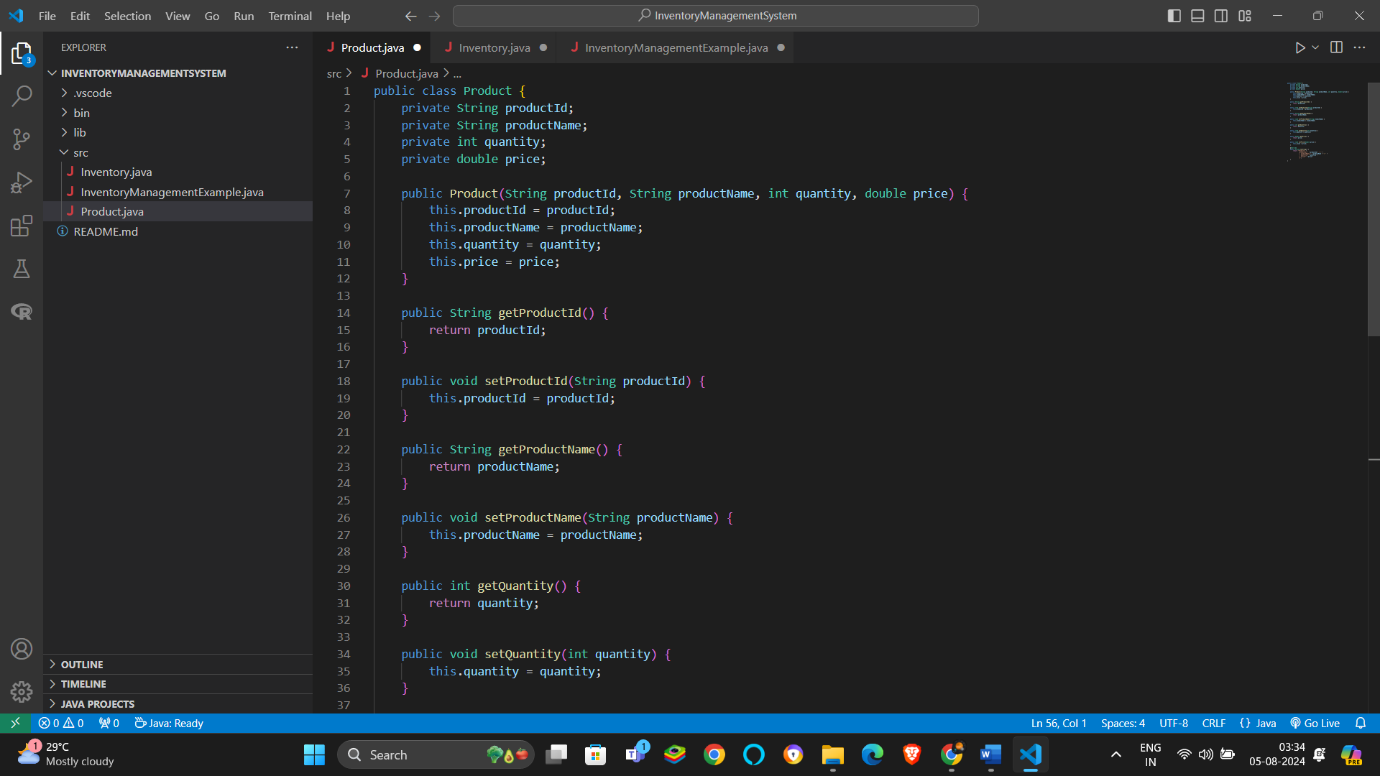
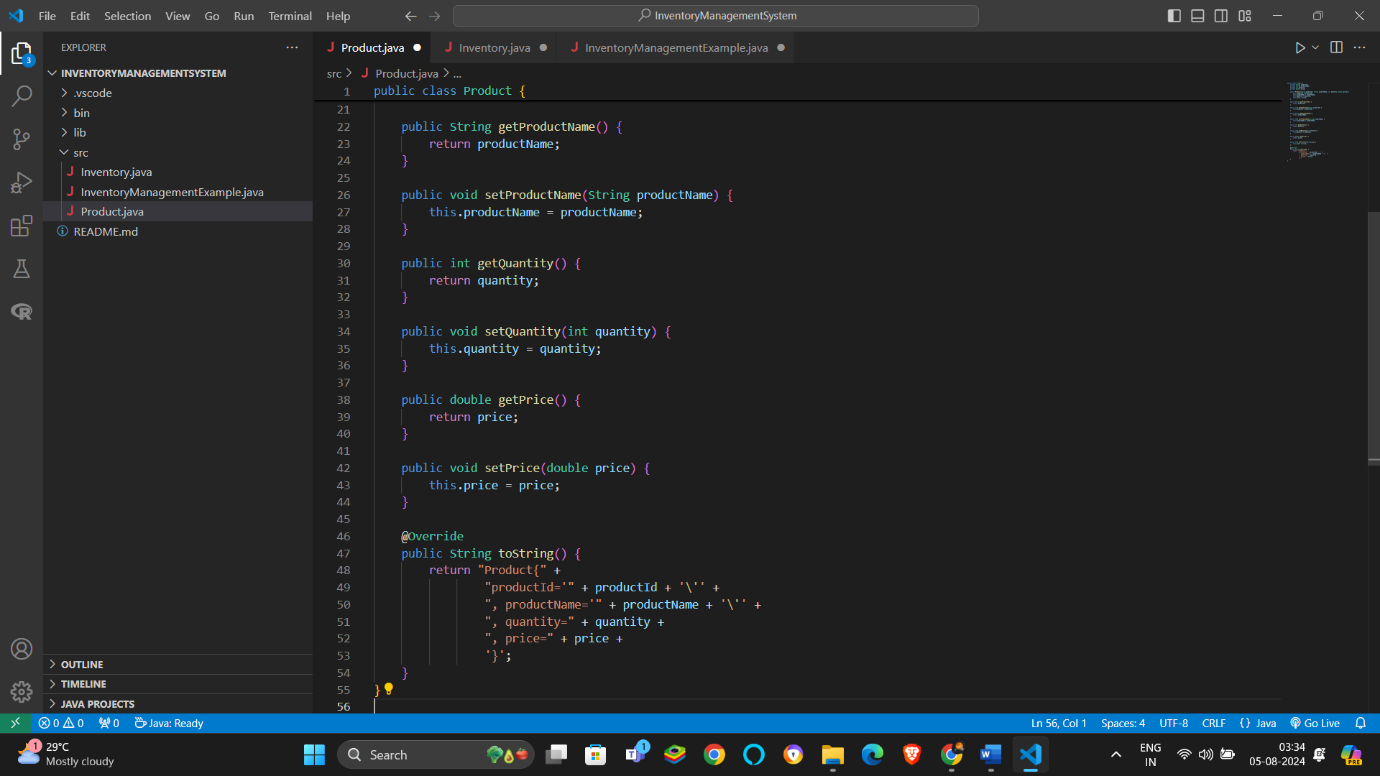
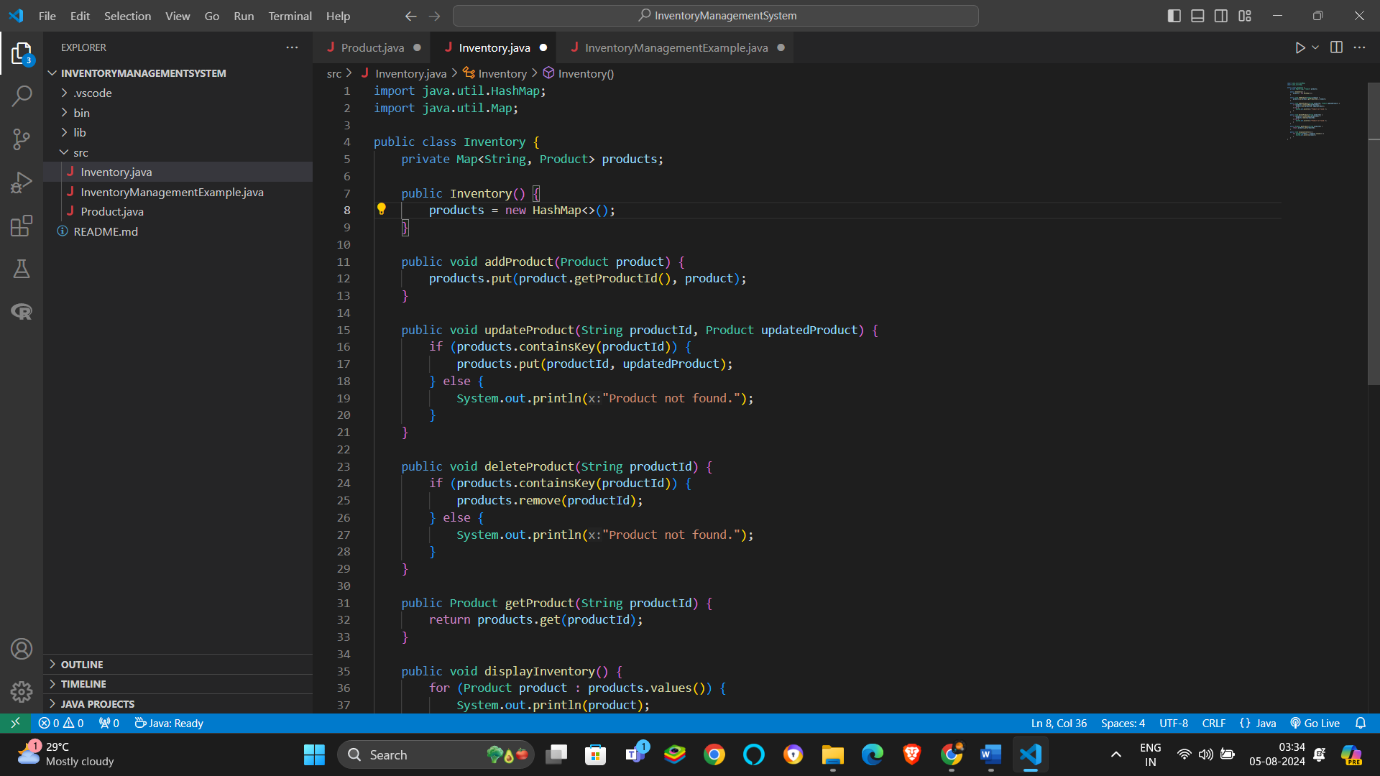
