Answer the following quiz

- What is the time complexity of the selection sort algorithm in the worst case?

A. O(n)

B. O(n log n)

C. $O(n^2)$

D. $O(n^3)$

❖ *Answer in the comment section*

Merge Sort

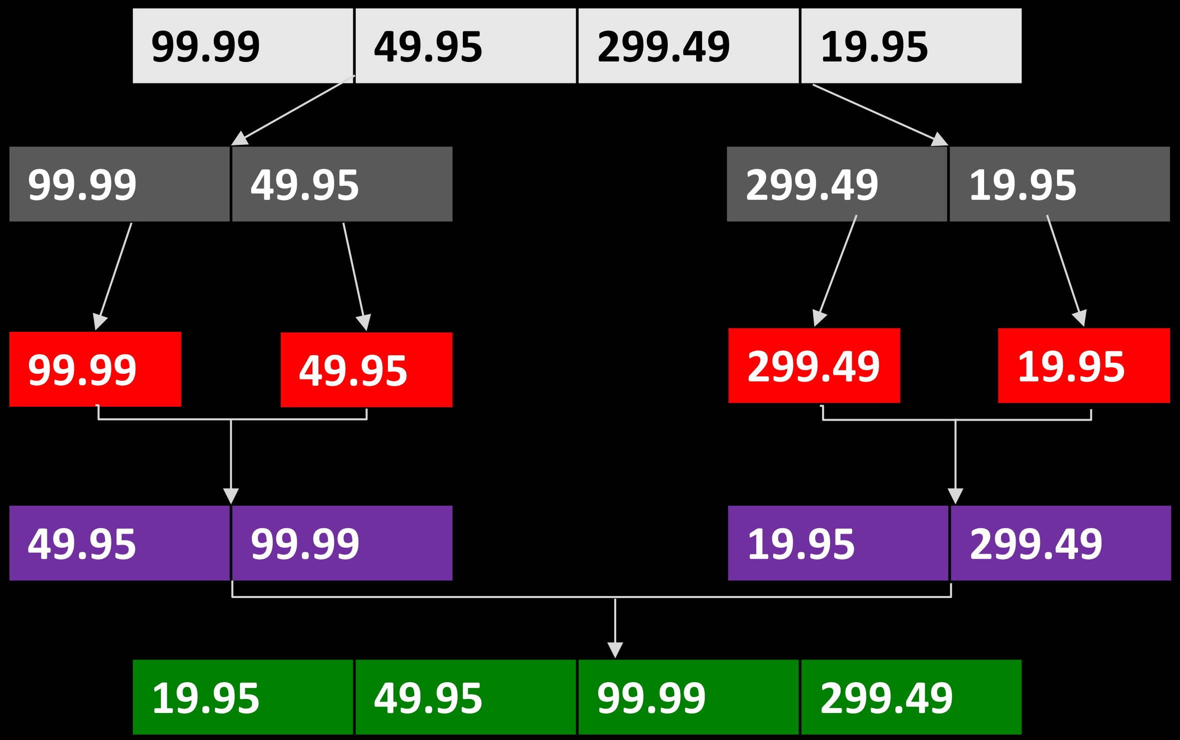
- A divide-and-conquer sorting algorithm.
- Recursively splits the array into halves, sorts each half, and merges them back together.
- Suitable for large datasets due to its guaranteed O(n log n) time complexity.

- Stable: Preserves the relative order of equal elements.
- Efficient for large datasets: Consistently performs well regardless of the initial order.
- Not in-place: Requires extra space for merging.

Merge Sort algorithm

- **Input**:
  - A list of elements to be sorted.
- **Output**:
  - A sorted list in ascending order.
- **Steps:**
  1. Base Case: If the list has 1 or fewer elements, it is already sorted.
  2. Split the List: Divide the list into two halves.
  3. Recursive Sort: Recursively sort each half.
  4. Merge, Merge the two sorted halves into a single sorted list:
     - Compare elements from both halves and insert the smaller one into the result.
  5. Repeat until the entire list is sorted.

How merge Sort works



Recursive calls and merge

- Step 1: Initial Call
- merge_sort([99.99, 49.95, 299.49, 19.95])
- Split into two halves:
  - left_half = [99.99, 49.95]
  - right_half = [299.49, 19.95]
- Step 2: Recursive Call on left_half = [99.99, 49.95]
- merge_sort([99.99, 49.95])
- Split into:
  - left_half = [99.99]
  - right_half = [49.95]

Recursive calls and merge

- Step 3: Recursive Call on Single Elements
- merge_sort([99.99]) → Base case, return [99.99]
- merge_sort([49.95]) → Base case, return [49.95]

- Step 4: Merge [99.99] and [49.95]
- Compare 99.99 (left_half[0]) with 49.95 (right_half[0]).
  - 49.95 is smaller → Place 49.95 into prices[0]
- Left Elements:
  - Place 99.99 from left_half[0] into prices[1].
- Result of Merge:
  - [49.95, 99.99].

Recursive calls and merge

- Step 5: Recursive Call on right_half = [299.49, 19.95]
- merge_sort([299.49, 19.95])
- Split into:
    - left_half = [299.49]
    - right_half = [19.95]

- Step 6: Recursive Call on Single Elements
- merge_sort([299.49]) → Base case, return [299.49]
- merge_sort([19.95]) → Base case, return [19.95]

Recursive calls and merge

- Step 7: Merge [299.49] and [19.95]
- Compare 299.49 (left_half[0]) with 19.95 (right_half[0])
    - 19.95 is smaller → Place 19.95 into prices[0].
- Left Elements:
    - Place 299.49 from left_half[0] into prices[1].
- Result of Merge:
    - [19.95, 299.49].

Recursive calls and merge

- Step 8: Merge [49.95, 99.99] and [19.95, 299.49]

1. Compare 49.95 (left_half[0]) with 19.95 (right_half[0]):
   - 19.95 is smaller → Place 19.95 into prices[0].
2. Compare 49.95 (left_half[0]) with 299.49 (right_half[1]):
   - 49.95 is smaller → Place 49.95 into prices[1].
3. Compare 99.99 (left_half[1]) with 299.49 (right_half[1]):
   - 99.99 is smaller → Place 99.99 into prices[2].

- Left Elements:
  - Place 299.49 from right_half[1] into prices[3].
- Final Result:
  - [19.95, 49.95, 99.99, 299.49].