CENTRO UNIVERSITÁRIO INTERNACIONAL UNINTER
ESCOLA SUPERIOR POLITÉCNICA
ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
ESTRUTURA DE DADOS

ATIVIDADE PRÁTICA

THATIANA DE ASSIS NAPOLITANO – RU: 4302056

RIO DE JANEIRO - RJ
2024

1 EXERCÍCIO 1

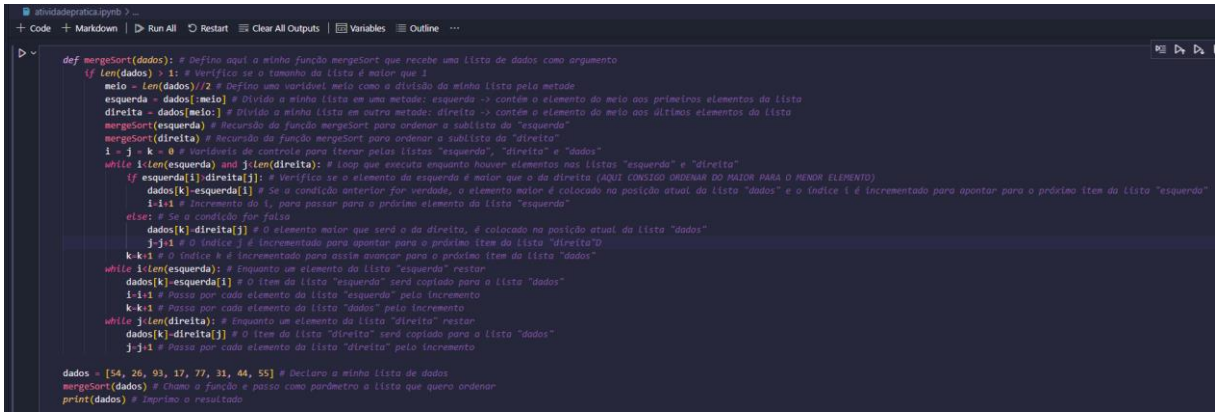
- Enunciado da Questão:

O algoritmo de ordenação por intercalação, ou Merge Sort, usufrui da estratégia de dividir para conquistar. O Merge Sort realiza a ordenação dividindo um conjunto de dados em metades iguais e reorganizando essas metades. É um algoritmo que opera de maneira recursiva, dividindo de maneira contínua o conjunto de dados até eles tornarem-se indivisíveis.” Abaixo temos o código da função em Python do algoritmo Merge Sort.

Reescreva a função “mergeSort” de forma que ela realize a ordenação dos elementos do maior para o menor elemento. Por exemplo: Para a entrada: [54, 26, 93, 17, 77, 31, 44, 55] A saída deve ser: [93, 77, 55, 54, 44, 31, 26, 17] Além de reescrever a função, você estudante deve comentar todas as linhas de código, explicando o que está sendo realizado. Não se esqueça de testar o código e anexar a print do terminal no documento de entrega.

