Quick Sort

- Quick Sort is an efficient, comparison-based, divide-and-conquer sorting algorithm.
- It works by selecting a "pivot" element and partitioning the array into two sub-arrays:
  - Elements less than or equal to the pivot.
  - Elements greater than the pivot.
- It recursively applies the same process to the subarrays.
- In-Place: It sorts the array in place, meaning it requires only a small, constant amount of additional storage space.

Quick sort algorithm

- **Input**:
  - A list of elements to be sorted.
- **Output**:
  - A sorted list in ascending order.
- **Steps**:
  1. Choose a Pivot:
     - Select an element from the list as the pivot
  2. Partition the List:
     - Rearrange elements such that those less than the pivot are on the left, and those greater are on the right.
  3. Recursively Sort Subarrays:
     - Recursively apply the above steps to the sub-arrays of elements with smaller values and greater values.

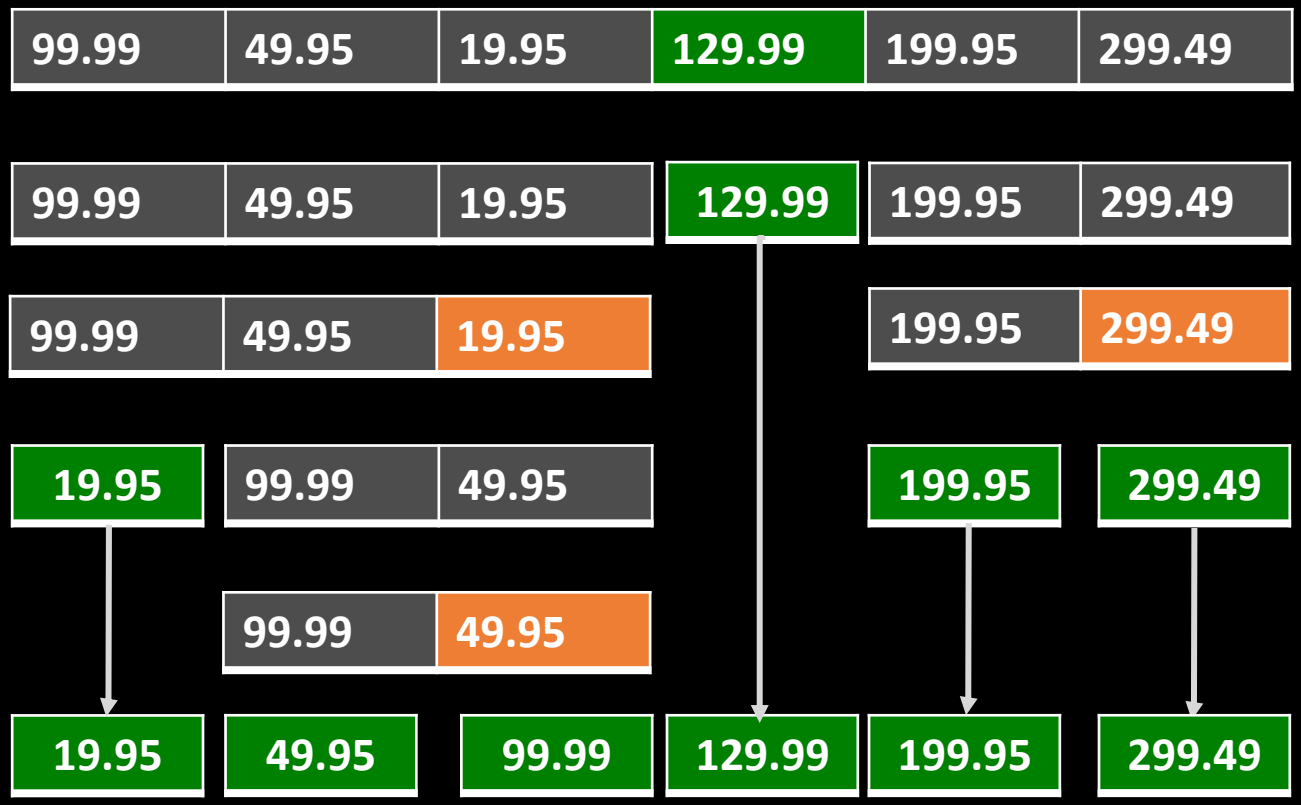
How quick sort works

- Select a pivot

- Put all the elements that are less than the pivot on its left side and elements that are greater than on its right side

