Sorting Algorithms

A escolha do “ Melhor” algoritmo de busca depende do contexto específico e das características do problema. A notação BIG O, fornece uma visão geral da eficiência do algoritmo em termos de tempo de execução em relação ao tamanho da entrada.

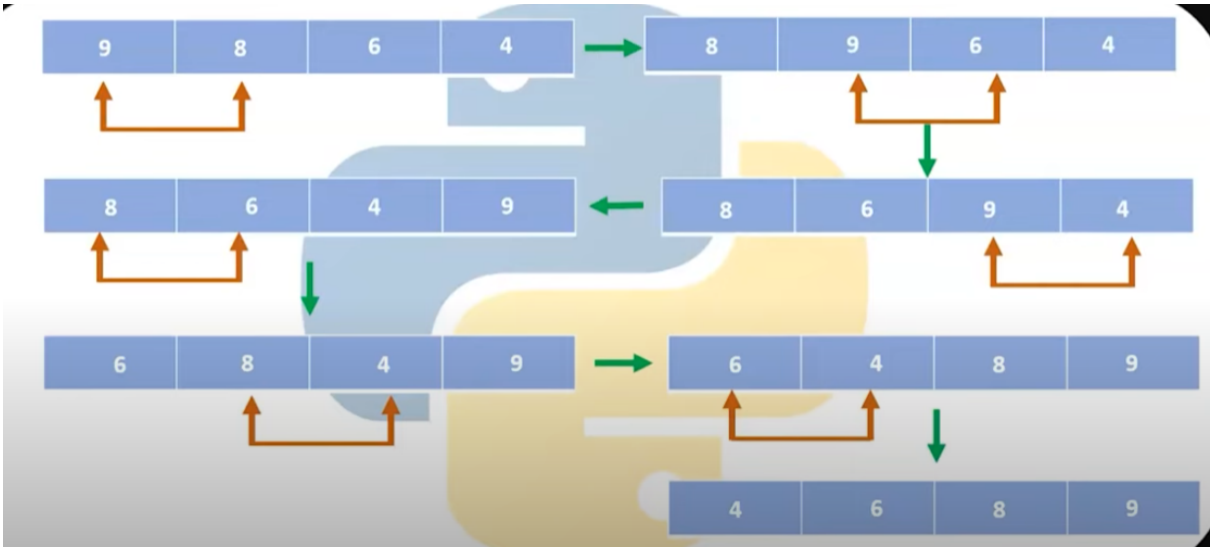
Big O notation

Bubble Sort

Neste algoritmo, o primeiro elemento é comparado com o elemento adjacente

Se o elemento adjacente é menor que o primeiro elemento, os elementos são trocados de posição.

Este algorismo então, compara o segundo elemento com o próximo elemento e o processo continua até achar o maior elemento e colocá-lo na ultima posição da direita até organizar de maneira ascendente.