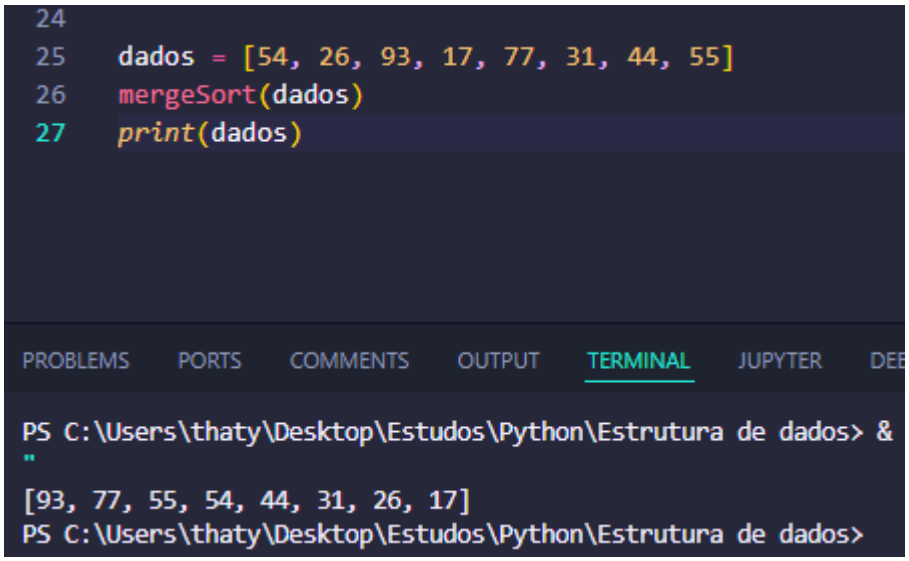
- Código completo:

```
def mergeSort(dados):
    if len(dados) > 1:
        meio = len(dados)//2
        esquerda = dados[:meio]
        direita = dados[meio:]
        mergeSort(esquerda)
        mergeSort(direita)
        i = j = k = 0
        while i<len(esquerda) and j<len(direita):
            if esquerda[i]>direita[j]:
                dados[k]=esquerda[i]
                i=i+1
            else:
                dados[k]=direita[j]
                j=j+1
            k=k+1
        while i<len(esquerda):
            dados[k]=esquerda[i]
            i=i+1
            k=k+1
        while j<len(direita):
            dados[k]=direita[j]
            j=j+1

dados = [54, 26, 93, 17, 77, 31, 44, 55]
mergeSort(dados)
print(dados)
```



```
def mergeSort(dados): # Defino aqui a minha função mergeSort que recebe uma lista de dados como argumento
    if len(dados) > 1: # Verifico se o tamanho da lista é maior que 1
        meio = len(dados)//2 # Defino uma variável meio como a divisão da minha lista pela metade
        esquerda = dados[:meio] # Divido a minha lista em uma metade: esquerda -> contém o elemento do meio aos primeiros elementos da lista
        direita = dados[meio:] # Divido a minha lista em outra metade: direita -> contém o elemento do meio aos últimos elementos da lista
        mergeSort(esquerda) # Recursão da função mergeSort para ordenar a sublista da "esquerda"
        mergeSort(direita) # Recursão da função mergeSort para ordenar a sublista da "direita"
        i = j = k = 0 # Variáveis de controle para iterar pelas listas "esquerda", "direita" e "dados"
        while i<len(esquerda) and j<len(direita): # Loop que executa enquanto houver elementos nas listas "esquerda" e "direita"
            if esquerda[i]>direita[j]: # Verifico se o elemento da esquerda é maior que o da direita (AQUI CONSIGO ORDENAR DO MAIOR PARA O MENOR ELEMENTO)
                dados[k]=esquerda[i] # Se a condição anterior for verdade, o elemento maior é colocado na posição atual da lista "dados" e o índice i é incrementado para apontar para o próximo item da lista "esquerda"
                i=i+1 # Incremento do 1, para passar para o prórimo elemento da lista "esquerda"
            else: # Se a condição for falsa
                dados[k]=direita[j] # O elemento maior que será o da direita, é colocado na posição atual da lista "dados"
                j=j+1 # O índice j é incrementado para apontar para o próximo item da lista "direita":D
            k=k+1 # O índice k é incrementado para assim avançar para o próximo item da lista "dados"
        while i<len(esquerda): # Enquanto um elemento da lista "esquerda" restar
            dados[k]=esquerda[i] # O item da lista "esquerda" será copiado para a lista "dados"
            i=i+1 # Passo por cada elemento da lista "esquerda" pelo incremento
            k=k+1 # Passa por cada elemento da lista "dados" pelo incremento
        while j<len(direita): # Enquanto um elemento da lista "direita" restar
            dados[k]=direita[j] # O item da lista "direita" será copiado para a lista "dados"
            j=j+1 # Passa por cada elemento da lista "direita" pelo incremento

dados = [54, 26, 93, 17, 77, 31, 44, 55] # Declaro a minha lista de dados
mergeSort(dados) # Chamo a função e passo como parâmetro a lista que quero ordenar
print(dados) # Imprimo o resultado
```