- Recursively sort left and right of the pivot

How quick sort works



Partition process

- First call to partition:

low high

0	1	2	3	4	5
99.99	49.95	299.49	19.95	199.95	129.99

Partition process

- **First call to partition:**
  - Pivot = prices[high] = 129.99.
  - i = low -1 = -1.

low high

0	1	2	3	4	5
99.99	49.95	299.49	19.95	199.95	129.99

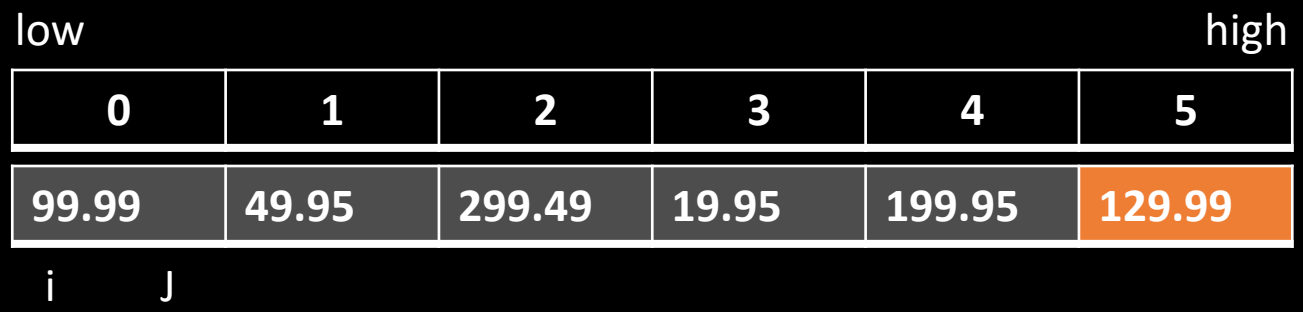
J

- **Iteration** 1:
- J = 0
- i = -1

Comparison:
99.99 < 129.99 (True)
i +=1

Partition process

- **First call to partition:**
  - Pivot = prices[high] = 129.99.
  - i = low -1 = -1.

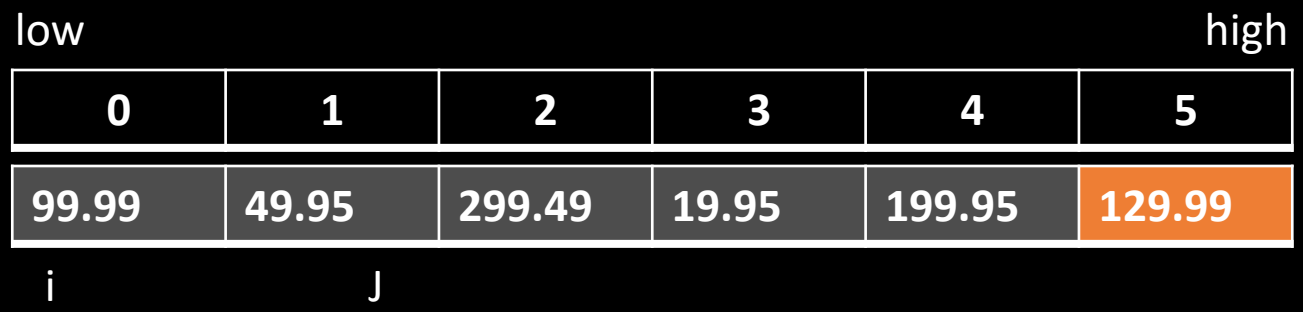


- **Iteration** 1:
- J = 0
- i = 0

Comparison:
99.99 < 129.99 (True)
i +=1

Partition process

- **First call to partition:**
  - Pivot = prices[high] = 129.99.
  - i = low -1 = -1.