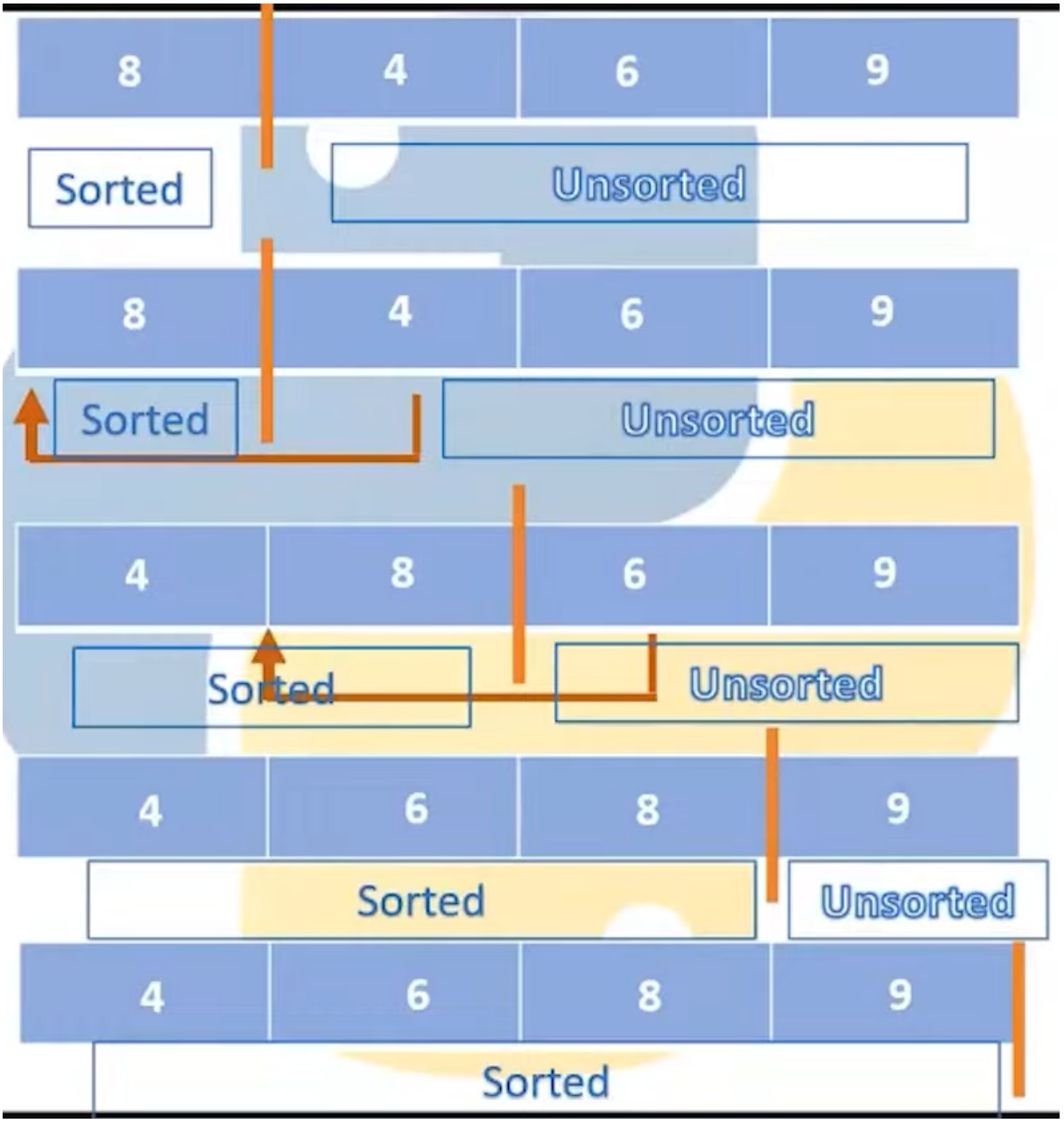



```
def bubbleSort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            print("j", j)
            print("j+1", j+1)
            print(arr)
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j+1], arr[j]
                print(arr)


elements = [9,8,6,4]
bubbleSort(elements)
```

