Understand Sorting Algorithms:

* Bubble Sort
  + Compares adjacent elements and swaps if out of order.
  + Repeats until the list is sorted.
  + Time Complexity:
    - Best: O(n) (when already sorted)
    - Average/Worst: O(n²)
* Insertion Sort
  + Builds sorted list one element at a time.
  + Time Complexity:
    - Best: O(n)
    - Average/Worst: O(n²)
* Quick Sort
  + Picks a pivot and partitions array into two halves.
  + Recursively sorts each half.
  + Time Complexity:
    - Best/Average: O(n log n)
    - Worst: O(n²) (rare)
* Merge Sort
  + Divides the array, sorts each part, and merges them.
  + Time Complexity: Always O(n log n)
    - Needs extra memory for merging.

Analysis:

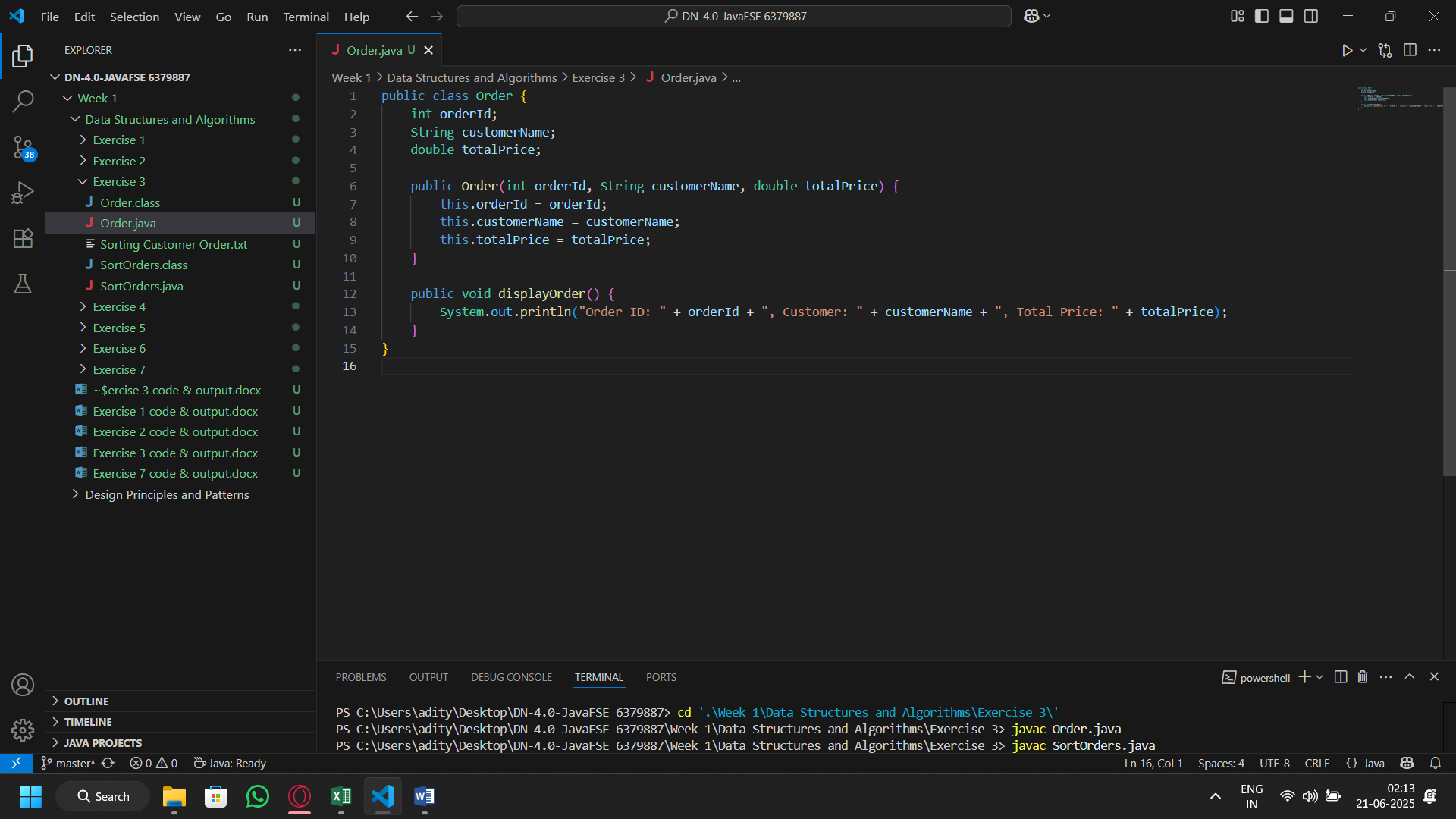
|  |  |  |
| --- | --- | --- |
| Operation | Bubble Sort | Quick Sort |
| Best Case | O(n) | O(n log n) |
| Average Case | O(n²) | O(n log n) |
| Worst Case | O(n²) | O(n²) (rare) |
| Stability | Stable | Not stable (usually) |
| Memory Usage | In-place | In-place |