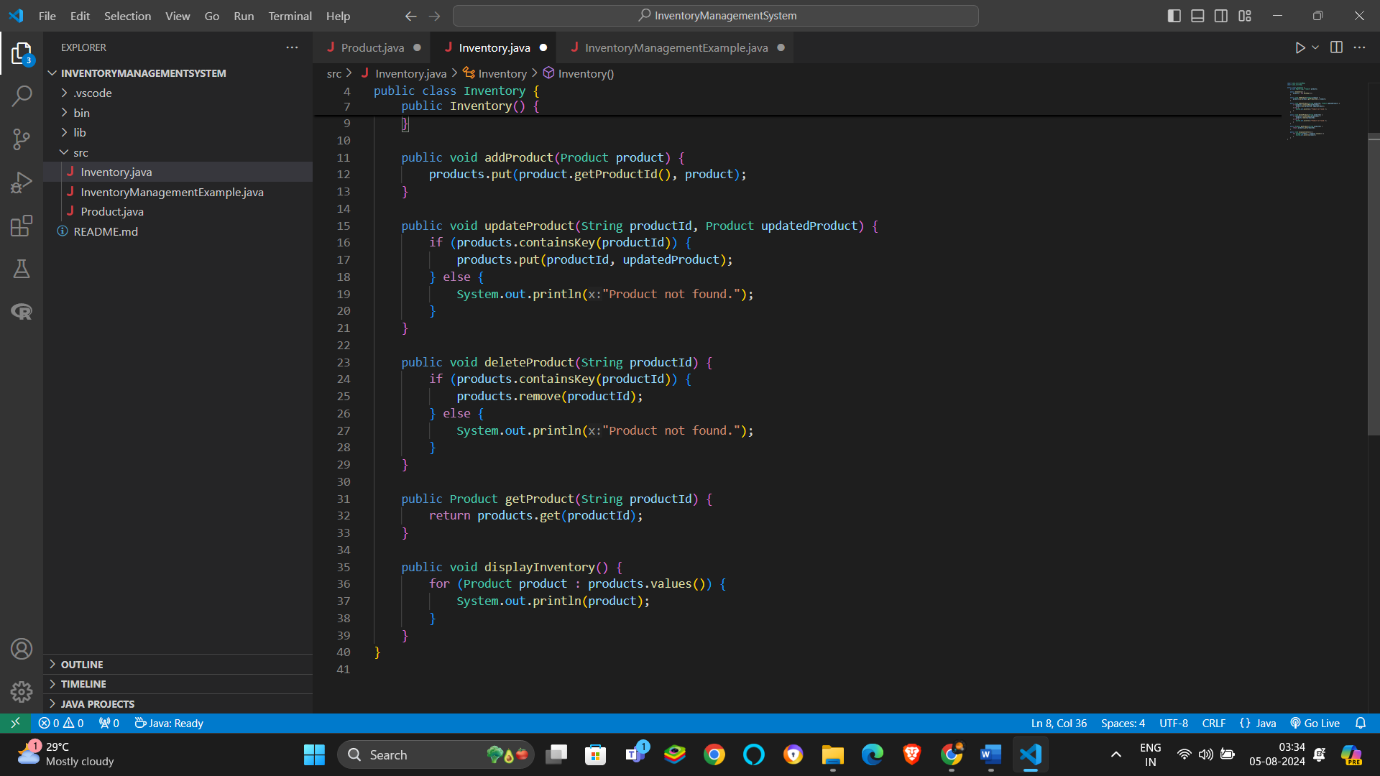
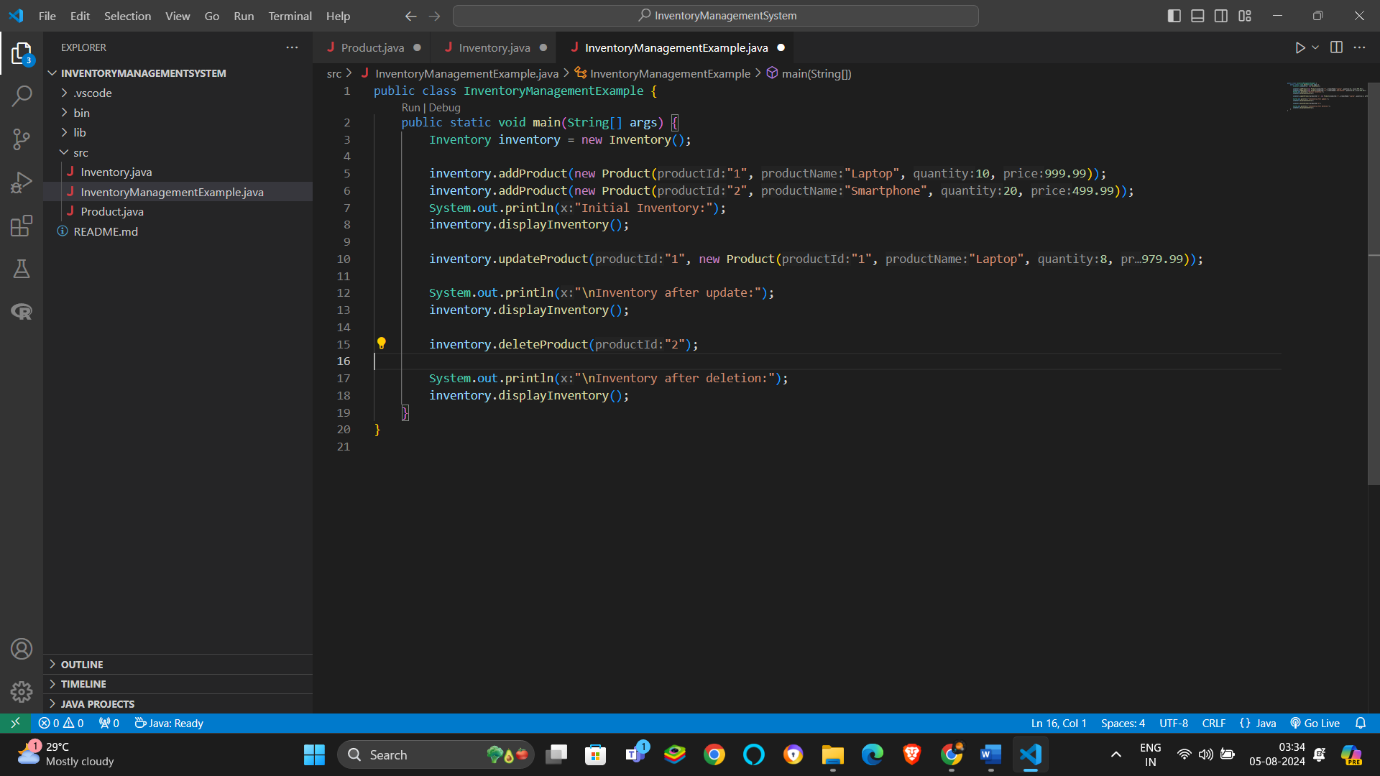
Exercise 1-