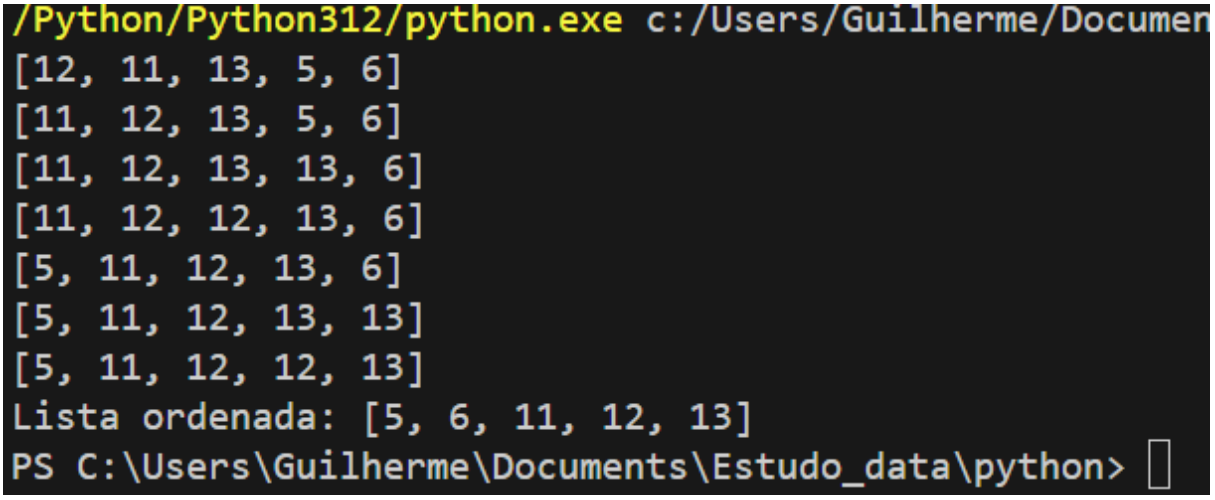
```
PS C:\Users\Guilherme\Documents\Estudo_data\python> & C:/Users/Guilherme/AppData/Local/Programs
/Python/Python312/python.exe c:/Users/Guilherme/Documents/Estudo_data/python/bubbleSort.py
[9, 8, 6, 4]
[8, 9, 6, 4]
[8, 9, 6, 4]
[8, 6, 9, 4]
[8, 6, 9, 4]
[8, 6, 4, 9]
[8, 6, 4, 9]
[6, 8, 4, 9]
[6, 8, 4, 9]
[6, 4, 8, 9]
[6, 4, 8, 9]
[4, 6, 8, 9]
PS C:\Users\Guilherme\Documents\Estudo_data\python>
```

Insertion Sort

Os elementos são percorridos um por um, e em cada passo, o elemento atual é comparado com os elementos anteriores. Se o elemento atual for menor do que o elemento anterior, eles são trocados até que a sequência esteja ordenada.



```python
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1

        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key


lista = [12, 11, 13, 5, 6]
insertion_sort(lista)
print("Lista ordenada:", lista)
```