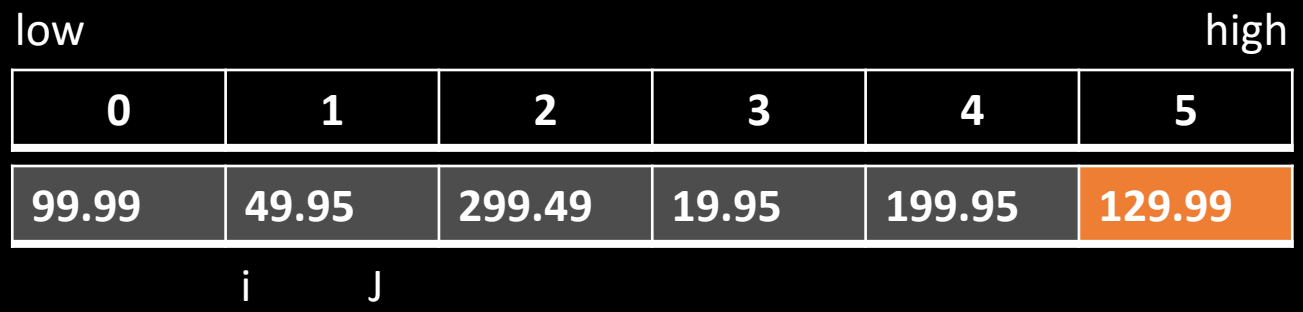


- **Iteration** 2:
- J = 1
- i = 0

Comparison:
49.95 < 129.99 (True)
i +=1

Partition process

- **First call to partition:**
  - Pivot = prices[high] = 129.99.
  - i = low -1 = -1.

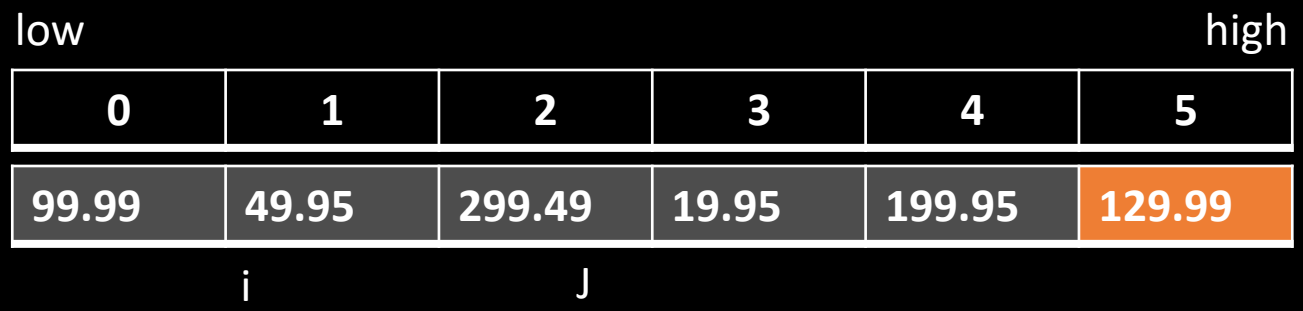


- **Iteration** 2:
- J = 1
- i = 1

Comparison:
49.95 < 129.99 (True)
i +=1

Partition process

- **First call to partition:**
  - Pivot = prices[high] = 129.99.
  - i = low -1 = -1.

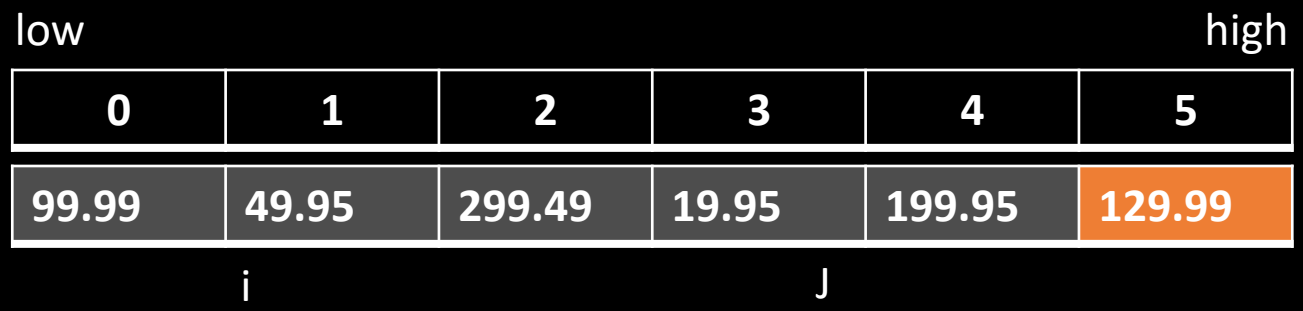


- **Iteration** 3:
- J = 2
- i = 1

Comparison:
299.49 < 129.99 (False)
i +=1

Partition process

- **First call to partition:**
  - Pivot = prices[high] = 129.99.
  - i = low -1 = -1.

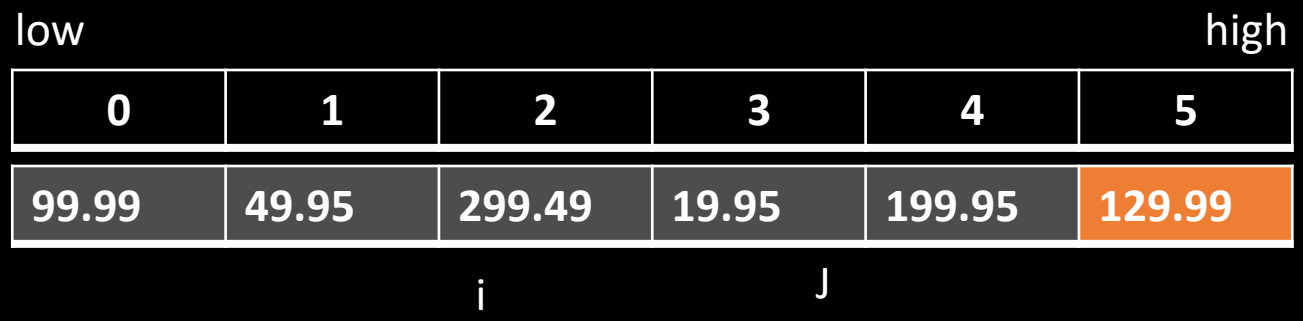


- **Iteration** 4:
- J = 3
- i = 1

Comparison:
19.95 < 129.99 (True)
i +=1

Partition process

- **First call to partition:**
  - Pivot = prices[high] = 129.99.
  - i = low -1 = -1.