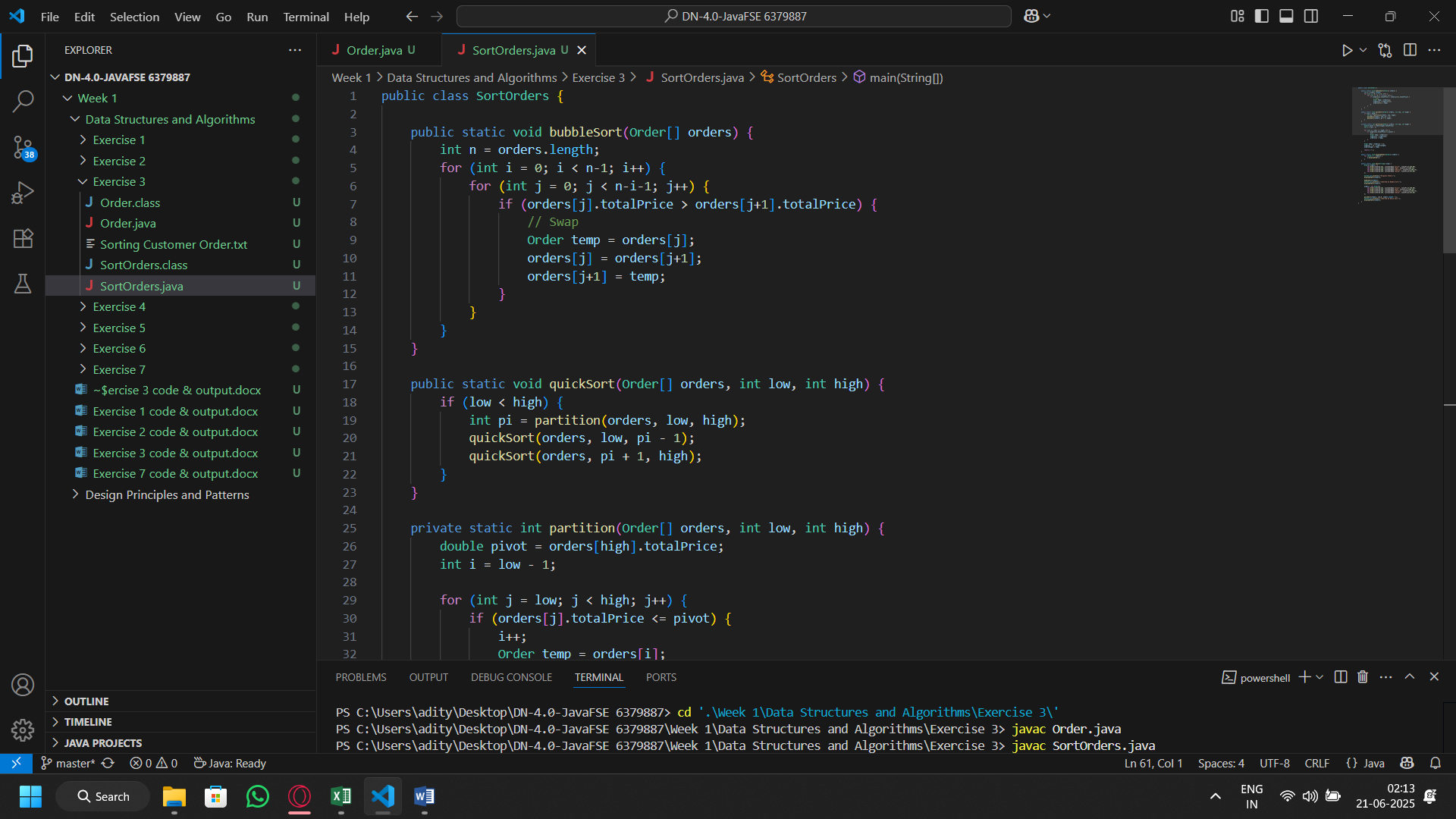
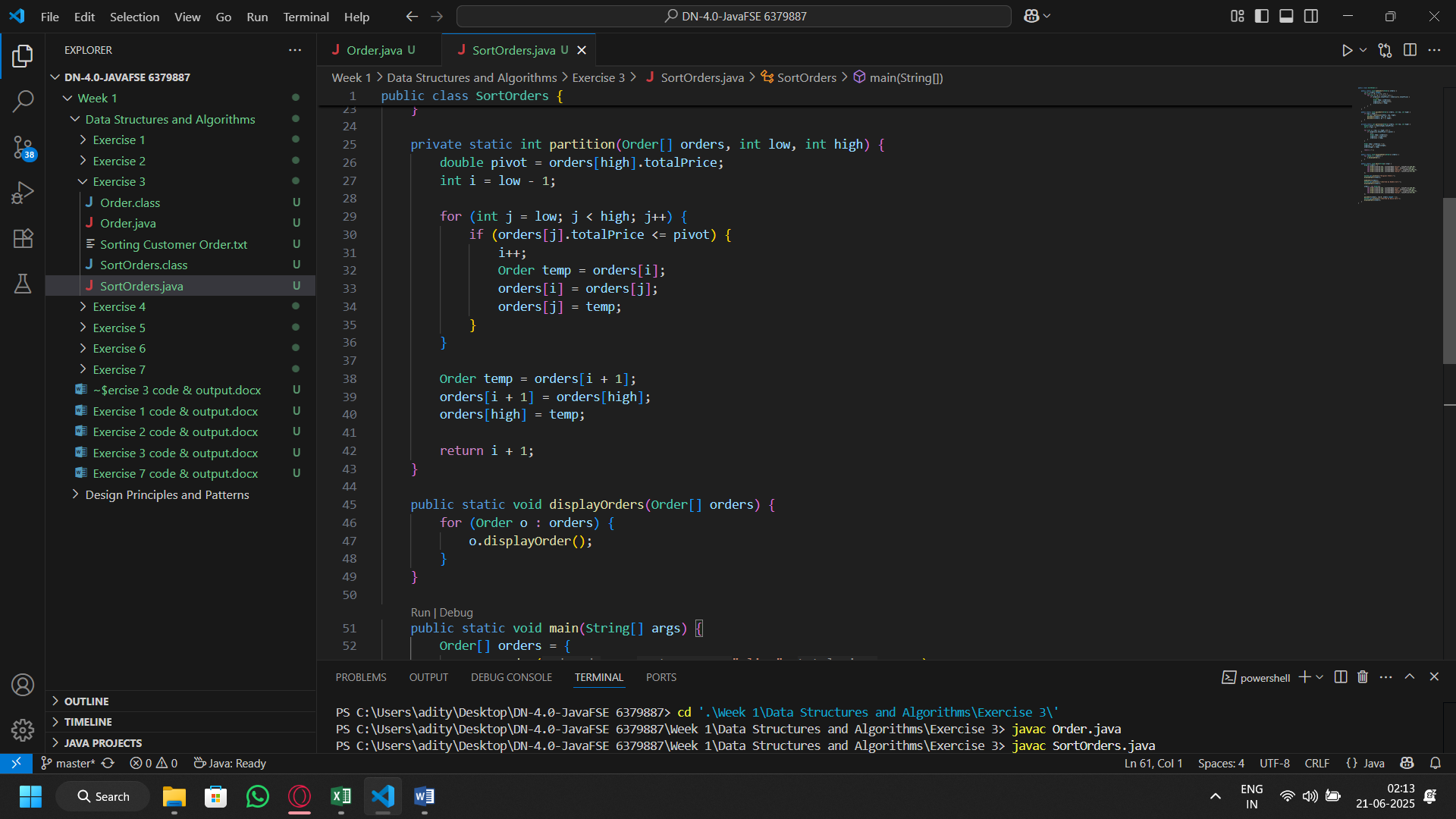
Quick sort is preferred as it is much faster on average than Bubble Sort.