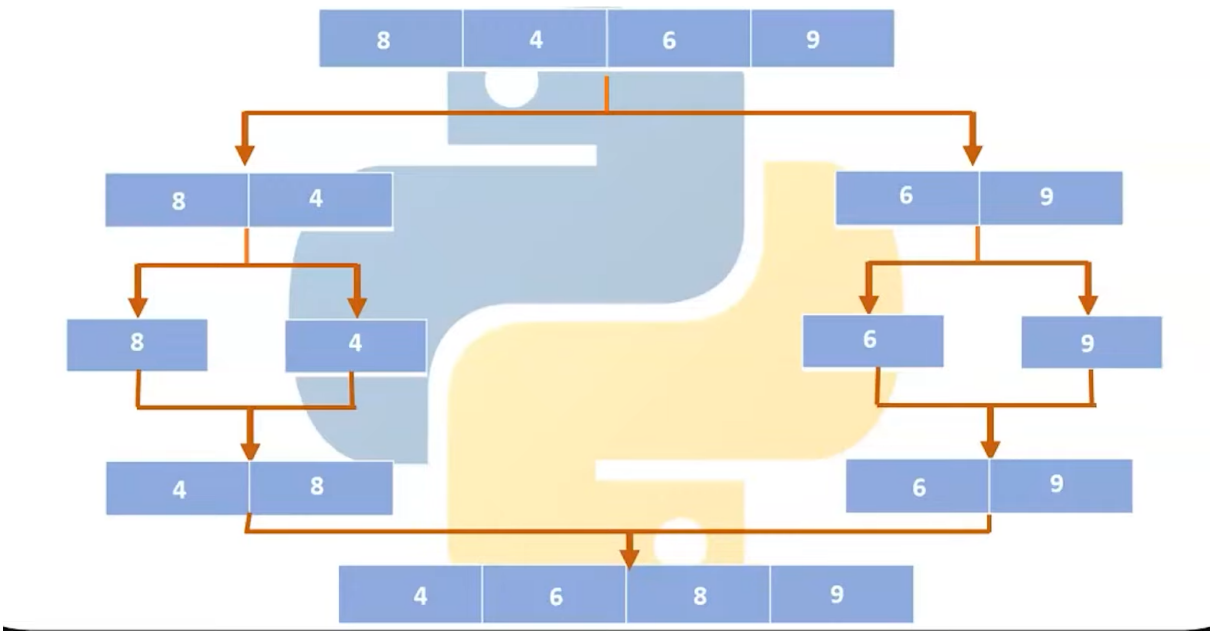


Merge Sort

O Merge Sort é um algoritmo de ordenação baseado no conceito de dividir para conquistar. Ele divide a lista não ordenada em N sublistas, cada uma com um único elemento, e então mescla essas sublistas em pares, repetindo o processo até que toda a lista esteja ordenada.

A ideia principal do Merge Sort é dividir a lista pela metade, ordenar cada metade recursivamente e, em seguida, combinar as duas metades ordenadas

para obter a lista final ordenada.



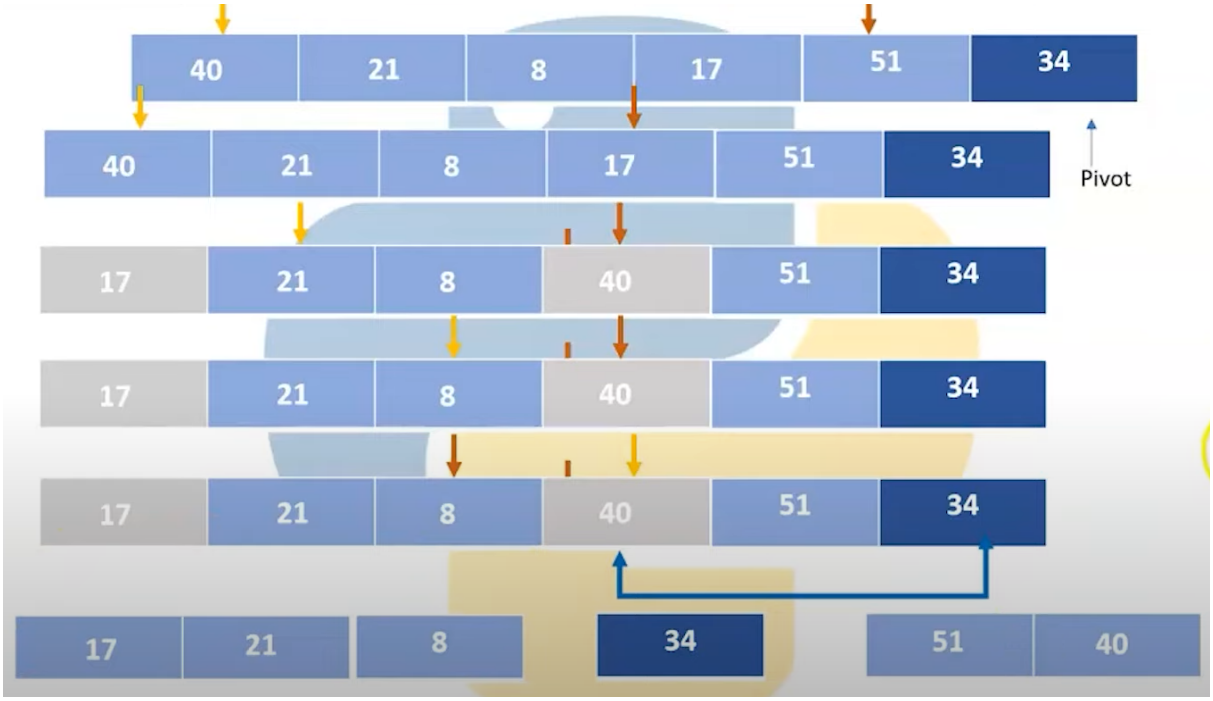

```
def merge_sort(arr):
    if len(arr) > 1:
        meio = len(arr) // 2
        esquerda = arr[:meio]
        direita = arr[meio:]
        print("esquerda: ", esquerda,"Direita: ", direita)
        merge_sort(esquerda)
        merge_sort(direita)
        i = j = k = 0

        while i < len(esquerda) and j < len(direita):
            if esquerda[i] < direita[j]:
                arr[k] = esquerda[i]
                i += 1
            else:
                arr[k] = direita[j]
                j += 1
            k += 1

        while i < len(esquerda):
```

```
            arr[k] = esquerda[i]
            i += 1
            k += 1

        while j < len(direita):
            arr[k] = direita[j]
            j += 1
            k += 1
            print("Resultado parcial: ",arr)


lista = [12, 11, 1, 13, 5, 20, 6, 7]
merge_sort(lista)
print("Lista ordenada:", lista)
```

```
/Python/Python312/python.exe c:/Users/Guilherme/Documents/Estudo_data/python/mergeSort.py
esquerda:  [12, 11, 1, 13] Direita:  [5, 20, 6, 7]
esquerda:  [12, 11] Direita:  [1, 13]
esquerda:  [12] Direita:  [11]
esquerda:  [1] Direita:  [13]
Resultado parcial:  [1, 13]
Resultado parcial:  [1, 11, 12, 13]
esquerda:  [5, 20] Direita:  [6, 7]
esquerda:  [5] Direita:  [20]
Resultado parcial:  [5, 20]
esquerda:  [6] Direita:  [7]
Resultado parcial:  [6, 7]
Resultado parcial:  [1, 5, 6, 7, 11, 12, 13, 20]
Lista ordenada: [1, 5, 6, 7, 11, 12, 13, 20]
PS C:\Users\Guilherme\Documents\Estudo_data\python>
```

