ESTUDO DE CASO:

O ALGORITMO DE ORDENAÇÃO BUBBLESORT

Lembremos o problema de ordenação: dada uma sequência de entrada não ordenada (para facilitar considere números naturais distintos), o problema de ordenação é entregar como saída a mesma sequência, porém em ordem crescente.

Você conhecerá agora uma outra forma de resolver o problema de ordenação, o método Bubblesort ou método da ordenação por bolha. O nome desse algoritmo vem da ideia de flutuar um elemento por vez para o fim da lista, como se fosse um conjunto de bolhas flutuando para o topo de um líquido.

Bubblesort – o algoritmo

A ideia do método Bubblesort é você iniciar com um array contendo a lista desordenada. Então, o algoritmo vai percorrer o array fazendo trocas par a par entre dois elementos consecutivos, quando estiverem fora da ordem. Ao chegar ao final do array, você recomeça do início, encerrando o algoritmo quando uma das passagens não realizar nenhuma troca (o que significa que o array está ordenado).

Vamos ver isso na prática? Comece com um array não ordenado [10,5,20,1,4].

Comparar 10 com 5: 10>5, logo é preciso trocar. Array atualizado [5,10,20,1,4]

Comparar 10 com 20: 10<20, não há troca. Array atualizado [5,10,20,1,4]

Comparar 20 com 1: 20>1, logo é preciso trocar. Array atualizado [5,10,1,20,4]

Comparar 20 com 4: 20>4, logo é preciso trocar. Array atualizado [5,10,1,4,20]

Chegamos ao fim da primeira passagem, mas ocorreram trocas. Recomecemos do início.

Comparar 5 com 10:5 <10, não há troca. Array atualizado [5,10,1,4,20]

Comparar 10 com 1:10>1, logo é preciso trocar. Array atualizado [5,1,10,4,20]

Comparar 10 com 4: 10>4, logo é preciso trocar. Array atualizado [5,1,4,10,20]

Comparar 10 com 20: 10<20, não há troca. Array atualizado [5,1,4,10,20]

Chegamos ao fim da segunda passagem, mas ocorreram trocas. Recomecemos do início.

Na terceira passagem, trocaremos 5 com 1, depois 5 com 4. Experimente fazer o passo a passo.

A lista nesse momento está ordenada, mas o algoritmo não sabe disso. Apenas sabe que houve trocas, logo ele precisará de uma quarta passagem para verificar que o array já está ordenado.

Implementando em Python

Você agora está pronto para implementar o bubblesort em Python. Recomendamos que você tente por conta própria, mas seguiremos juntos:

- Para implementar o algoritmo bubblesort, você precisa criar um array para guardar a entrada (que também será a saída).
- Você precisará criar um laço de repetição que percorrerá o array e só terminará quando não acontecerem trocas, ou seja, um laço com variável de controle para encerrar.
- Você precisará criar um código para comparar dois números consecutivos e trocá-los, se necessário.

```
def trocar (seq,i):
    tmp = seq[i]
    seq[i] = seq[i+1]
    seq[i+1] = tmp


sequencia = [10,5,20,1,4]
troca=1
while troca:
        troca=0
        i=0
        for i in range(0,len(sequencia)-1):
                if sequencia[i]>sequencia[i+1]:
                    trocar(sequencia,i)
                    troca=1
print(sequencia)
```

Complexidade do bubblesort

Premissas

Para começar, você precisa determinar qual será o tamanho da entrada, que no caso da ordenação é o tamanho da sequência (não ordenada) de entrada **n**.

Além disso, você precisa dar um custo para cada operação do algoritmo. Ele percorre a lista, realizando uma comparação entre dois elementos. Você pode atribuir o custo **T** para essa operação.

Quando necessário, o algoritmo troca dois elementos, no procedimento trocar(). Analisando o procedimento, você pode atribuir o custo de **3T** para cada troca, baseado nas três operações de atribuição. Se você se lembrar dos princípios de complexidade

de algoritmo, o custo não ser exatamente **3T** não é muito relevante para o comportamento assintótico do algoritmo.

Cálculo de melhor caso

No melhor caso, a lista já está ordenada na entrada. Nesse caso, o algoritmo fará apenas uma passagem, verificando que não precisou realizar nenhuma troca. Em seguida, o algoritmo se encerrará. Nesse caso, a complexidade fica em **(n−1)T**, portanto **O(n)**, ou seja, complexidade linear.

Cálculo de pior caso

No pior caso, a lista está em ordem decrescente na entrada, ou seja, o algoritmo precisará inverter todas as posições.

Na primeira passagem, o primeiro elemento será trocado n−1 vezes, indo para a última posição. Na segunda passagem, o novo primeiro elemento será trocado n−2 vezes, indo para a penúltima posição. E assim por diante, até que na passagem (n− 1), haverá apenas uma troca entre a primeira e a segunda posição. Na n-ésima passagem, o algoritmo verificará que a lista está ordenada, sem trocas.

Passagem	**Custo das comparações**	**Trocas feitas**	**Custo das trocas**
1	**(n−1)t**	**n-1**	**3(n−1)t**
2	**(n−1)t**	**n-2**	**3(n−2)t**
...	**...**	**...**	**...**
n-1	**(n−1)t**	**1**	**3t**
n	**(n−1)t**	**0**	**0**

Você pode observar que o custo tem dois componentes, um devido às passagens e outro devido às trocas. Pelas passagens, temos **n(n−1)t,** ou seja **n(n²).** Pelo lado das trocas, temos **3t∑i=1n−1 i=3t.n(n−1)2**, que também é **O(n²)**. Portanto, no pior caso, o bubblesort também é **O(n²).**

Você pode perceber que o bubblesort possui complexidade **O(n²).** em seu pior caso, mas é um algoritmo mais rápido quanto mais próximo de ordenada a lista de entrada estiver.