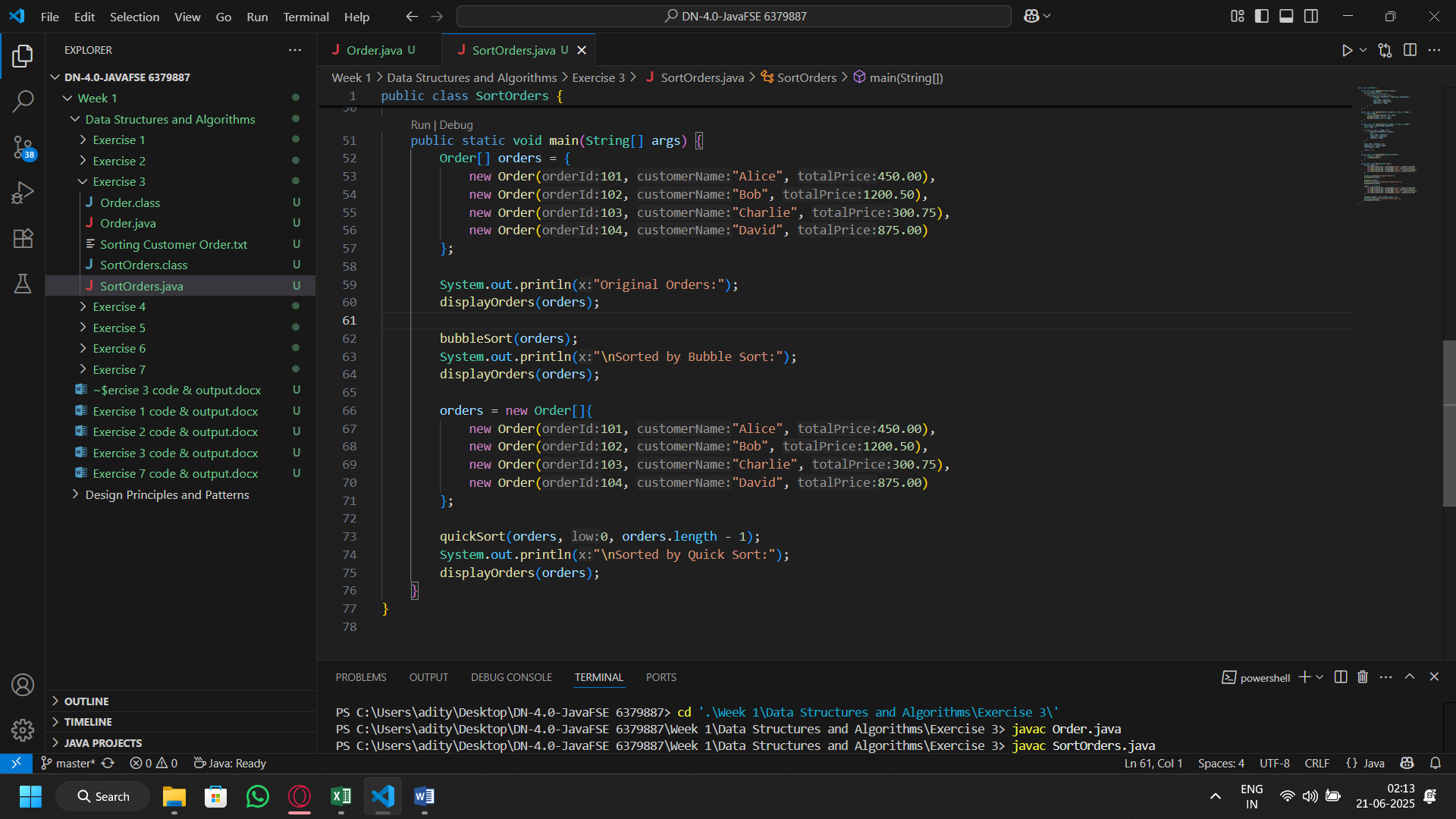
Efficient on large datasets due to O(n log n) performance.