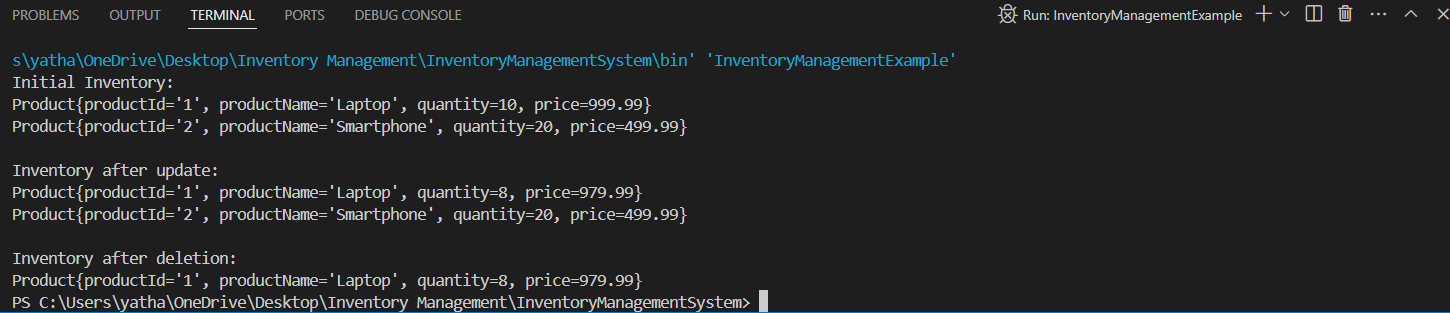










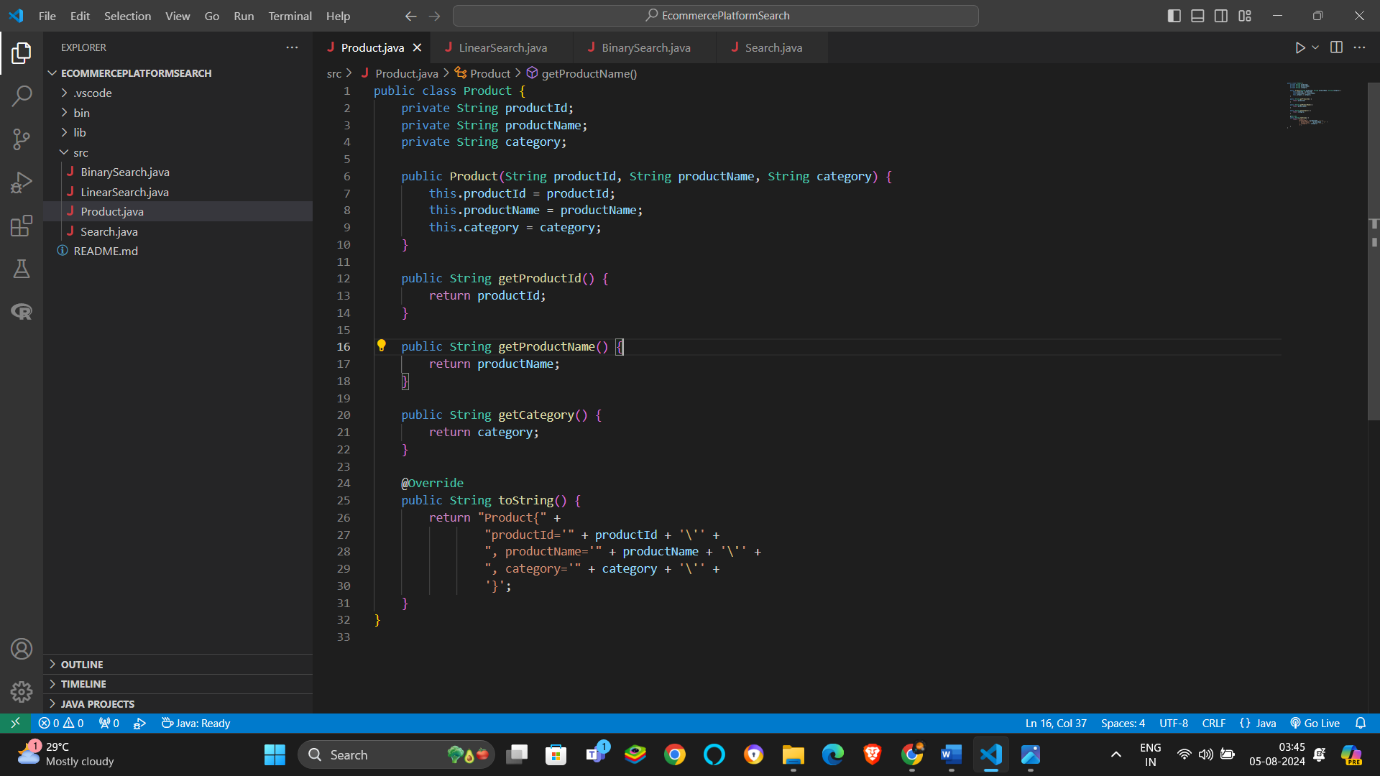


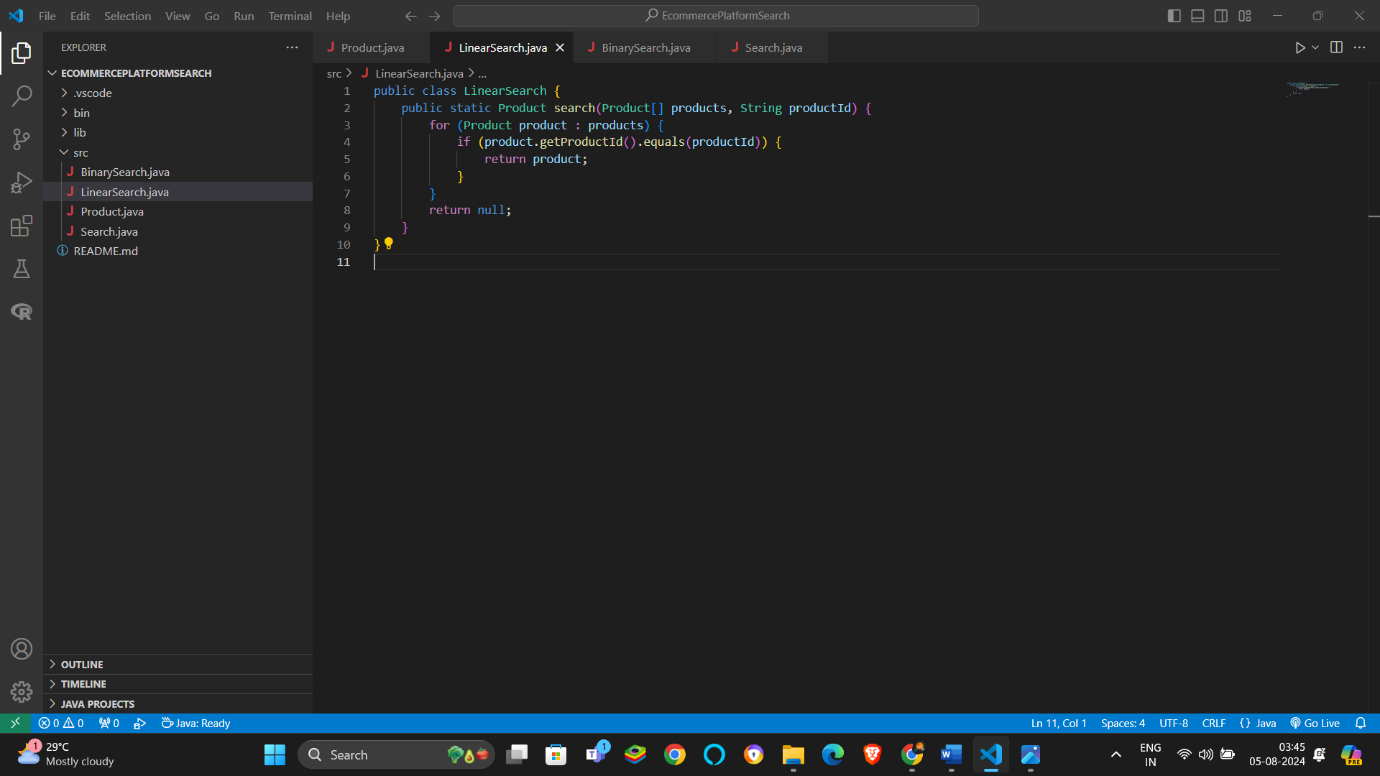
