeIndexLeft,
                             start=changeIndexRight)
        else:

            break

    _changePosHelper(arr=arr, goal=changeIndexRight, start=left)
    return changeIndexRight



def endPartition(arr, left, right):
    pivot = arr[right]
    changeIndex = left - 1
    for checkIndex in range(left, right):
        if arr[checkIndex] <= pivot:
            changeIndex = changeIndex + 1
            _changePosHelper(arr=arr, start=checkIndex, goal=changeIndex)

    _changePosHelper(arr=arr, goal=changeIndex+1, start=right)
```

```python
    # arr[changeIndex+1], arr[right] = arr[right], arr[changeIndex+1]
    return changeIndex+1


def randomPartition(arr, left, right):
    randomValue = randint(left, right)
    pivot = arr[randomValue]

    _changePosHelper(arr=arr, goal=left, start=randomValue)
    return endPartition(arr=arr, left=left, right=right)


def centerPartition(arr, left, right):

    pivot = arr[(left + right) // 2]
    # print("pivot inicial: pos, value", ((left + right) // 2), pivot)
    _changePosHelper(arr=arr, goal=left, start=(left + right) // 2)
    changeIndexLeft = left
    changeIndexRight = right

    while (changeIndexLeft < changeIndexRight):
        # Lê da esquerda para direita
        while changeIndexLeft <= changeIndexRight and arr[changeIndexLeft] <= pivot:
            changeIndexLeft = changeIndexLeft + 1

        # Lê da direita para a esquerda
        while changeIndexLeft <= changeIndexRight and arr[changeIndexRight] >= pivot:
            changeIndexRight = changeIndexRight - 1

        # Troca o da direita com o da esquerda
        if (changeIndexLeft < changeIndexRight):
            _changePosHelper(arr=arr, goal=changeIndexRight,
                             start=changeIndexLeft)
        else:
            break
    _changePosHelper(arr=arr, goal=changeIndexRight, start=left)
    # arr[left], arr[changeIndexLeft] = arr[changeIndexLeft], arr[left]
    # return (left + right) // 2
    return changeIndexRight


def _changePosHelper(arr, start, goal):
    # print("swap: [pos, value]", goal, arr[goal], start, arr[start])
    arr[goal], arr[start] = arr[start], arr[goal]
    return
```

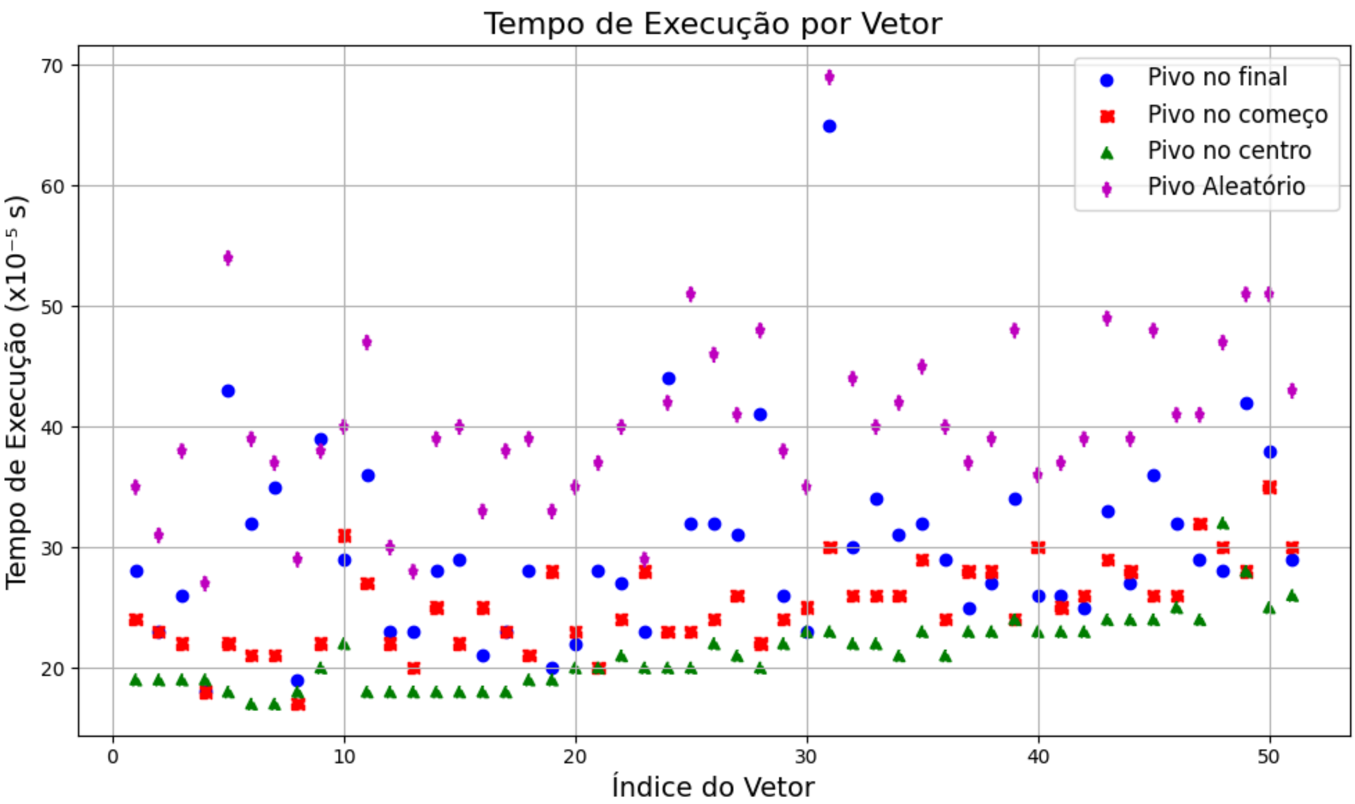
```python
# // @DOCSTART
# // @CBS python
def quickSort(arr: list, left: int, right: int, partitionFunction=endPartition):
    # print('left: ', left)
    # print('right: ', right)
    if left < right:
        pivotIndex = partitionFunction(arr, left, right)
        # print('pivot: ', pivotIndex)
        quickSort(arr=arr, left=left, right=pivotIndex -
                  1, partitionFunction=partitionFunction)
        quickSort(arr=arr, left=pivotIndex+1, right=right,
                  partitionFunction=partitionFunction)