- Imagem do código funcionando no seu computador (print do terminal):



```
24
25  dados = [54, 26, 93, 17, 77, 31, 44, 55]
26  mergeSort(dados)
27  print(dados)
```

PROBLEMS PORTS COMMENTS OUTPUT TERMINAL JUPYTER DEE

```
PS C:\Users\thaty\Desktop\Estudos\Python\Estrutura de dados> &
"
[93, 77, 55, 54, 44, 31, 26, 17]
PS C:\Users\thaty\Desktop\Estudos\Python\Estrutura de dados>
```

2 EXERCÍCIO 2

- Enunciado da Questão:

Você e seus amigos decidiram desenvolver um jogo de cartas em linguagem python, e você ficou encarregado de desenvolver as funções: embaralhaCartas e compraCarta. As 52 cartas do baralho estão na lista de string abaixo:

```
baralho = ["A-Copas","A-Paus","A-Espadas","A-Ouros",
 "2-Copas","2-Paus","2-Espadas","2-Ouros",
 "3-Copas","3-Paus","3-Espadas","3-Ouros",
 "4-Copas","4-Paus","4-Espadas","4-Ouros",
 "5-Copas","5-Paus","5-Espadas","5-Ouros",
 "6-Copas","6-Paus","6-Espadas","6-Ouros",
 "7-Copas","7-Paus","7-Espadas","7-Ouros",
 "8-Copas","8-Paus","8-Espadas","8-Ouros",
 "9-Copas","9-Paus","9-Espadas","9-Ouros",
 "10-Copas","10-Paus","10-Espadas","10-Ouros",
 "J-Copas","J-Paus","J-Espadas","J-Ouros",
```

```
 "Q-Copas","Q-Paus","Q-Espadas","Q-Ouros",
 "K-Copas","K-Paus","K-Espadas","K-Ouros"]
```

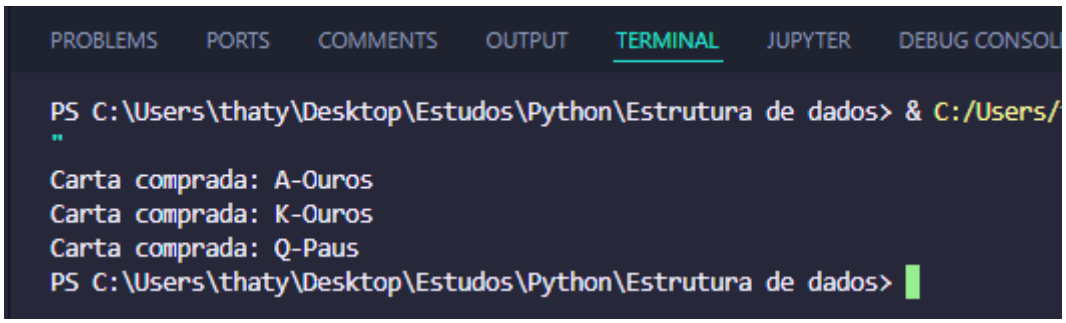
A função “embaralhaCartas”, deve receber a lista “baralho” e retornar uma pilha de cartas, a qual será utilizada pela função “compraCartas”. A função “compraCartas” vai receber a pilha de cartas e retirar a carta do topo da pilha, e imprimindo a mesma na tela.

- Código completo:

```
# importo a biblioteca random para utilizar método de embaralho
import random

# Definição da classe para representar uma pilha de cartas
class Stack:
    def __init__(self):
        self.items = []  # Inicializa a lista que vai armazenar as cartas da
pilha

    def isEmpty(self):
        return self.items == []  # Verifica se a pilha está vazia

    def push(self, item):
        self.items.append(item)  # Adiciona um item ao topo da pilha

    def pop(self):
        return self.items.pop()  # Remove e retorna o item do topo da pilha

# Função para embaralhar o baralho e retornar uma pilha de cartas
def embaralhaCartas(baralho):
    random.shuffle(baralho)  # Embaralha o baralho
    pilha_de_cartas = Stack()  # Cria uma nova instância da classe Stack para
representar a pilha
    for carta in baralho: # Para cada carta no baralho
        pilha_de_cartas.push(carta)  # Adiciona cada carta embaralhada à pilha
    return pilha_de_cartas  # Retorna a pilha de cartas

# Função para comprar uma carta da pilha e imprimir na tela
def compraCartas(pilha_de_cartas):
    if not pilha_de_cartas.isEmpty():  # Verifica se a pilha não está vazia
        carta_comprada = pilha_de_cartas.pop()  # Remove a carta do topo da
pilha
        print("Carta comprada:", carta_comprada)  # Imprime a carta comprada
    else:
        print("A pilha de cartas está vazia.")  # Caso a pilha esteja vazia,
imprime uma mensagem de aviso
```

```python
# Exemplo de uso

baralho = ["A-Copas","A-Paus","A-Espadas","A-Ouros",
 "2-Copas","2-Paus","2-Espadas","2-Ouros",
 "3-Copas","3-Paus","3-Espadas","3-Ouros",
 "4-Copas","4-Paus","4-Espadas","4-Ouros",
 "5-Copas","5-Paus","5-Espadas","5-Ouros",
 "6-Copas","6-Paus","6-Espadas","6-Ouros",
 "7-Copas","7-Paus","7-Espadas","7-Ouros",
 "8-Copas","8-Paus","8-Espadas","8-Ouros",
 "9-Copas","9-Paus","9-Espadas","9-Ouros",
 "10-Copas","10-Paus","10-Espadas","10-Ouros",
 "J-Copas","J-Paus","J-Espadas","J-Ouros",
 "Q-Copas","Q-Paus","Q-Espadas","Q-Ouros",
 "K-Copas","K-Paus","K-Espadas","K-Ouros"]  # Lista de cartas disponíveis no
baralho


pilhaCartas = embaralhaCartas(baralho) # Pilha de cartas embaralhadas
compraCartas(pilhaCartas) # Imprimo a carta comprada
compraCartas(pilhaCartas) # Imprimo a carta comprada
compraCartas(pilhaCartas) # Imprimo a carta comprada
```

- Imagem do código funcionando no seu computador (print do terminal):



3 EXERCÍCIO 3

- Enunciado da Questão:

Com a finalidade de melhorar o atendimento e priorizar os casos mais urgentes, a direção de um hospital criou um sistema de triagem em que um profissional da saúde classifica a ordem de atendimento com base numa avaliação prévia do paciente, entregando-lhe um cartão numerado verde (V) ou amarelo (A), que define o menor ou maior grau de urgência da ocorrência,

respectivamente. Para informatizar esse processo, o software desenvolvido tem como base o seguinte trecho de código-fonte:

- Código completo:

Na linha 31, a função inserir recebe o número e a cor do cartão entregue ao paciente na triagem. Pacientes com cartão verde são inseridos no final da fila pela função inserirNoFinal (linhas 11-16). Pacientes com cartão amarelo têm prioridade no atendimento e são inseridos no início da fila, em ordem de chegada (ou seja, um paciente com cartão amarelo será inserido após os outros pacientes com cartão amarelo que já estão na fila), pela função inserirPrioridade (linha 18-20). Portanto, se os seguintes trechos de códigos são executados:

```
#programa principal
filaPacientes = ListaEncadeadaSimples() #cria a lista que ira receber os dados
dos pacientes

filaPacientes.inserir(1, "V") #insere um paciente com senha "V" 1
filaPacientes.inserir(2, "V") #insere um paciente com senha "V" 2
filaPacientes.inserir(101, "A") #insere um paciente com senha "A" 101
filaPacientes.inserir(3, "V") #insere um paciente com senha "V" 3
filaPacientes.inserir(102, "A") #insere um paciente com senha "A" 102
filaPacientes.inserir(103, "A") #insere um paciente com senha "A" 103
filaPacientes.inserir(4, "V") #insere um paciente com senha "V" 4
filaPacientes.inserir(104, "A") #insere um paciente com senha "A" 104
filaPacientes.inserir(105, "A") #insere um paciente com senha "A" 105
nodo_atual = filaPacientes.head
while nodo_atual is not None:
    print(f"Cartão: {nodo_atual.cor}, Senha: {nodo_atual.dado}")
    nodo_atual = nodo_atual.proximo
```

A saída deve ser:

```
Cartão: A, Senha: 101
Cartão: A, Senha: 102
Cartão: A, Senha: 103
Cartão: A, Senha: 104
Cartão: A, Senha: 105
Cartão: V, Senha: 1
Cartão: V, Senha: 2
Cartão: V, Senha: 3
Cartão: V, Senha: 4
```

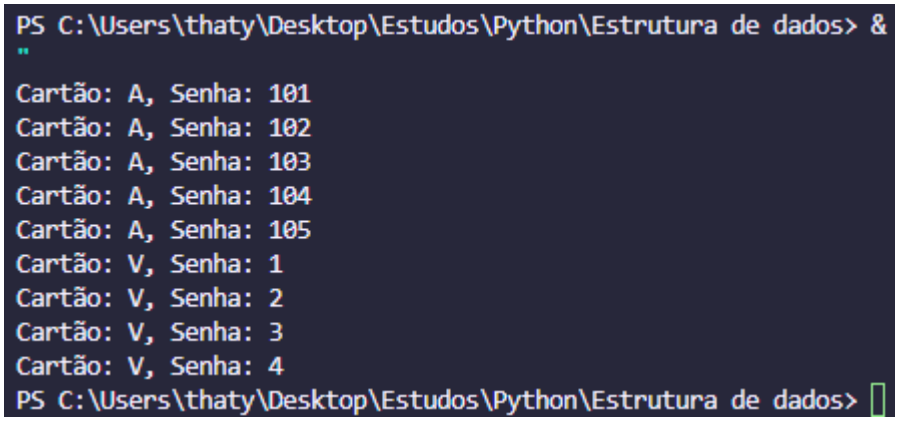
Considerando o processo de triagem descrito e os trechos de código-fonte apresentados, implemente a função inserirPrioridade conforme indicado (linha 18-20) de forma que a saída esteja correta. Você estudante deve comentar todas as linhas de código desenvolvidas (função inserirPrioridade apenas), explicando o que está sendo realizado. Não se esqueça de testar o código

e anexar a print do terminal no documento de entrega. Deve-se utilizar apenas a estrutura de Lista Encadeada Simples, a utilização de outras estruturas, listas ou bibliotecas, acarretará redução de nota.

```
# Classe que representa um elemento individual da lista encadeada
class ElementoDaListaSimples:
    # Método construtor que inicializa o elemento com um dado e uma cor
    def __init__(self, dado, cor):
        self.dado = dado  # O dado armazenado no elemento (número do cartão)
        self.cor = cor  # A cor do cartão (verde ou amarelo)
        self.proximo = None  # O próximo elemento na lista encadeada, inicial-
mente None

# Classe que representa a lista encadeada simples
class ListaEncadeadaSimples:
    # Método construtor que inicializa a lista encadeada
    def __init__(self, nodos=None):
        self.head = None  # O primeiro elemento da lista, inicialmente None

    # Método para inserir um elemento no final da lista
    def inserirNoFinal(self, nodo):
        if not self.head:  # Se a lista estiver vazia
            self.head = nodo  # O novo nodo se torna o primeiro elemento
        else:  # Se a lista não estiver vazia
            nodo_atual = self.head  # Começa pelo primeiro elemento
            while nodo_atual.proximo:  # Enquanto houver um próximo elemento
                nodo_atual = nodo_atual.proximo  # Avança para o próximo ele-
mento
            nodo_atual.proximo = nodo  # Insere o novo nodo no final da lista

    # Método para inserir um elemento com prioridade (amarelo) na lista
    def inserirPrioridade(self, nodo):
        if not self.head or self.head.cor == "V":  # Se a lista estiver vazia
ou o primeiro elemento for verde
            nodo.proximo = self.head  # O novo nodo aponta para o primeiro
elemento atual
            self.head = nodo  # O novo nodo se torna o primeiro elemento
        else:  # Se o primeiro elemento for amarelo
            nodo_atual = self.head  # Começa pelo primeiro elemento
            # Enquanto houver um próximo elemento e esse elemento for amarelo
            while nodo_atual.proximo and nodo_atual.proximo.cor == "A":
                nodo_atual = nodo_atual.proximo  # Avança para o próximo ele-
mento
            # Insere o novo nodo após o último amarelo e antes do primeiro
verde
            nodo.proximo = nodo_atual.proximo
            nodo_atual.proximo = nodo
```

```
    # Método para inserir um elemento na lista, considerando sua cor
    def inserir(self, dado, cor):
        novo_nodo = ElementoDaListaSimples(dado, cor)  # Cria um novo elemento
        if cor == "V":  # Se a cor for verde
            self.inserirNoFinal(novo_nodo)  # Insere no final da lista
        else:  # Se a cor for amarela
            self.inserirPrioridade(novo_nodo)  # Insere com prioridade

# Programa principal
filaPacientes = ListaEncadeadaSimples()  # Cria a lista que irá receber os dados dos pacientes
# Insere pacientes na lista, passando o número do cartão e a cor como argumentos
filaPacientes.inserir(1, "V")
filaPacientes.inserir(2, "V")
filaPacientes.inserir(101, "A")
filaPacientes.inserir(3, "V")
filaPacientes.inserir(102, "A")
filaPacientes.inserir(103, "A")
filaPacientes.inserir(4, "V")
filaPacientes.inserir(104, "A")
filaPacientes.inserir(105, "A")

# Itera sobre a lista encadeada e imprime a cor e o número do cartão de cada paciente
nodo_atual = filaPacientes.head
while nodo_atual:
    print(f"Cartão: {nodo_atual.cor}, Senha: {nodo_atual.dado}")
    nodo_atual = nodo_atual.proximo
```

- Imagem do código funcionando no seu computador (print do terminal):



4 EXERCÍCIO 4

- Enunciado da Questão:

Uma árvore binária, por não ser uma estrutura linear, apresenta distintas maneiras de se percorrer por ela para visualizar, manipular ou processar os dados da árvore. Dado o código abaixo de uma árvore binária de busca, escreva uma função chamada “folhas“, que irá retornar um vetor contendo apenas os nós folha da arvore (nós sem filhos).

- Código completo:

```
class BST:
    def __init__(self, dado=None):
        self.dado = dado
        self.esquerda = None
        self.direita = None

    def inserir(self, dado):
        if (self.dado == None):
            self.dado = dado
        else:
            if (dado < self.dado):
                if (self.esquerda):
                    self.esquerda.inserir(dado)
                else:
                    self.esquerda = BST(dado)
            else:
                if(self.direita):
                    self.direita.inserir(dado)
                else:
                    self.direita = BST(dado)

        # Função para encontrar os nós folha da árvore
    def folhas(self, lst):
        # Se o nó atual não tem filhos, é um nó folha e adicionamos seu valor
ao vetor
        if self.esquerda is None and self.direita is None:
            lst.append(self.dado)
        # Se o nó atual tem filho à esquerda, percorremos esse sub-árvore
        if self.esquerda is not None:
            self.esquerda.folhas(lst)
        # Se o nó atual tem filho à direita, percorremos essa sub-árvore
        if self.direita is not None:
            self.direita.folhas(lst)
        # Retornamos o vetor com os valores dos nós folha
```

```
        return lst

# Código para testar a função folhas
Teste = BST()
Teste.inserir(7)
Teste.inserir(4)
Teste.inserir(9)
Teste.inserir(0)
Teste.inserir(5)
Teste.inserir(8)
Teste.inserir(13)
print('Folhas: ', Teste.folhas([]))  # Saída esperada: Folhas: [0, 5, 8, 13]
```

- Imagem do código funcionando no seu computador (print do terminal):