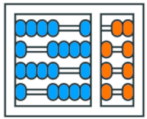


Instituto de Computação

UNIVERSIDADE ESTADUAL DE CAMPINAS

UNICAMP

MC102 - Aula 24

Algoritmos de Ordenação Recursivos - QuickSort

Algoritmos e Programação de Computadores

Turmas

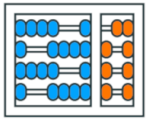
OVXZ

Prof. Lise R. R. Navarrete

lrommel@ic.unicamp.br

Quinta-feira, 23 de junho de 2022

19:00h - 21:00h (CB06)



Instituto de Computação

UNIVERSIDADE ESTADUAL DE CAMPINAS

UNICAMP

MC102 – Algoritmos e Programação de Computadores

Turmas

OVXZ

https://ic.unicamp.br/~mc102/

Site da Coordenação de MC102

Aulas teóricas:

Terça-feira, 21:00h - 23:00h (CB06)

Quinta-feira, 19:00h - 21:00h (CB06)

Conteúdo

- Quicksort
  - Partition
  - Partition com Listas Auxiliares
  - Partition "in place"
  - Partition no quick_sort
  - Análise de complexidade do QuikSort
- Tempo de Execução

Quicksort

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

- O Quicksort foi desenvolvido por Charles A. R. Hoare em 1959.
- O algoritmo Quicksort é baseado em uma operação de particionamento (`partition`) que, com base num elemento pivô, divide a lista em duas partes:
  - Valores menores que o pivô são colocados antes do pivô na lista, enquanto valores maiores são colocados depois.
- O algoritmo pode ser construído a partir dos seguintes passos:
  - Divisão: a lista é dividida em duas partes, usando a função `partition`.
  - Conquista: cada parte é ordenada recursivamente.
  - Combinação: nada precisa ser feito, já que os números menores que o pivô estão antes do pivô (e ordenados), enquanto os maiores estão depois do pivô (e também ordenados).

https://colab.research.google.com/

```
[12] def quick_sort(lista, inicio, fim):
       if fim - inicio > 1:
         # divisão:
         pivo = partition(lista, inicio, fim)
         # conquista
         quick_sort(lista, inicio, pivo)
         quick_sort(lista, pivo+1, fim)
         # combinação : nada a fazer
```

```
[13] lista = [45, 53, 13, 25, 89, 75, 46, 32, 20, 11]
     n = len(lista)
     quick_sort(lista, 0, n)
```

```
print(lista)
```

```
[11, 13, 20, 25, 32, 45, 46, 53, 75, 89]
```