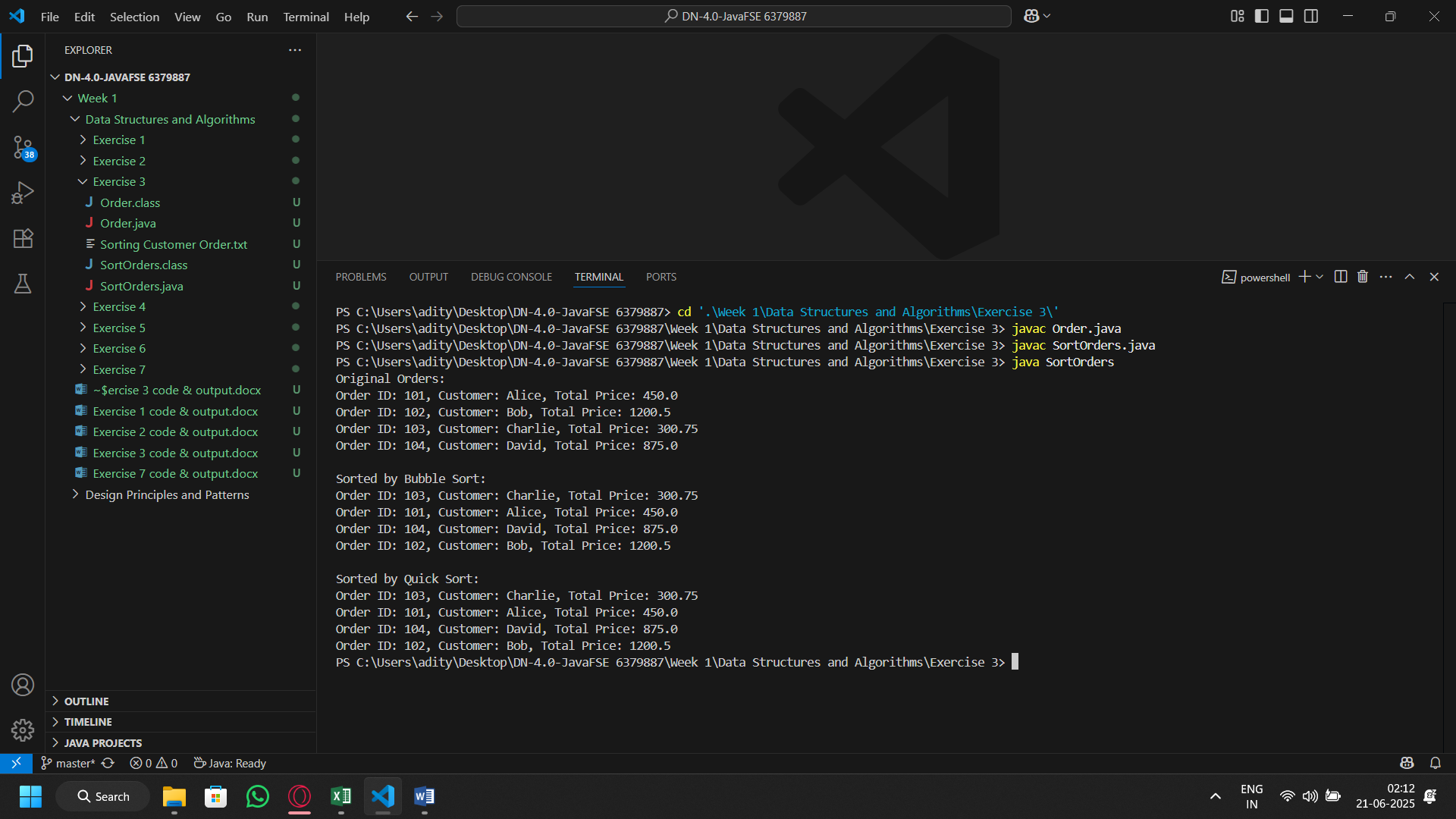
Bubble Sort is simple but inefficient for most real-world use.

Code:

Order Class

Sort Orders Class



Output: