



[www.python.pro.br](http://www.python.pro.br)

# Roteiro - QuickSort

Definição

Análise

Visualização

Implementação

Merge versus Quick

Exercício \*

# Definição

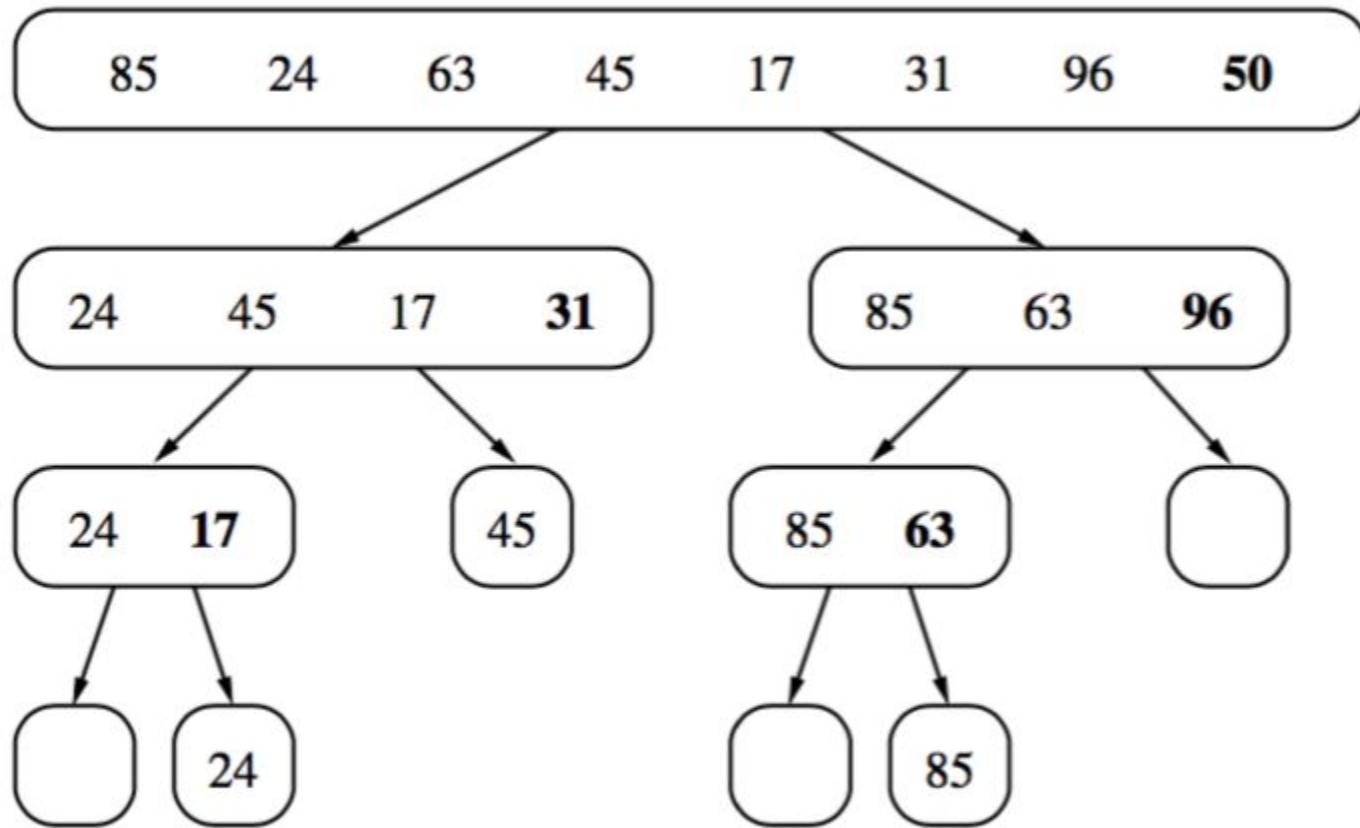
Solução óbvia quando  $\text{len}(\text{sequencia}) \leq 1$

Divisão: Escolher pivot. Criar duas listas. Uma com elementos menores que pivot e outra com maiores