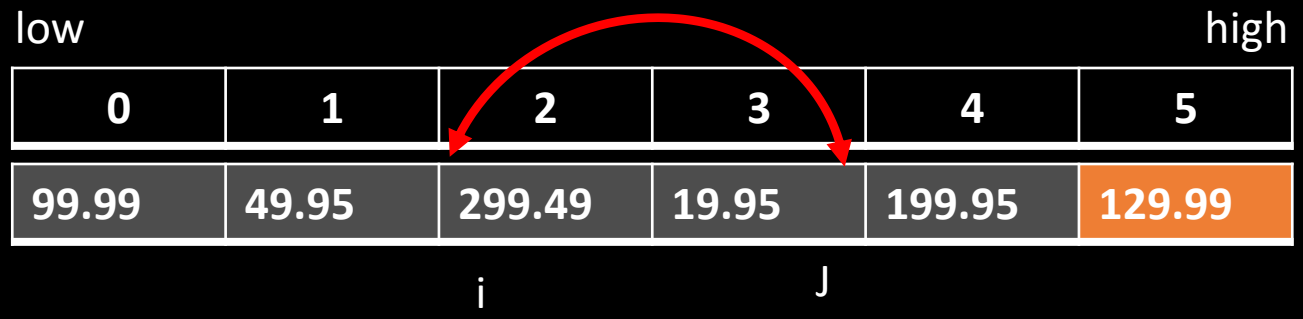


- **Iteration** 4:
- J = 3
- i = 2

Comparison:
19.95 < 129.99 (True)
i +=1

Partition process

- **First call to partition:**
  - Pivot = prices[high] = 129.99.
  - i = low -1 = -1.

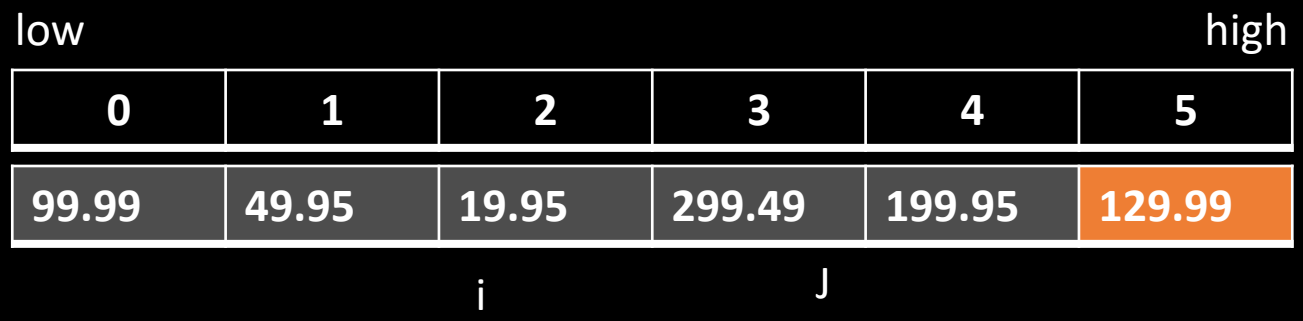


- **Iteration** 4:
- J = 3
- i = 2

Comparison:
19.95 < 129.99 (True)
i +=1

Partition process

- **First call to partition:**
  - Pivot = prices[high] = 129.99.
  - i = low -1 = -1.

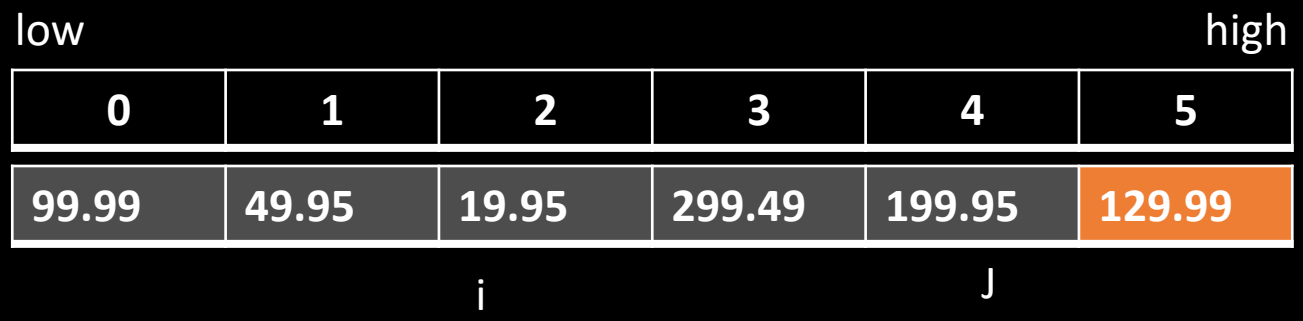


- **Iteration** 4:
- J = 3
- i = 2

Comparison:
19.95 < 129.99 (True)
i +=1

Partition process

- **First call to partition:**
  - Pivot = prices[high] = 129.99.
  - i = low -1 = -1.

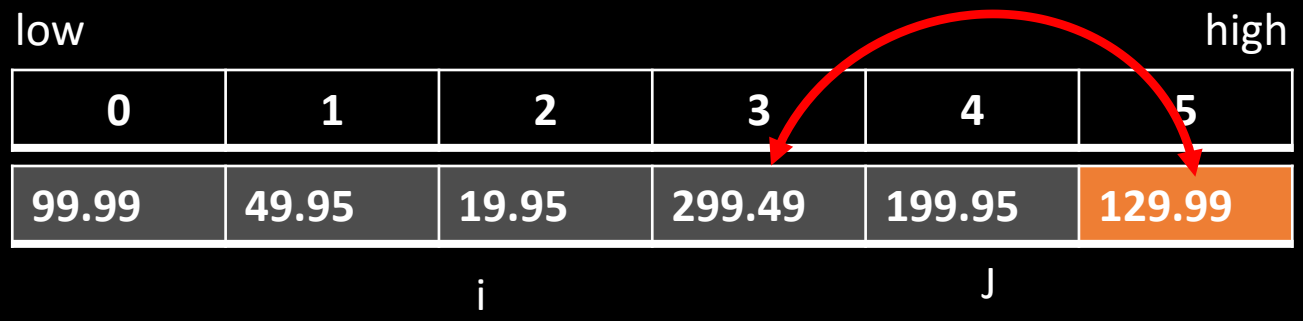


- **Iteration** 5:
- J = 4
- i = 2

Comparison:
199.95 < 129.99 (false)
i +=1

Partition process

- **First call to partition:**
  - Pivot = prices[high] = 129.99.
  - i = low -1 = -1.

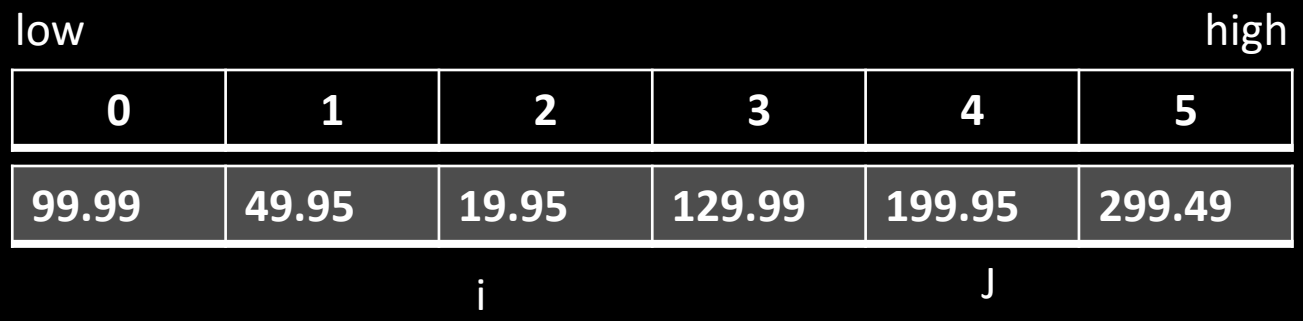


- **Final swap**:
- J = 4
- i = 2

final:
swap prices[i + 1]
with prices[high]

Partition process

- **First call to partition:**
  - Pivot = prices[high] = 129.99.
  - i = low -1 = -1.



- **Final swap**:
- J = 4
- i = 2

final:
swap prices[i + 1]
with prices[high]

Partition process

- Result from first partition:

low high

0	1	2	3	4	5
99.99	49.95	19.95	129.99	199.95	299.49

i J

- Final swap:
- J = 4
- i = 2

Return i+1