Quicksort

Partition

45	53	13	25	89	75	46	32	20	11

45	53	13	25	89	75	46	32	20	11

escolher pivô

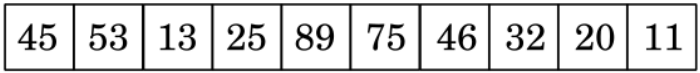
45	53	13	25	89	75	46	32	20	11

escolher pivô

45	53	13	25	89	75	46	32	20	11

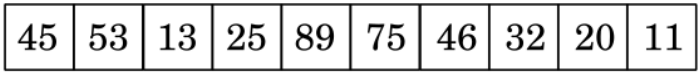
escolher pivô

45

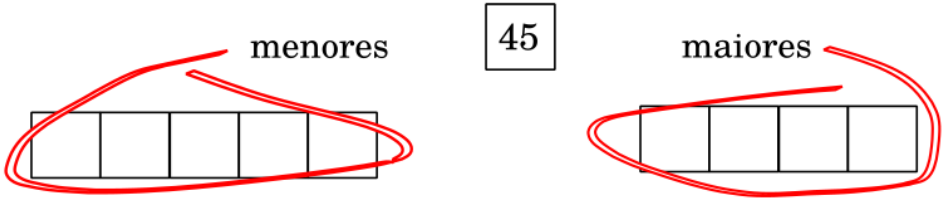


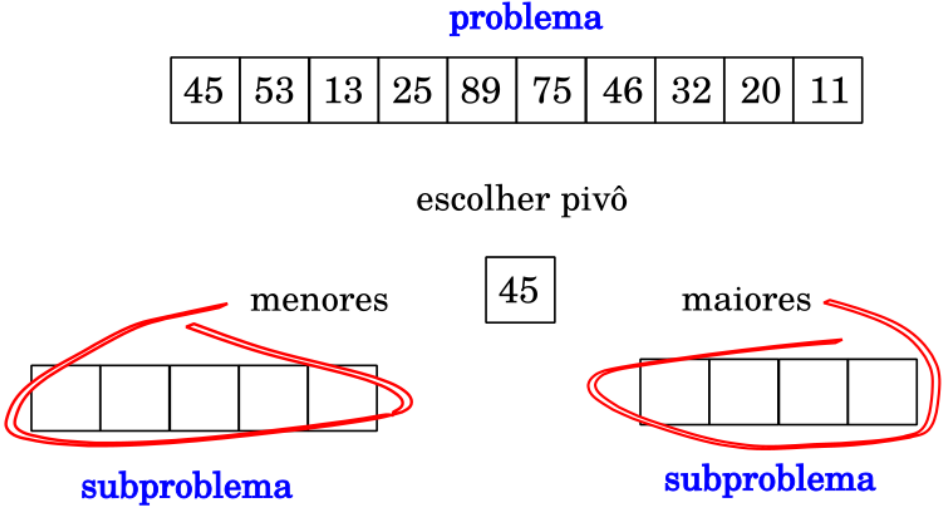
escolher pivô

menores 45 maiores



escolher pivô




problema
45 53 13 25 89 75 46 32 20 11
escolher pivô
45
menores
maiores
subproblema
subproblema

Quicksort

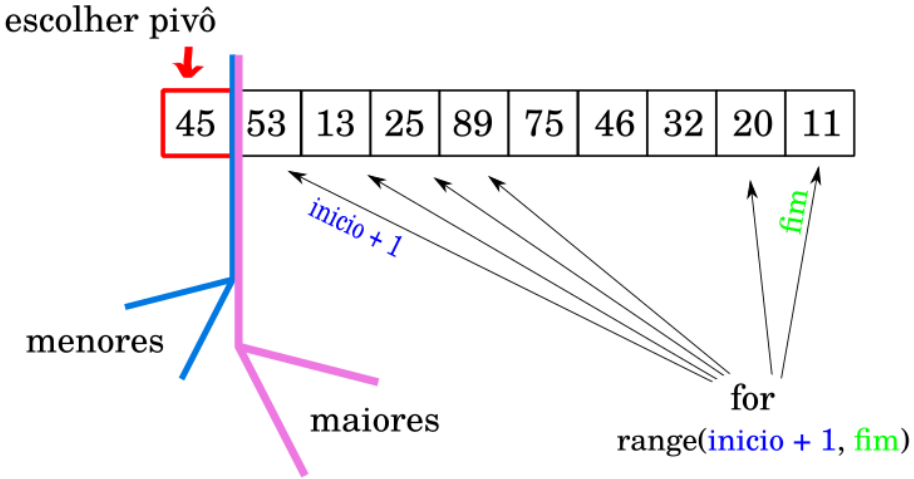
Partition com Listas Auxiliares

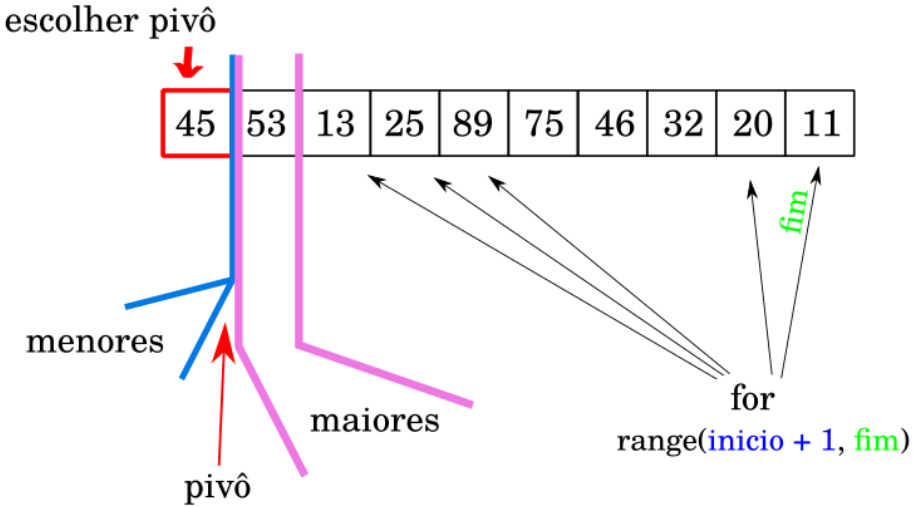
https://ic.unicamp.br/~mc102/aulas/aula13.pdf

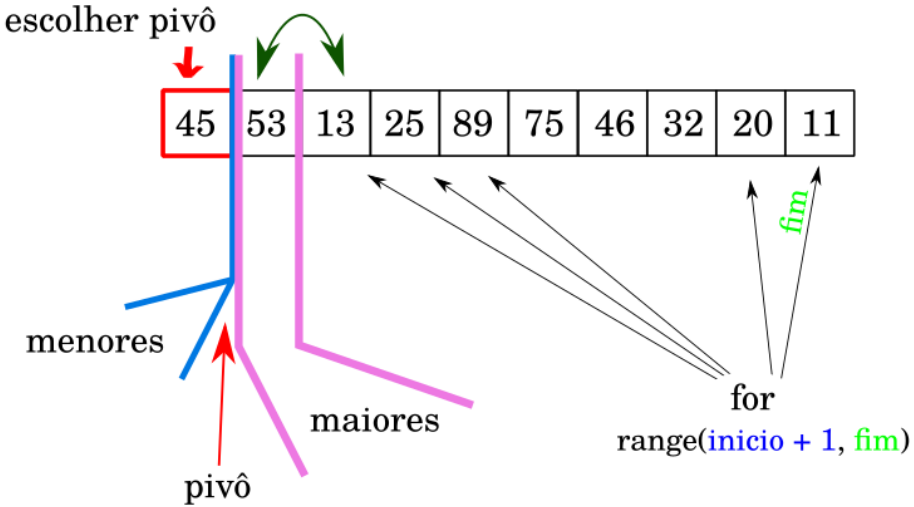
```
1  def partition(lista, inicio, fim):
2    pivo = lista[inicio]
3    menores = []
4    maiores = []
5
6    for k in range(inicio + 1, fim):
7      if lista[k] <= pivo:
8        menores.append(lista[k])
9      else:
10       maiores.append(lista[k])
11
12   lista[inicio:fim] = menores + [pivo] + maiores
13
14   return inicio + len(menores)
```

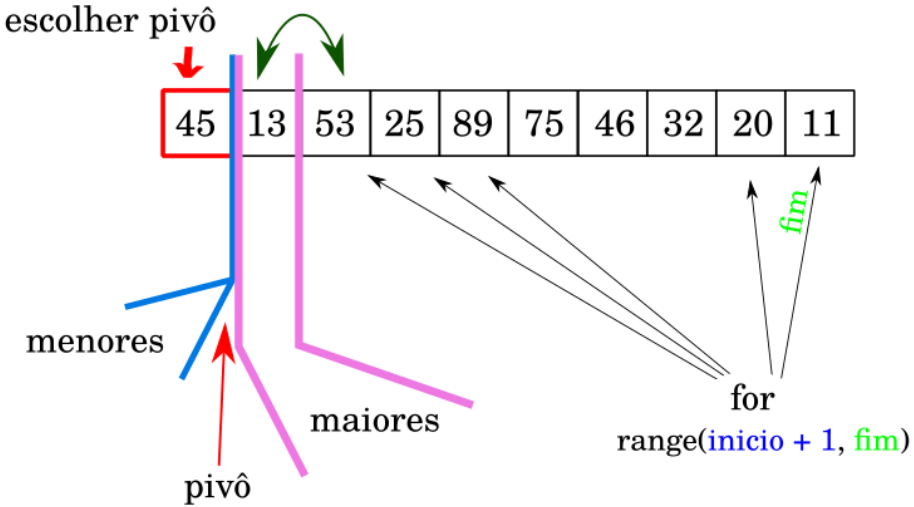
Quicksort

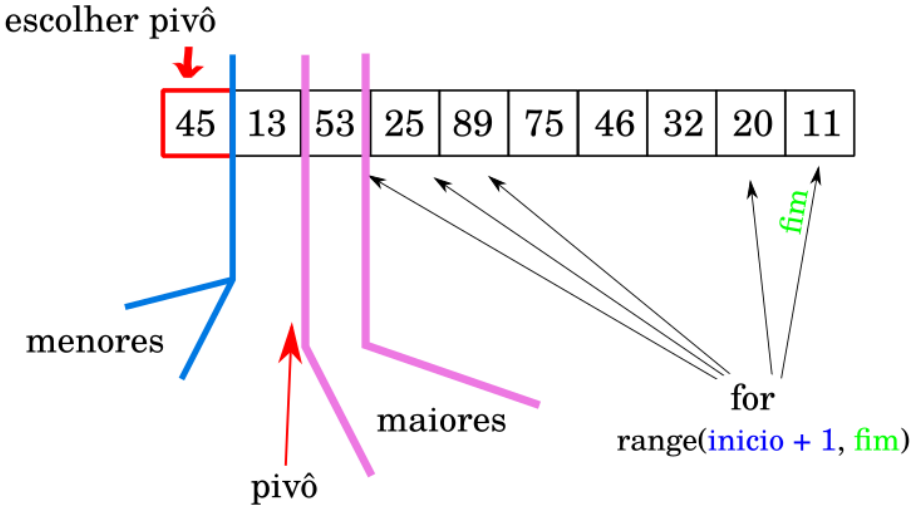
Partition "in place"

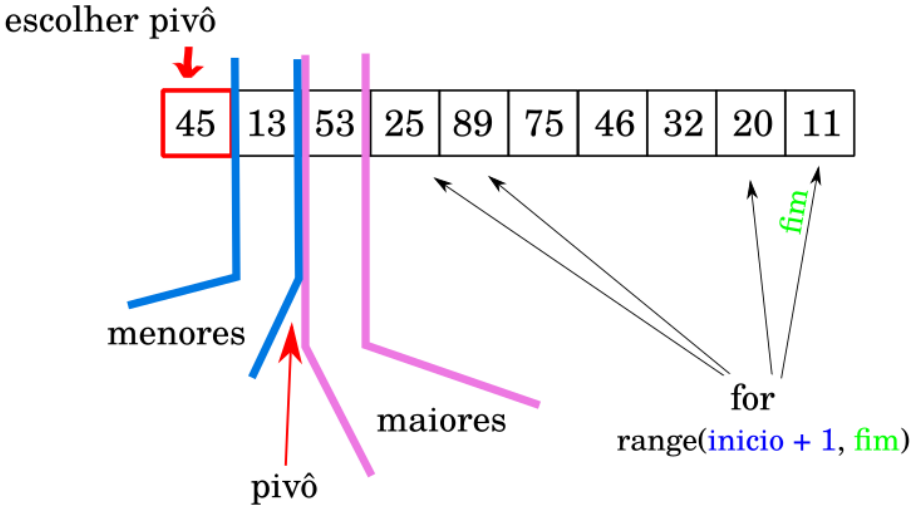

escolher pivô
45
53
13
25
89
75
46
32
20
11
inicio + 1
fim
menores
maiores
for
range(inicio + 1, fim)

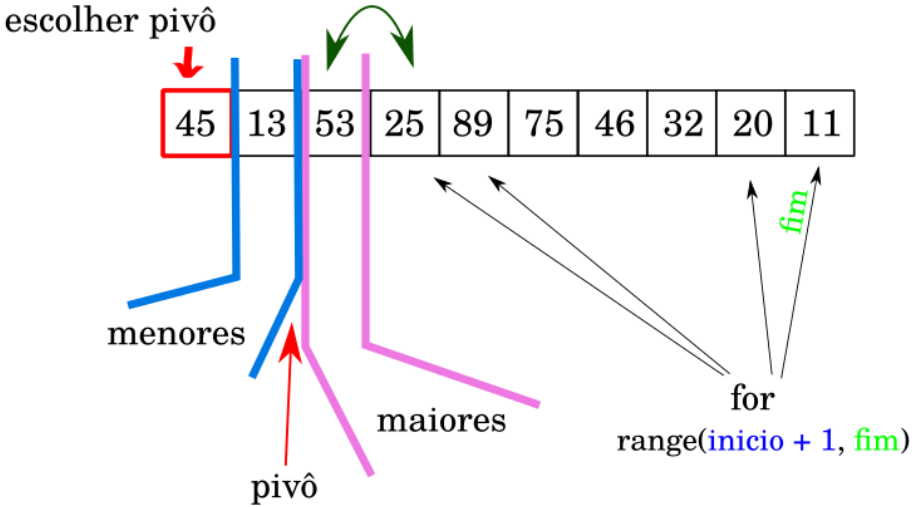

escolher pivô
45
53
13
25
89
75
46
32
20
11
fim
menores
maiores
pivô
for
range(inicio + 1, fim)

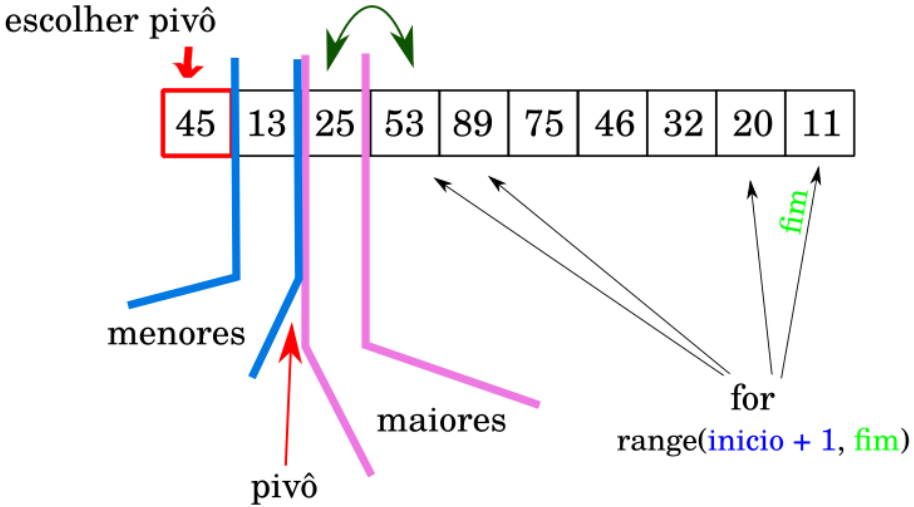

escolher pivô
45
53
13
25
89
75
46
32
20
11
fim
menores
maiores
pivô
for
range(inicio + 1, fim)

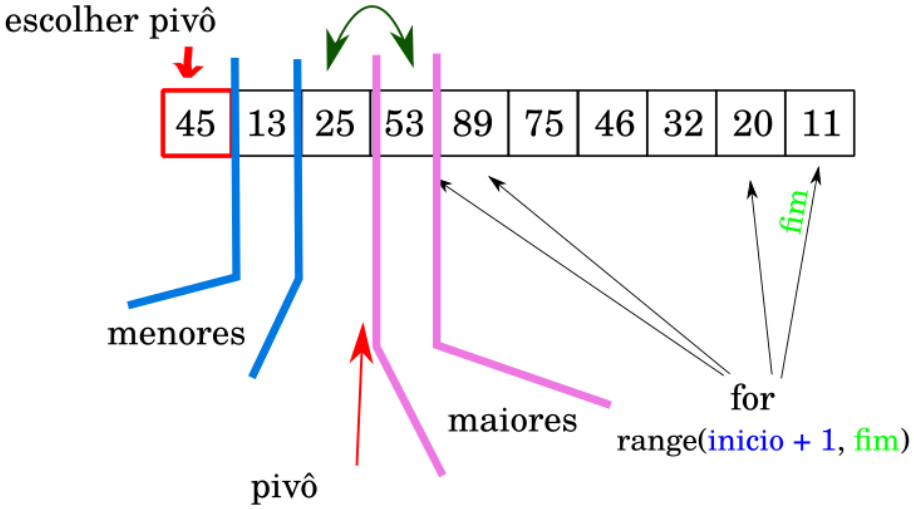

escolher pivô
45
13
53
25
89
75
46
32
20
11
fim
menores
maiores
pivô
for
range(inicio + 1, fim)

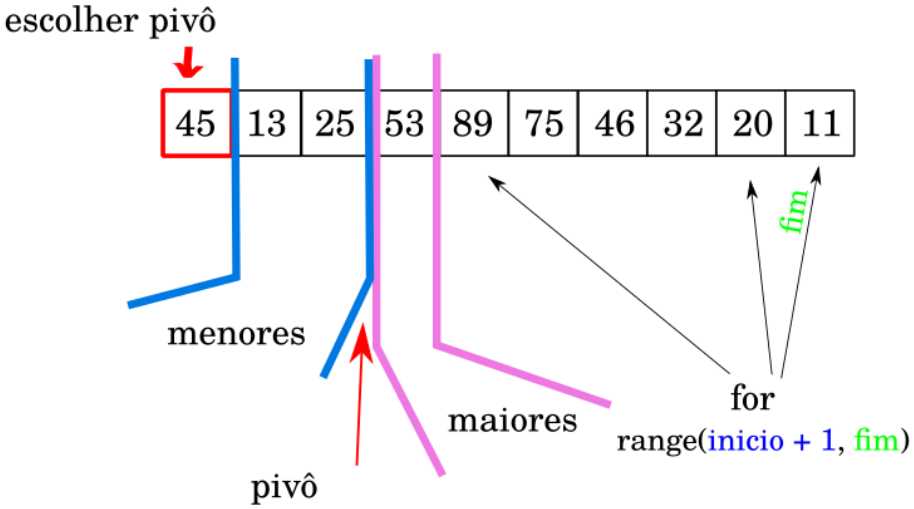

escolher pivô
45
13
53
25
89
75
46
32
20
11
fim
menores
maiores
pivô
for
range(inicio + 1, fim)


escolher pivô
45
13
53
25
89
75
46
32
20
11
fim
menores
maiores
pivô
for
range(inicio + 1, fim)


escolher pivô
45
13
53
25
89
75
46
32
20
11
fim
menores
maiores
pivô
for
range(inicio + 1, fim)


escolher pivô
45
13
25
53
89
75
46
32
20
11
fim
menores
maiores
pivô
for
range(inicio + 1, fim)


escolher pivô
45
13
25
53
89
75
46
32
20
11
fim
menores
maiores
for
range(inicio + 1, fim)
pivô


escolher pivô
45
13
25
53
89
75
46
32
20
11
fim
menores
maiores
for
range(inicio + 1, fim)
pivô

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45]  53   13   25   89   75   46   32   20  [11]
inicio                                      fim-1
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45]  53   13   25   89   75   46   32   20   11
 j
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45] [53]  13   25   89   75   46   32   20   11
 j    i
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45] [53   13]  25   89   75   46   32   20   11
 j         i
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45   53] [13]  25   89   75   46   32   20   11
      j    i
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45   13] [53]  25   89   75   46   32   20   11
      j    i
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45   13] [53   25]  89   75   46   32   20   11
      j         i
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45   13   53] [25]  89   75   46   32   20   11
          j    i
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45   13   25] [53]  89   75   46   32   20   11
          j    i
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45   13   25] [53   89]  75   46   32   20   11
          j         i
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45   13   25] [53   89   75]  46   32   20   11
          j              i
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45   13   25] [53   89   75   46]  32   20   11
          j                  i
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45   13   25] [53   89   75   46   32]  20   11
          j                        i
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45   13   25   53] [89   75   46   32]  20   11
          j                    i
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45   13   25   32] [89   75   46   53]  20   11
                j                    i
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45   13   25   32] [89   75   46   53   20]  11
          j                        i
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45   13   25   32   89] [75   46   53   20]  11
                  j                   i
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45   13   25   32   20] [75   46   53   89]  11
                  j                   i
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45   13   25   32   20] [75   46   53   89   11]
                  j                        i
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45   13   25   32   20   75] [46   53   89   11]
                     j                    i
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45   13   25   32   20   11] [46   53   89   75]
                      j                    i
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[45   13   25   32   20   11] [46   53   89   75]
inicio                   j
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[11   13   25   32   20   45] [46   53   89   75]
inicio                    j
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[11   13   25   32   20] [45] [46   53   89   75]
                          j
```

```
1  def partition(lista, inicio, fim):
2    j = inicio
3
4    for i in range(inicio + 1, fim):
5      if lista[i] <= lista[inicio]:
6        j = j + 1
7        (lista[i], lista[j]) = (lista[j], lista[i])
8
9    (lista[inicio], lista[j]) = (lista[j], lista[inicio])
10
11   return j
```

Partition no quick_sort

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

11 13 25 32 20 [45] 46 53 89 75
pivo

```
1 def partition(lista, inicio, fim):
2   ...
3
4 def quick_sort(lista, inicio, fim):
5   if fim - inicio > 1:
6     pivo = partition(lista, inicio, fim)
7     quick_sort(lista, inicio, pivo)
8     quick_sort(lista, pivo + 1, fim)
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[11   13   25   32   20] [45]  46   53   89   75
                         pivo
```

```python
def partition(lista, inicio, fim):
  ...

def quick_sort(lista, inicio, fim):
  if fim - inicio > 1:
    pivo = partition(lista, inicio, fim)
    quick_sort(lista, inicio, pivo)
    quick_sort(lista, pivo + 1, fim)
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[11   13   20   25   32] [45]  46   53   89   75
       quicksort          pivo
```

```
1  def partition(lista, inicio, fim):
2    ...
3
4  def quick_sort(lista, inicio, fim):
5    if fim - inicio > 1:
6      pivo = partition(lista, inicio, fim)
7      quick_sort(lista, inicio, pivo)
8      quick_sort(lista, pivo + 1, fim)
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[11   13   20   25   32] [45] [46   53   89   75]
                          pivo
```

```python
def partition(lista, inicio, fim):
  ...

def quick_sort(lista, inicio, fim):
  if fim - inicio > 1:
    pivo = partition(lista, inicio, fim)
    quick_sort(lista, inicio, pivo)
    quick_sort(lista, pivo + 1, fim)
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[11   13   20   25   32] [45] [46   53   75   89]
                         pivo      quicksort
```

```python
def partition(lista, inicio, fim):
  ...

def quick_sort(lista, inicio, fim):
  if fim - inicio > 1:
    pivo = partition(lista, inicio, fim)
    quick_sort(lista, inicio, pivo)
    quick_sort(lista, pivo + 1, fim)
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
[11   13   20   25   32] [45] [46   53   75   89]
                          pivo
```

```python
def partition(lista, inicio, fim):
  ...

def quick_sort(lista, inicio, fim):
  if fim - inicio > 1:
    pivo = partition(lista, inicio, fim)
    quick_sort(lista, inicio, pivo)
    quick_sort(lista, pivo + 1, fim)
```

Partition no quick_sort

Análise de complexidade do QuikSort

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

- Melhor caso: ocorre quando o `partition` sempre divide a lista em duas partes de tamanhos aproximadamente iguais.
- A seguinte recorrência define o tempo de execução do Quicksort no melhor caso:

$$T(1) = c_1$$
$$T(n) = T(\lceil n/2 \rceil - 1) + T(\lfloor n/2 \rfloor) + P(n) + c_2$$

- É fácil ver que $P(n)$, o tempo de execução da função `partition`, é proporcional à função $f(n) = n$.
- É possível mostrar que $T(n)$, o tempo de execução do Quicksort no melhor caso, é proporcional à função $f(n) = n \log n$.

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

- Pior caso: ocorre quando o `partition` sempre divide a lista em duas partes de tamanhos muito diferentes.
- A seguinte recorrência define o tempo de execução do Quicksort no pior caso:

$$T(1) = c_1$$
$$T(n) = T(n-1) + P(n) + c_2$$

- Como sabemos, $P(n)$, o tempo de execução da função `partition`, é proporcional à função $f(n) = n$.
- É possível mostrar que $T(n)$, o tempo de execução do Quicksort no pior caso, é proporcional à função $f(n) = n^2$.

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

- Caso médio: a probabilidade de uma partição de um tamanho qualquer ocorrer é igual a $1/n$.
- A seguinte recorrência define o tempo de execução do Quicksort no caso médio:

$$T(1) = c_1$$
$$T(n) = \frac{1}{n}\sum_{i=0}^{n-1}[T(i) + T(n-1-i)] + P(n) + c_2$$

- Como sabemos, $P(n)$, o tempo de execução da função `partition`, é proporcional à função $f(n) = n$.
- É possível mostrar que $T(n)$, o tempo de execução do Quicksort no caso médio, é proporcional à função $f(n) = n \log n$.

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

- Dada uma lista aleatória qualquer, é extremamente raro o Quicksort se comportar como no seu pior caso.
- No entanto, o Quicksort, devido à escolha do primeiro elemento da lista como pivô, apresenta seu pior comportamento quando recebe como entrada um dos casos mais simples possíveis para qualquer algoritmo de ordenação: uma lista já ordenada.
- Uma forma de contornar este caso (lista ordenada) e evitar partes de tamanho zero é utilizar como pivô a mediana de três elementos da lista: o primeiro, o do meio e o último.
- Uma outra alternativa bastante utilizada é definir o pivô como um elemento da lista escolhido de forma aleatória.
- Uma vantagem do Quicksort em relação ao Merge Sort é em relação ao uso de memória auxiliar: o Quicksort não usa uma lista auxiliar, consumindo apenas o espaço para armazenar as variáveis locais na pilha de recursão.

Tempo de Execução

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

- O tempo de execução de um código pode ser verificado utilizando a biblioteca `time`.
- Essa biblioteca possui funções bastante úteis.
- Função `time`:
  - Retorna um `float` com o tempo em segundos passados desde um marco de tempo padrão.
  - Para sistemas Unix, esse marco de tempo é 01/01/1970, 00:00:00 UTC.
- Função `ctime`:
  - Recebe como parâmetro o tempo passado em segundos desde um marco de tempo padrão.
  - Retorna a data e o horário correspondente em formato `String`.
- Função `sleep`:
  - Suspende a execução de um programa pelo número de segundos especificado.

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

- Para utilizar a biblioteca é necessário que a mesma seja importada no código.