Conquista: ordenar sublistas com `quick_sort`

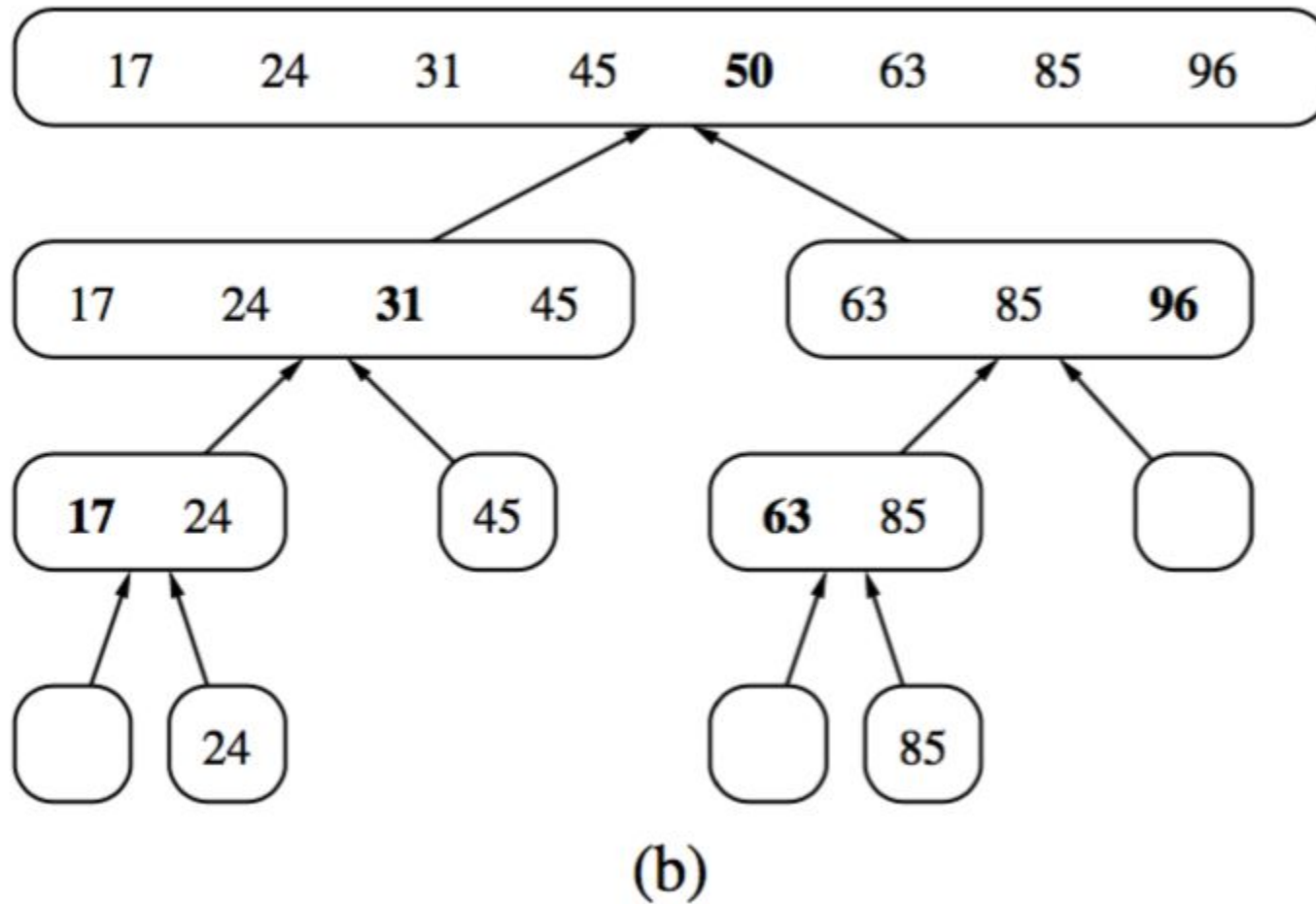
Combinação: concatenar listas ordenadas \*

# Análise



(a)

# Análise



# Análise - Caso Médio

Pivot separa itens na metade

Cada Nível Possui  $n$  elementos

Concatenação de sublistas será então  $O(n)$   
para cada nível

O número de níveis é igual ao teto de  $\log_2(n)$

Portanto complexidade é  $O(n \cdot \log(n))$

# Análise - Pior Caso

Lista ordenada e escolha de pivot como último elemento

Separação sempre será em um lista vazia e outra com  $n-1$  elementos

Nesse caso o algoritmo é  $O(n^2)$

Escolha de Pivot aleatório ou com mediana evita problema \*

# Visualiação

	 Insertion	 Selection	 Bubble	 Shell	 Merge	 Heap	 Quick	 Quick3
 Random								
 Nearly Sorted								
 Reversed								
 Few Unique								



# Implementação

Usar o código base de testes de ordenação  
Implementar o QuickSort recursivo:

[https://github.com/pythonprobr/estrutura-de-dados/blob/main/aula\\_06/test\\_ordenacao.py](https://github.com/pythonprobr/estrutura-de-dados/blob/main/aula_06/test_ordenacao.py)

# Merge versus Quick

Escolha de pivot randomico

Merge exige criação de muitas listas adicionais com grande footprint de memória

Por conta disso Quick costuma ser mais eficiente \*

# Exercício QuickSort in-place

Usar teste base

Implementar QuickSort interno (sem usar criação de sublistas) \*

# Obrigado

renzo@python.pro.br  
@renzoprobr