# // @CBE
# // @NL
# // @DOCEND



def test(inputArray: str):
    strArray = inputArray.split(
        ' ')
    intArray = [int(string) for string in strArray]
    # generated = generateArray()
    # intArray = generated
    print("Init State")
    print(intArray)

    print("Random Partition")
    quickSort(intArray, 0, len(intArray) - 1, randomPartition)
    print(intArray)

    intArray = [int(string) for string in strArray]
    # intArray = generated
    quickSort(intArray, 0, len(intArray) - 1, endPartition)
    print("End Partition")
    print(intArray)

    intArray = [int(string) for string in strArray]
    # intArray = generated
    quickSort(intArray, 0, len(intArray) - 1, initPartition)
    print("Init Partition")
    print(intArray)

    intArray = [int(string) for string in strArray]
```

```
    # intArray = generated
    quickSort(intArray, 0, len(intArray) - 1, centerPartition)
    print("Center Partition")
    print(intArray)


def runTests():
    print(sys.getrecursionlimit())
    with open('data.txt', 'r') as f:
        lines = f.readlines()
        i = 0
        for line in lines:

            print("new exec, num: ", i)

            test(str(line))

            print("\n\n\n")
            i = i + 1
        f.close()


def generateArray():
    import random
    array = random.sample(range(1_000_000), 100_000)

    print(array)
    return array


import matplotlib.pyplot as plt
import time
from quicksort import *


def getExecutionTime(callFunction):
    initTime = time.time()
    callFunction()
    endTime = time.time()
    execTime = endTime - initTime
    print('Duration: {}'.format(execTime))


"""
Outros testes
"""
```

```
with open('data.txt', 'r') as f:
    lines = f.readlines()
    i = 0
    timearrEndPartition = []
    timearrinitPartition = []
    timearrCenterPartition = []
    timearrRandomPartition = []
    for line in lines:

        strArray = line.split(
            ' ')
        intArray = [int(string) for string in strArray]
        initTime = time.time()
        quickSort(intArray, 0, len(intArray) - 1, endPartition)
        endTime = time.time()
        execTime = endTime - initTime
        timearrEndPartition.append(int(execTime * (10**5)))

        initTime = time.time()
        intArray = [int(string) for string in strArray]
        quickSort(intArray, 0, len(intArray) - 1, initPartition)
        endTime = time.time()
        execTime = endTime - initTime
        timearrinitPartition.append(int(execTime * (10**5)))

        initTime = time.time()
        intArray = [int(string) for string in strArray]
        quickSort(intArray, 0, len(intArray) - 1, centerPartition)
        endTime = time.time()
        execTime = endTime - initTime
        timearrCenterPartition.append(int(execTime * (10**5)))

        initTime = time.time()
        intArray = [int(string) for string in strArray]
        quickSort(intArray, 0, len(intArray) - 1, randomPartition)
        endTime = time.time()
        execTime = endTime - initTime
        timearrRandomPartition.append(int(execTime * (10**5)))

    print(timearrEndPartition)
    print(timearrinitPartition)
    print(timearrCenterPartition)
    print(timearrRandomPartition)
    f.close()
```

```
num_vectors = 51


plt.figure(figsize=(10, 6))
plt.plot(range(1, num_vectors + 1), timearrEndPartition, marker='o',
         linestyle='-', color='b', label="Pivo no final")
plt.plot(range(1, num_vectors + 1), timearrinitPartition, marker='s',
         linestyle='--', color='r', label="Pivo no começo")
plt.plot(range(1, num_vectors + 1), timearrCenterPartition,
         marker='^', linestyle='-.', color='g', label="Pivo no centro")
plt.plot(range(1, num_vectors + 1), timearrRandomPartition,
         marker='d', linestyle=':', color='m', label="Pivo Aleatório")

plt.title("Tempo de Execução por Vetor", fontsize=16)
plt.xlabel("Índice do Vetor", fontsize=14)
plt.ylabel("Tempo de Execução (x10⁻⁵ s)", fontsize=14)
plt.grid(True)
plt.legend(fontsize=12)
plt.tight_layout()

plt.show()


plt.figure(figsize=(10, 6))
plt.scatter(range(1, num_vectors + 1), timearrEndPartition,
            marker='o', linestyle='-', color='b', label="Pivo no final")
plt.scatter(range(1, num_vectors + 1), timearrinitPartition,
            marker='s', linestyle='--', color='r', label="Pivo no começo")
plt.scatter(range(1, num_vectors + 1), timearrCenterPartition,
            marker='^', linestyle='-.', color='g', label="Pivo no centro")
plt.scatter(range(1, num_vectors + 1), timearrRandomPartition,
            marker='d', linestyle=':', color='m', label="Pivo Aleatório")

#
plt.title("Tempo de Execução por Vetor", fontsize=16)
plt.xlabel("Índice do Vetor", fontsize=14)
plt.ylabel("Tempo de Execução (x10⁻⁵ s)", fontsize=14)
plt.grid(True)
plt.legend(fontsize=12)
plt.tight_layout()

plt.show()
```

Item B) Graficos



Item C) Conclusões

O Pivo no centro se mostrou mais rapido na ordenação dos vetores dados como exemplos. Esse caso da função de Partição é mais complexo de implementar pois as comparações ocorrem tanto do lado direito quanto do lado esquerdo do vetor simultaneamente, podendo fazer uma quantidade maior de swaps por iteração. O algoritmo de particionamento mais comum (o pivo no final) possui um desempenho pior que o pivo no centro (nesses casos apresentados) porém é bem mais simples de implementar. O Pivo aleatorio é o que possui pior desempenho (Por se

tratar de um caso especial do pivo no inicio) possui um desempenho pior, podendo variar dependendo das "escolhas" aléatorias de posição do pivo. O pivo no inicio tem um desempenho semelhante ao pivo no centro, porém a implementação do dois é muito parecida sendo melhor utilizar o pivo no centro.
Pelo fato de serem vetores pequenos a diferença entre os metodos de particionamento não é gritante, em casos de vetores maiores é necessário escolher bem qual o algoritmo de particionamento do vetor.

Questão 7)

Pseudo Código:

```
Fib(posição):
    arrayValores = [0, 1, null]
    contador = 2
    se posição = 1
        retorna arrayValores[0]

    se posição = 2
        retorna arrayValores[1]

    enquanto contador < posicao:
        arrayValores[2] = arrayValores[0] + arrayValores[1]
        contador++
        arrayValores[0] = arrayValores[1]
        arrayValores[1] = arrayValores[2]

    retorna arrayValores[2]
```

Questão 8)

Item a)

Podemos cortar de 512 formas, obedecendo a regra:

$$cortes = 2^{n-1}$$

Isso ocorre por conta de subcortes.

Item B)

Usando o seguinte algoritmo o resultado foi 30 (Valor ótimo)

```
import math



def corta_haste(p: list, n: int):
```

```
if (n == 0):
    return 0

auxiliar = - math.inf
i = 0
while i < n:
    # print(i)
```