```
1 import time
```

- Exemplo de uso das funções `time` e `ctime`.

```
1 import time
2 tempo = time.time()
3 print(tempo)
4 # 1581932985.0103931
5 tempo_str = time.ctime(tempo)
6 print(tempo_str)
7 # Mon Feb 17 09:49:45 2020
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

- Computando o tempo de execução de um trecho de código.

```
import time
inicio = time.time() # tempo de início
time.sleep(3) # pausa a execução por 3 segundos
for i in range(100):
  time.sleep(0.1) # pausa a execução por 0.1 segundo
fim = time.time() # tempo final
print(fim - inicio) # tempo total (em segundos)
# 13.029353380203247
```

- Será computado o tempo gasto para executar o trecho de código entre as declarações das variáveis `inicio` e `fim`.
- Dessa forma, é possível mensurar o tempo de execução gasto por um programa, uma função ou um trecho específico de código.

Perguntas ....

Referências

- Zanoni Dias, MC102, Algoritmos e Programação de Computadores, IC/UNICAMP, 2021. https://ic.unicamp.br/~mc102/

  Aula Introdutória [slides] [vídeo]
  Primeira Aula de Laboratório [slides] [vídeo]
  Python Básico: Tipos, Variáveis, Operadores, Entrada e Saída [slides] [vídeo]
  Comandos Condicionais [slides] [vídeo]
  Comandos de Repetição [slides] [vídeo]
  Listas e Tuplas [slides] [vídeo]
  Strings [slides] [vídeo]
  Dicionários [slides] [vídeo]
  Funções [slides] [vídeo]
  Objetos Multidimensionais [slides] [vídeo]
  Algoritmos de Ordenação [slides] [vídeo]
  Algoritmos de Busca [slides] [vídeo]
  Recursão [slides] [vídeo]
  Algoritmos de Ordenação Recursivos [slides] [vídeo]
  Arquivos [slides] [vídeo]
  Expressões Regulares [slides] [vídeo]
  Execução de Testes no Google Cloud Shell [slides] [vídeo]
  Numpy [slides] [vídeo]
  Pandas [slides] [vídeo]

- Panda - Cursos de Computação em Python (IME -USP) https://panda.ime.usp.br/

  Como Pensar Como um Cientista da Computação https://panda.ime.usp.br/pensepy/static/pensepy/
  Aulas de Introdução à Computação em Python https://panda.ime.usp.br/aulasPython/static/aulasPython/

- Fabio Kon, Introdução à Ciência da Computação com Python http://bit.ly/FabioKon/
- Socratica, Python Programming Tutorials http://bit.ly/SocraticaPython/
- Google - online editor for cloud-native applications (Python programming) https://shell.cloud.google.com/
- w3schools - Python Tutorial https://www.w3schools.com/python/
- Outros, citados nos Slides.