1 Exercío 1 - Par de Parcelas Mínimas

```
def mergeSort(vetor):
    if len(vetor) <= 1:
        return vetor

    meio = len(vetor) // 2
    esquerda = vetor[:meio]
    direita = vetor[meio:]

    esquerda = mergeSort(esquerda)
    direita = mergeSort(direita)

    return merge(esquerda, direita)

def merge(esquerda, direita):
    resultado = []
    i = 0
    j = 0

    while i < len(esquerda) and j < len(direita):
        if esquerda[i] < direita[j]:
            resultado.append(esquerda[i])
            i += 1
        else:
            resultado.append(direita[j])
            j += 1

    while i < len(esquerda):
        resultado.append(esquerda[i])
        i += 1

    while j < len(direita):
        resultado.append(direita[j])
        j += 1

    return resultado

def calculaParcelasMinimas(vetor):
    tam = len(vetor)
    n = tam

    parcelas = []

    for i in range(n):
        if(i < n - i - 1):
            parcela = (vetor[i], vetor[n-i-1])
            parcelas.append(parcela)
        else:
```

```python
            break
    return parcelas

def main():
    vetor = [1, 3, 2, 9, 23, 4]

    if len(vetor) % 2 != 0:
        print("Erro! O vetor deve ter um n mero par de
    elementos.")
    else:
        vetor = mergeSort(vetor)

        parcelas = calculaParcelasMinimas(vetor)

        print("As parcelas m nimas seriam: ", parcelas)

main()
```

Listing 1: par de parcelas minimas

Resultado: As parcelas mínimas seriam: [(1, 23), (2, 9), (3, 4)]

Complexidade de tempo

Vamos analisar a complexidade do código por partes:

1. **mergeSort:** - A função ‘mergeSort‘ divide repetidamente o vetor ao meio até que cada subvetor tenha no máximo um elemento. Isso ocorre em logaritmo na base 2 do tamanho do vetor. - Em cada nível da recursão, a função ‘merge‘ é chamada, que tem complexidade O(n), onde n é o tamanho do vetor. Isso ocorre para cada nível da recursão. - Portanto, a complexidade total do ‘mergeSort‘ é O(n log n).

2. **merge:** - A função ‘merge‘ percorre ambas as metades do vetor uma vez, comparando e mesclando os elementos em um novo vetor. - A complexidade é linear em relação ao tamanho do vetor, O(n), onde n é a soma dos tamanhos das duas metades.

3. **calculaParcelasMinimas:** - A função ‘calculaParcelasMinimas‘ percorre metade do vetor (até n/2), onde n é o tamanho do vetor. - A complexidade é O(n/2), mas na análise de complexidade, podemos ignorar constantes e considerar a complexidade como O(n).

4. **main:** - A função ‘main‘ chama ‘mergeSort‘ e ‘calculaParcelasMinimas‘, ambas com complexidade O(n log n) e O(n), respectivamente. Portanto, a complexidade total é dominada pelo ‘mergeSort‘.

Assim, a complexidade total do código é O(n log n), onde n é o tamanho do vetor.

Corretude

Vetor Ordenado

Suponha que temos um vetor ordenado de forma crescente:

$$A[x_1, x_2, \ldots, x_n]$$

Soma Mínima do Par $(x_n, outroelemento)$

Ao considerar o par $(x_n, outroelemento)$, a prova afirma que a soma mínima desse par é (x_1, x_n). Isso faz sentido, pois, em um vetor ordenado crescentemente, x_n é o maior elemento, e escolher x_1 como o outro elemento garante a menor soma.

Pares ao Considerar $(x_{n-1}, outroelemento)$

Ao considerar o par $(x_{n-1}, outroelemento)$, existem duas opções possíveis: (x_1, x_{n-1}) e (x_2, x_n) ou (x_1, x_n) e (x_2, x_{n-1}).

Indução Geral

A prova afirma que, por indução, ao considerar o par $(x_{n-k+1}, outroelemento)$, x_k é o menor elemento que já não está em um par. Isso significa que, à medida que avançamos pelo vetor, escolhemos x_k como o próximo elemento a ser pareado.

Escolha do Par

A justificativa para escolher o par (x_k, x_{n-k+1}) é que, dado que os apontadores no laço principal andam necessariamente um elemento por vez, x_k é o menor elemento que ainda não está em um par.

2 Exercicio - MST

prova por contradição:

Vamos assumir o oposto: que existe uma MST A em que a aresta e não pertence. Se adicionarmos e a A, formaremos um ciclo, uma vez que e possui o custo mínimo. Para manter a conectividade e garantir que continuemos com uma árvore, precisamos remover outra aresta f de A, desde que não seja e, criando assim uma nova árvore B. A remoção de f não quebra a conectividade e mantém $V - 1$ arestas em B.

Agora, considerando os pesos, sabemos que $peso(B) = peso(A) - (f - e)$, e como e tem o custo mínimo, $f - e$ é positivo. Portanto, $peso(B) < peso(A)$. Isso implica que B é uma MST com a aresta de custo mínimo e, contradizendo a suposição inicial de que A é a MST.

Assim, mostramos que a afirmacao de que e não pertence a alguma MST de G é falsa, pois e pertence a toda MST de G.