Viajante.py > ...

```python
from itertools import permutations
import math
# Função para calcular a distância Euclidiana entre duas cidades
def calcular_distancia(cidade1, cidade2):
    x1, y1 = cidade1
    x2, y2 = cidade2
    return math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
# Função para calcular a distância total de uma determinada rota
def calcular_distancia_total(rota, cidades):
    distancia_total = 0
    for i in range(len(rota) - 1):
        distancia_total += calcular_distancia(cidades[rota[i]], cidades[rota[i + 1]])
    # Retornar à cidade de origem
    distancia_total += calcular_distancia(cidades[rota[-1]], cidades[rota[0]])
    return distancia_total

# Função para resolver o problema do Caixeiro Viajante utilizando força bruta
def caixeiro_viajante(cidades):
    n = len(cidades)
    cidades_indices = list(range(n))

    menor_distancia = float('inf')
    melhor_rota = []
    # Gerar todas as permutações possíveis das cidades
    for rota in permutations(cidades_indices):
        distancia_atual = calcular_distancia_total(rota, cidades)
        if distancia_atual < menor_distancia:
            menor_distancia = distancia_atual
            melhor_rota = rota
    return melhor_rota, menor_distancia
# Coordenadas de 10 cidades (x, y)
cidades = [
    (0, 0), (2, 3), (5, 2), (6, 6), (8, 3),
    (3, 7), (7, 5), (1, 8), (4, 4), (9, 0)
]
melhor_rota, menor_distancia = caixeiro_viajante(cidades)
# Printar a solução
print("Melhor rota:", [i + 1 for i in melhor_rota])
print("Distância total:", menor_distancia)
# Printar a sequência de cidades e a distância entre cada uma
print("\nCadeia de cidades e distâncias:")
for i in range(len(melhor_rota)):
    cidade_atual = melhor_rota[i]
    proxima_cidade = melhor_rota[(i + 1) % len(melhor_rota)]
    distancia = calcular_distancia(cidades[cidade_atual], cidades[proxima_cidade])
    print(f"Cidade {cidade_atual + 1} -> Cidade {proxima_cidade + 1}: {distancia:.2f}")

```

análise de complexidade de tempo

```
def calcular_distancia(cidade1, cidade2):
    x1, y1 = cidade1
    x2, y2 = cidade2
    return math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
```

	custo	repetições	subtotal
	c1	1	c1
	c2	1	c2
	c3	1	c3

T(n) = c1 + c2 + c3 -> T(n) = c4 T(n) = O(1)

```
def calcular_distancia_total(rota, cidades):
    distancia_total = 0
    for i in range(len(rota) - 1):
        distancia_total += calcular_distancia(cidades[rota[i]], cidades[rota[i + 1]])

    distancia_total += calcular_distancia(cidades[rota[-1]], cidades[rota[0]])
    return distancia_total
```

	custo	repetições	subtotal
	c1	1	c1
	c2	n	n*c2
	O(1)	n-1	n-1
	O(1)	1	1

T(n) = c1 + n*c2 + n-1 + 1 -> T(n) = n*(c2 + 1) + c1 -> T(n) = O(n)

	custo	repetições	subtotal
def caixeiro_viajante(cidades):			
n = len(cidades)	c1	1	c1
cidades_indices = list(range(n))	c2	n+1	(n+1)*c2
menor_distancia = float('inf')	c3	1	c3
melhor_rota = []	c4	1	c4
for rota in permutations(cidades_indices):	c5	n!	n!*c5
distancia_atual = calcular_distancia_total(rota, cidades)	O(n)	n!	n!*n
if distancia_atual < menor_distancia:	c6	n!	n!*c6
menor_distancia = distancia_atual	c7	n!	n!*c7
melhor_rota = rota	c8	n!	n!*c8
return melhor_rota, menor_distancia			

T(n) = c1 + (n+1)*c2 + c3 + c4 + n!*c5 + n!*n +n!*c6 + n!*c7 + n!*c8 -> T(n) = n!(n + c5 + c6 + c7 + c8) + c1 + n*c2 + c2 + c3 + c4

T(n) = O(n!)

análise de complexidade de espaço

	custo de espaço
def calcular_distancia(cidade1, cidade2):	
x1, y1 = cidade1	2
x2, y2 = cidade2	2
return math.sqrt((x2 - x1)**2 + (y2 - y1)**2)	0

S(n) = 2 + 2 -> S(n) = 4 -> S(n) = O(1)

---

	custo de espaço
def calcular_distancia_total(rota, cidades):	
distancia_total = 0	1
for i in range(len(rota) - 1):	1
distancia_total += calcular_distancia(cidades[rota[i]], cidades[rota[i + 1]])	O(1)
distancia_total += calcular_distancia(cidades[rota[-1]], cidades[rota[0]])	O(1)
return distancia_total	

S(n) = 1 + 1 + 1 + 1 -> S(n) = 4 -> S(n) = O(1)

```
def caixeiro_viajante(cidades):                                                    custo de espaço
    n = len(cidades)                                                                    1
    cidades_indices = list(range(n))                                                    n

    menor_distancia = float('inf')                                                      1
    melhor_rota = []                                                                    1

     for rota in permutations(cidades_indices):                                         n

        distancia_atual = calcular_distancia_total(rota, cidades)                       O(1)
        if distancia_atual < menor_distancia:                                           0
            menor_distancia = distancia_atual                                           0
            melhor_rota = rota                                                          0
    return melhor_rota, menor_distancia
```

S(n) = 1 + n + 1 + 1 + n + 1 -> S(n) = 2*n + 4 -> S(n) = O(n)