Quick Sort

O Quick Sort é outro algoritmo de ordenação baseado no paradigma de "dividir para conquistar". Ele seleciona um elemento como pivô e organiza os outros elementos em torno dele, de modo que os elementos menores fiquem à esquerda e os elementos maiores fiquem à direita. Esse processo é aplicado recursivamente às sublistas resultantes até que a lista inteira esteja ordenada.



O pivô pode ser o numero mais a esquerda ou mais a direita

```
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    else:
        pivo = arr[0]
        print('pivo', pivo)
        menores = [x for x in arr[1:] if x <= pivo]
        print(menores)
        maiores = [x for x in arr[1:] if x > pivo]
        print(maiores)
        return quick_sort(menores) + [pivo] + quick_sort(maio

# Exemplo de uso:
lista = [12, 11, 13, 5, 6, 7,1,20,19,25,23]
lista_ordenada = quick_sort(lista)
print("Lista ordenada:", lista_ordenada)
```

```
/Python/Python312/python.exe c:/Users/Guilherme/Documents/Estudo_data/python/quickSort.py
pivo 12
[11, 5, 6, 7, 1]
[13, 20, 19, 25, 23]
pivo 11
[5, 6, 7, 1]

pivo 5
[1]
[6, 7]
pivo 6

[7]
pivo 13

[20, 19, 25, 23]
pivo 20
[19]
[25, 23]
pivo 25
[23]

Lista ordenada: [1, 5, 6, 7, 11, 12, 13, 19, 20, 23, 25]
```

Neste exemplo, a função `quick_sort` recebe uma lista como entrada e ordena os elementos usando o algoritmo Quick Sort. A função verifica se a lista tem tamanho 1 ou menos (caso base da recursão) e retorna a lista se for o caso. Caso contrário, um elemento é escolhido como pivô (neste caso, o primeiro elemento da lista). Os elementos menores que o pivô são colocados em uma lista chamada `menores` e os elementos maiores em uma lista chamada `maiores`. A função é chamada recursivamente para ordenar as sublistas menores e maiores, e o resultado final é a concatenação das sublistas ordenadas com o pivô no meio.

> *OBS: Em resumo, a chamada recursiva é uma maneira de quebrar um problema maior em problemas menores e resolver cada subproblema de maneira semelhante até chegar a uma solução geral. No contexto do Quick Sort, as chamadas recursivas são responsáveis por ordenar as sublistas menores resultantes da divisão inicial.*