**Time Complexity Analysis:**

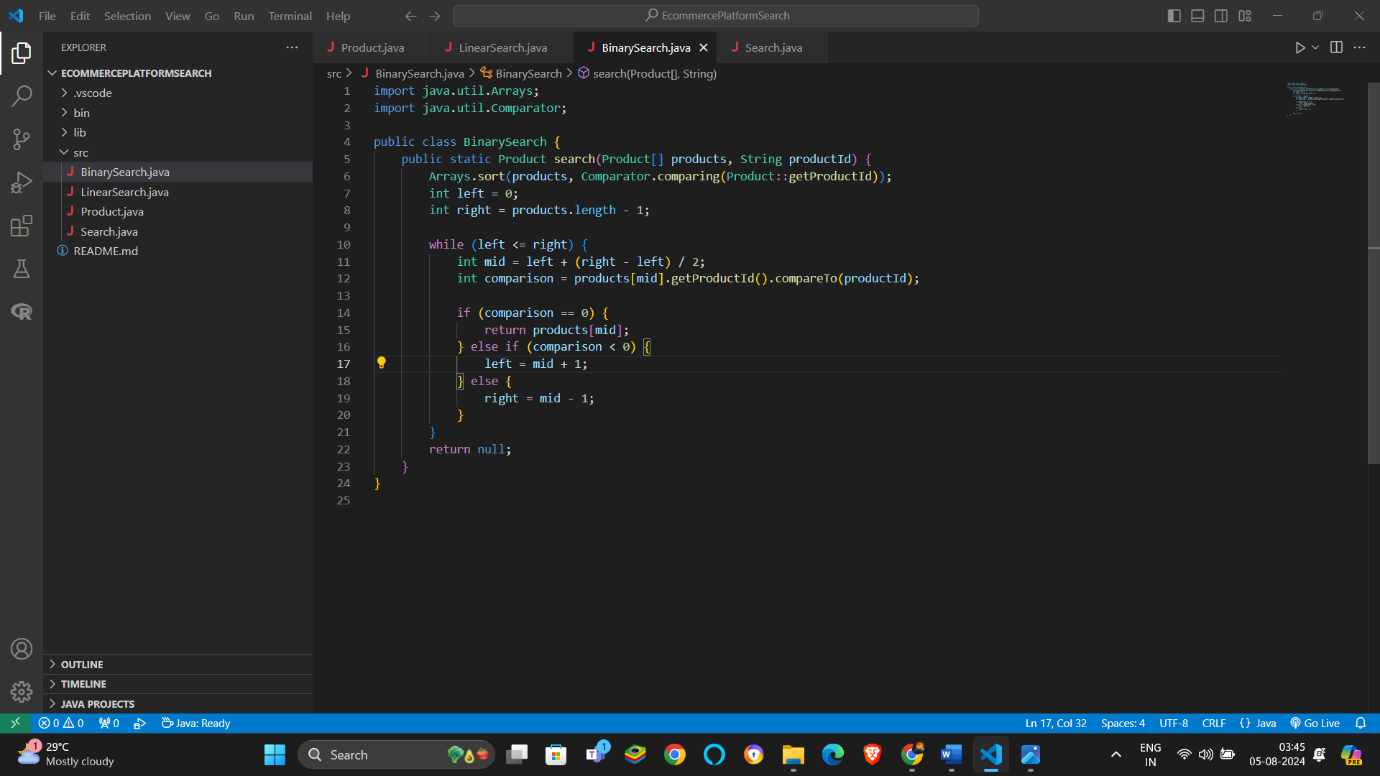
* **Add Product**: O(1)
* **Update Product**: O(1)
* **Delete Product**: O(1)
* **Get Product**: O(1)

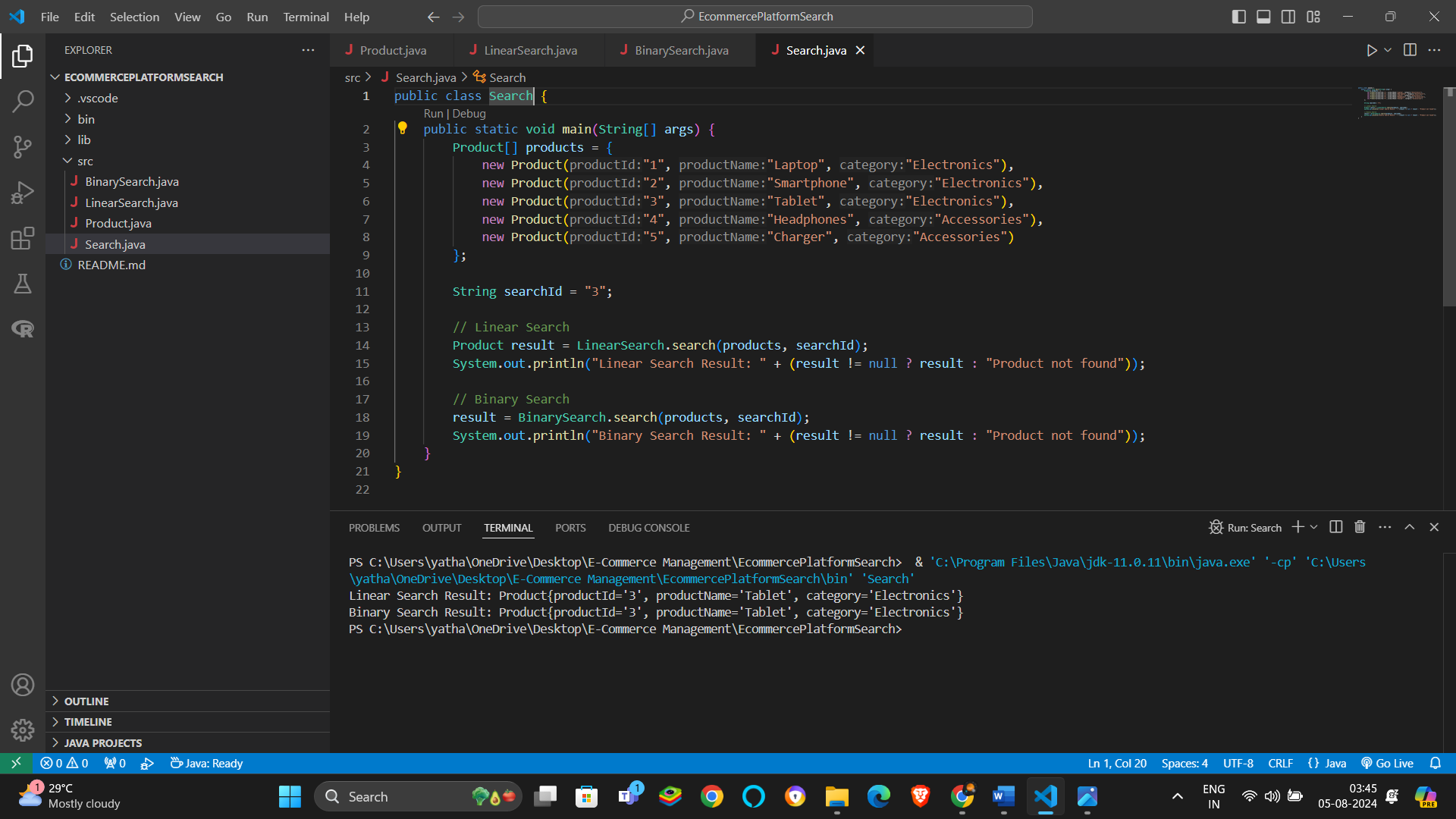
To Optimize these operations a unique product id should be maintained .

Exercise 2-









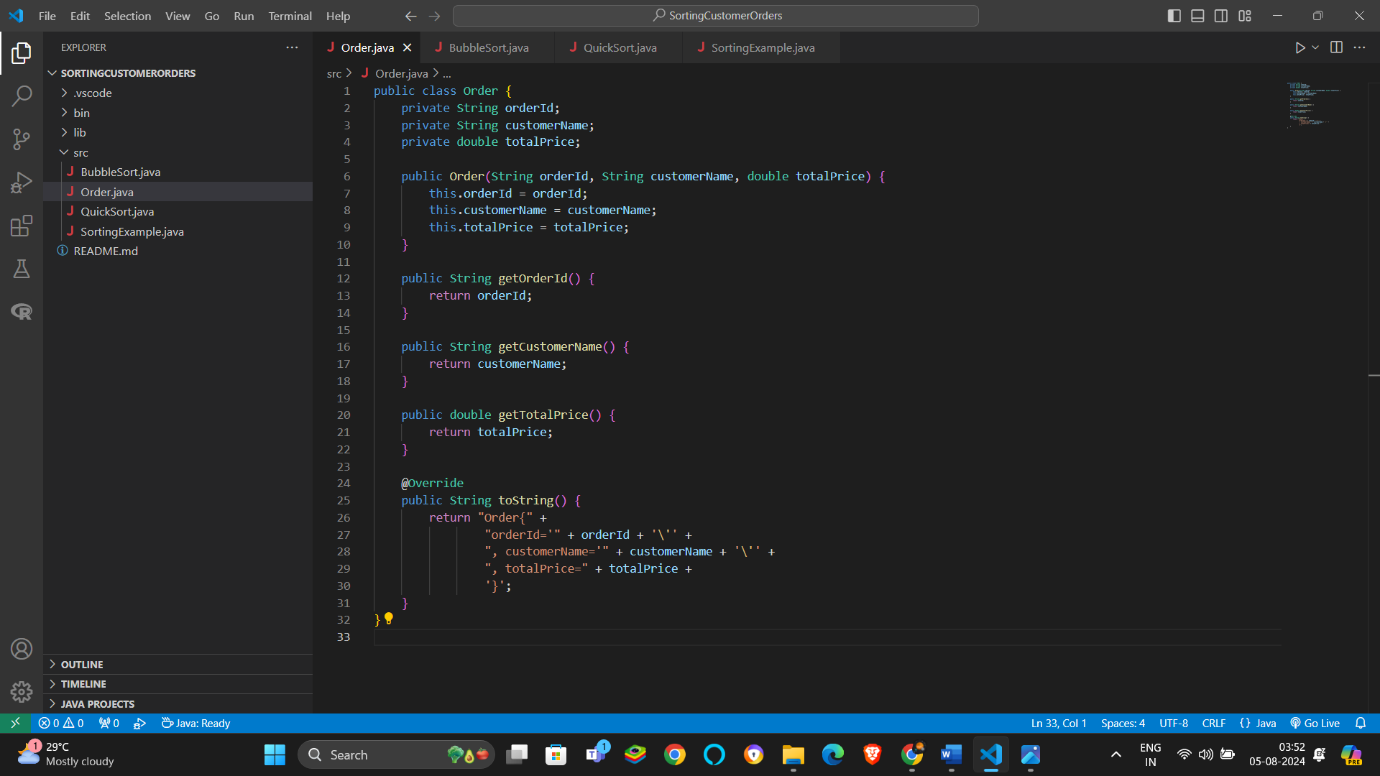
**Time Complexity Comparison:**

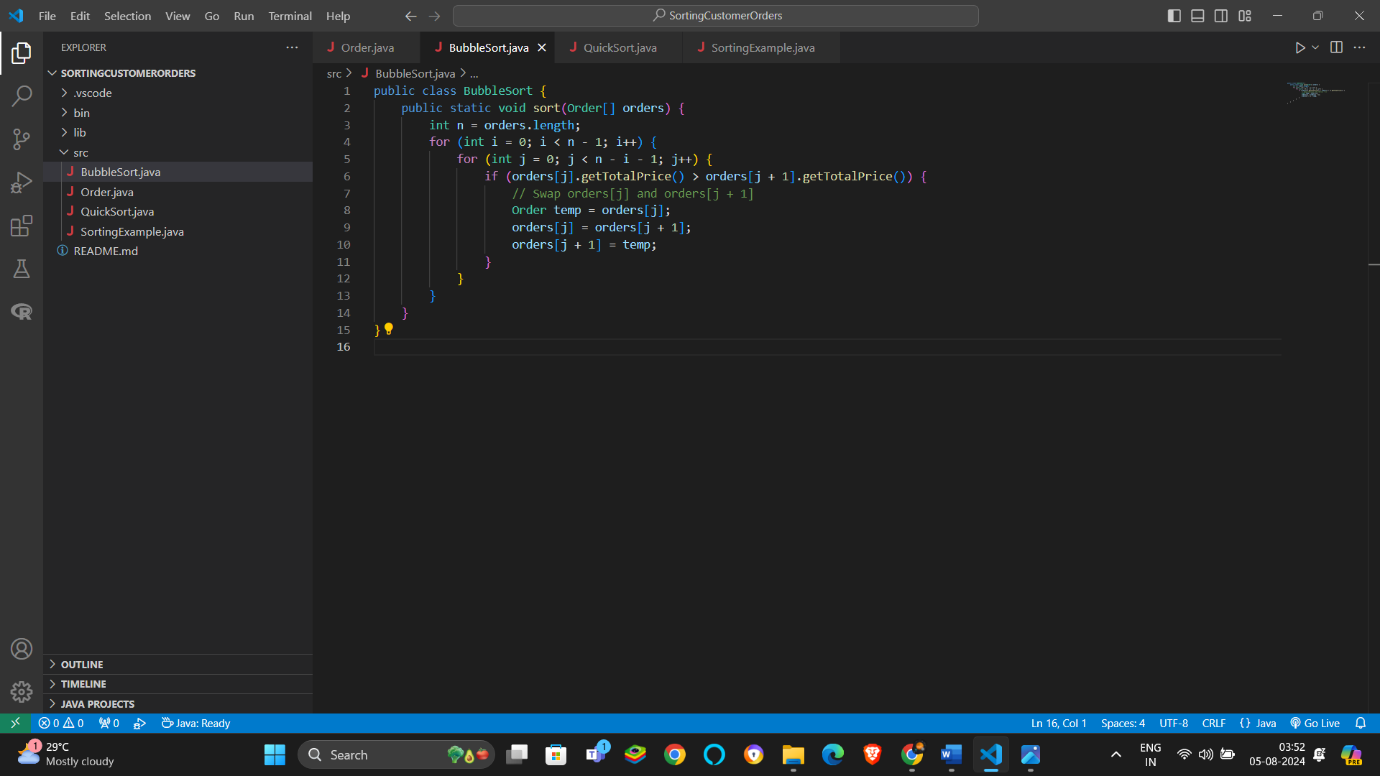
* **Linear Search**: O(n)
* **Binary Search:** O(log n)

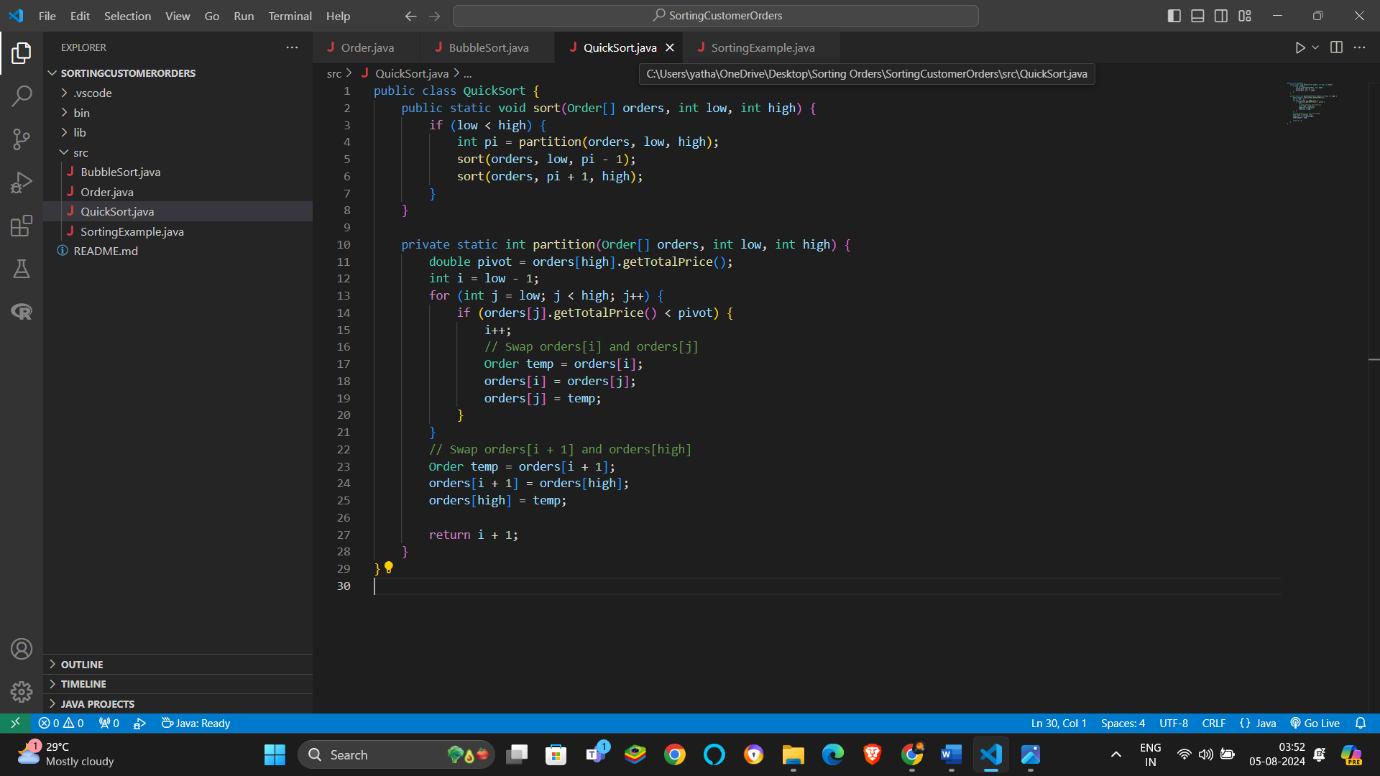
**Which Algorithm is More Suitable:**

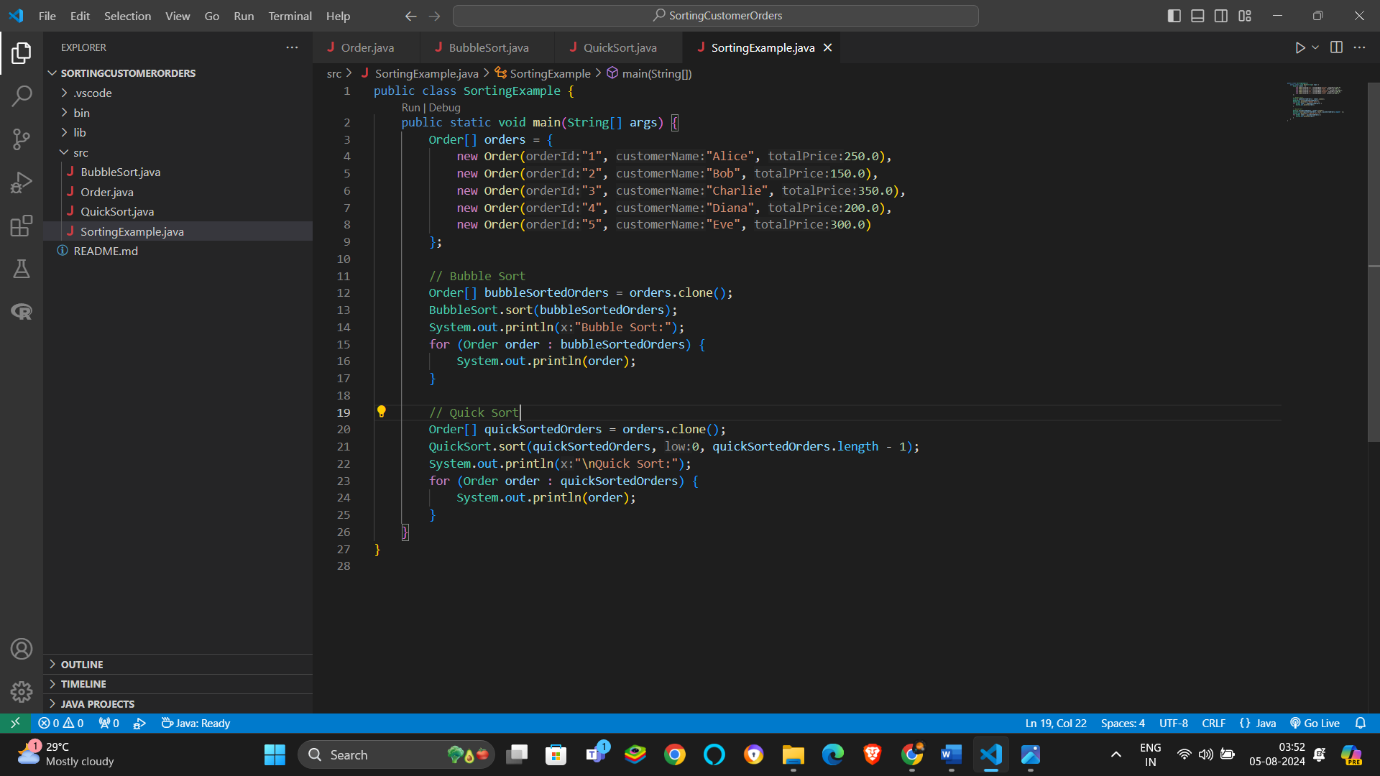
* **Linear Search** is more suitable for unsorted arrays or when the dataset is small and search operations are infrequent.
* **Binary Search** is more suitable for sorted arrays or when the dataset is large and search operations are frequent due to its logarithmic time complexity.

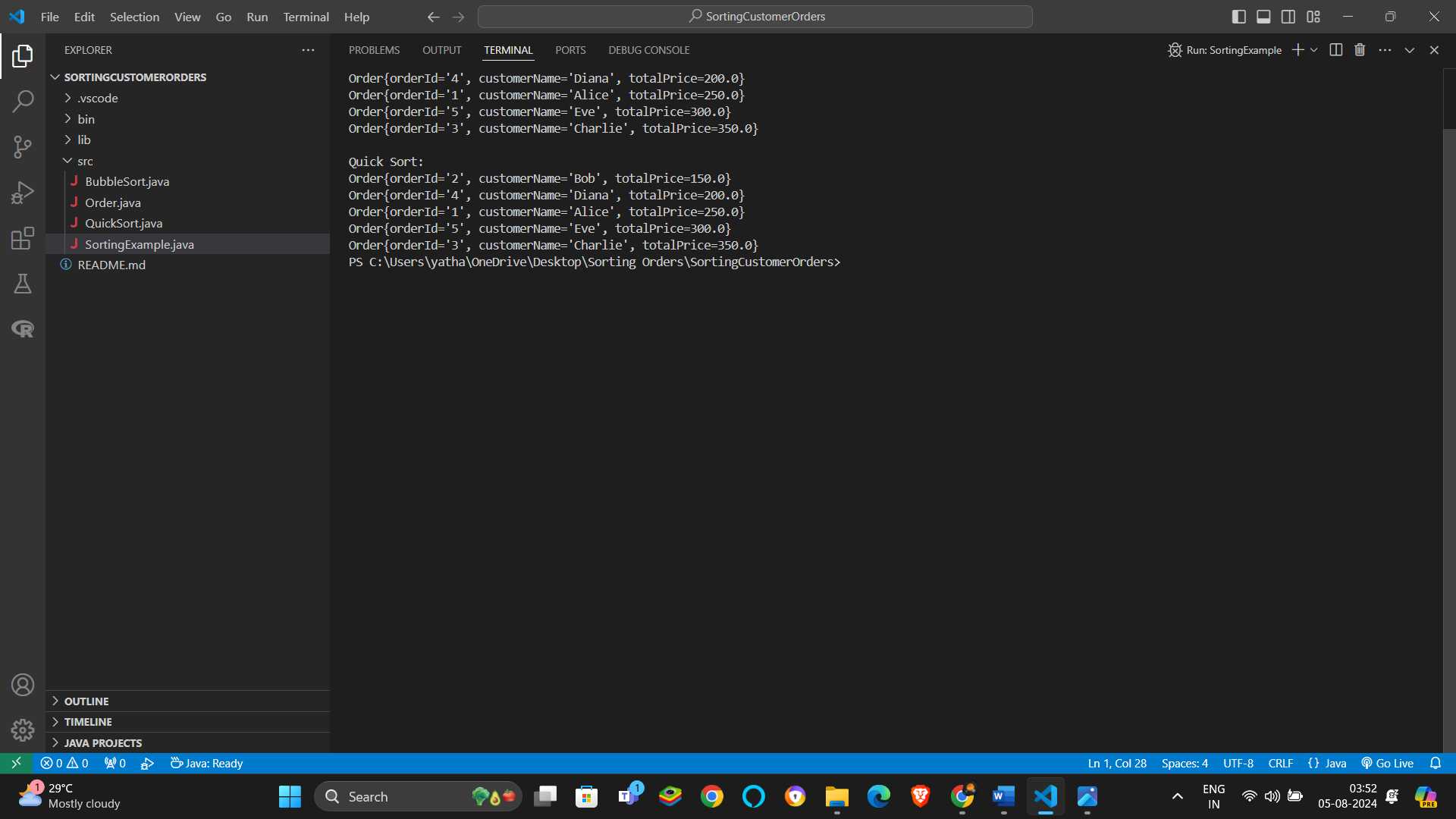
Exercise 3-











Time Complexity-

Bubble Sort-

**Best Case**: O(n) when the list is already sorted.

**Average Case**: O(n²)

**Worst Case**: O(n²)

Quick Sort-

**Best Case**: O(n log n)

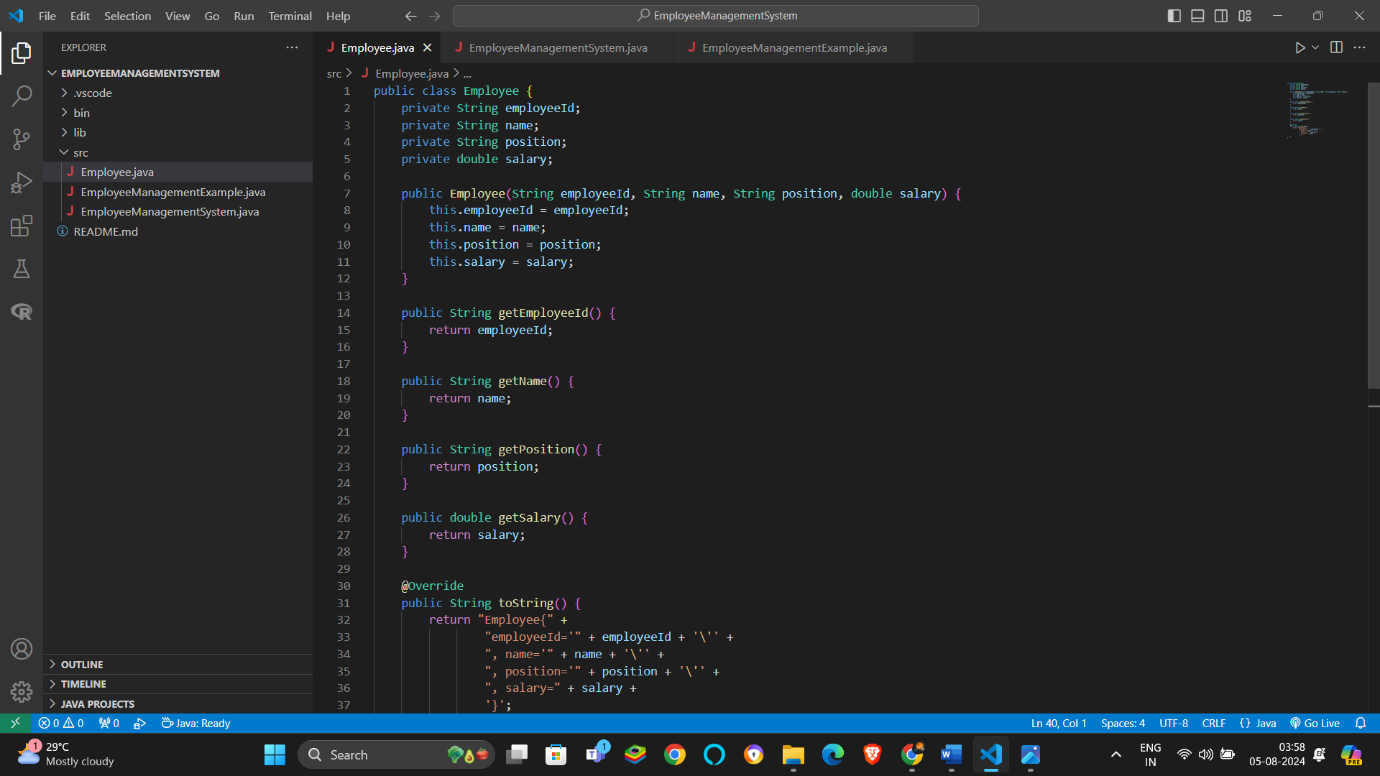
**Average Case**: O(n log n)

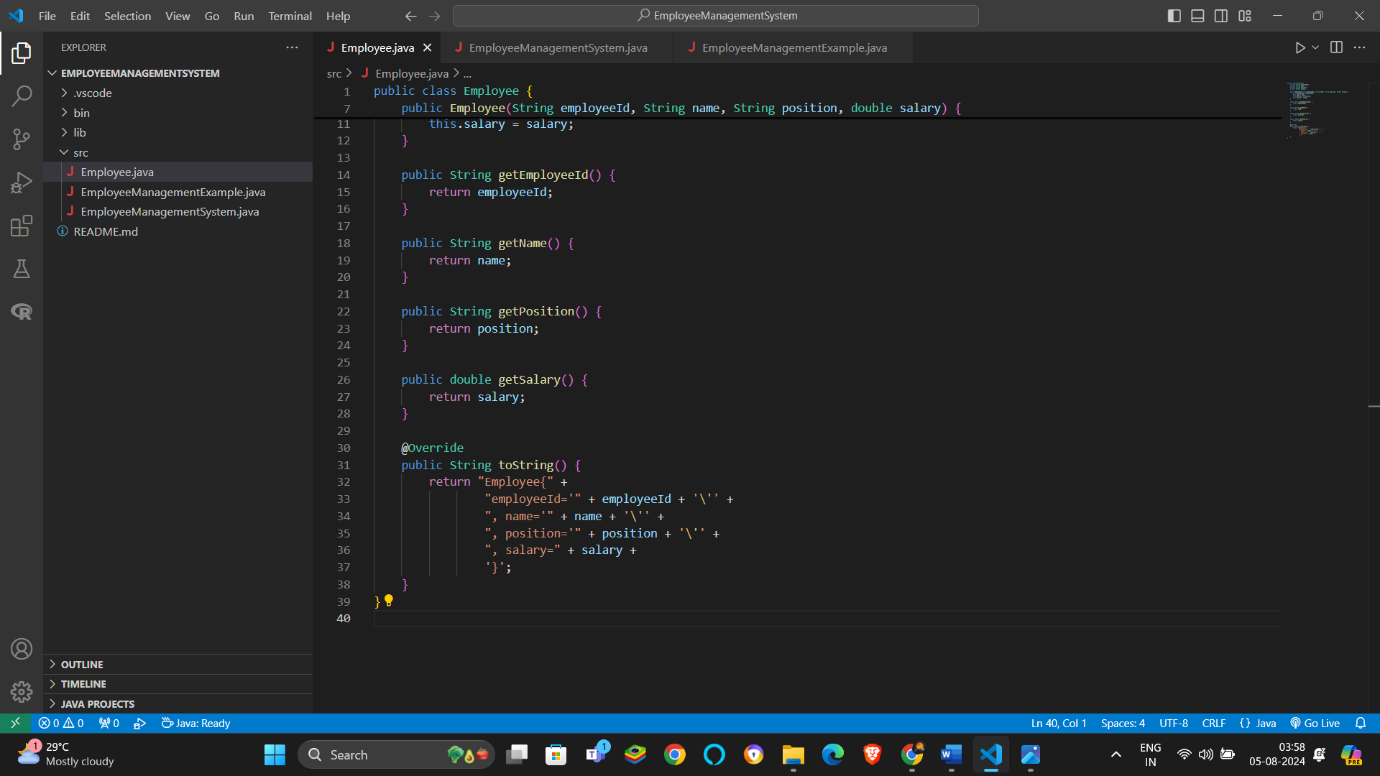
**Worst Case**: O(n²) (can be mitigated with better pivot strategies)

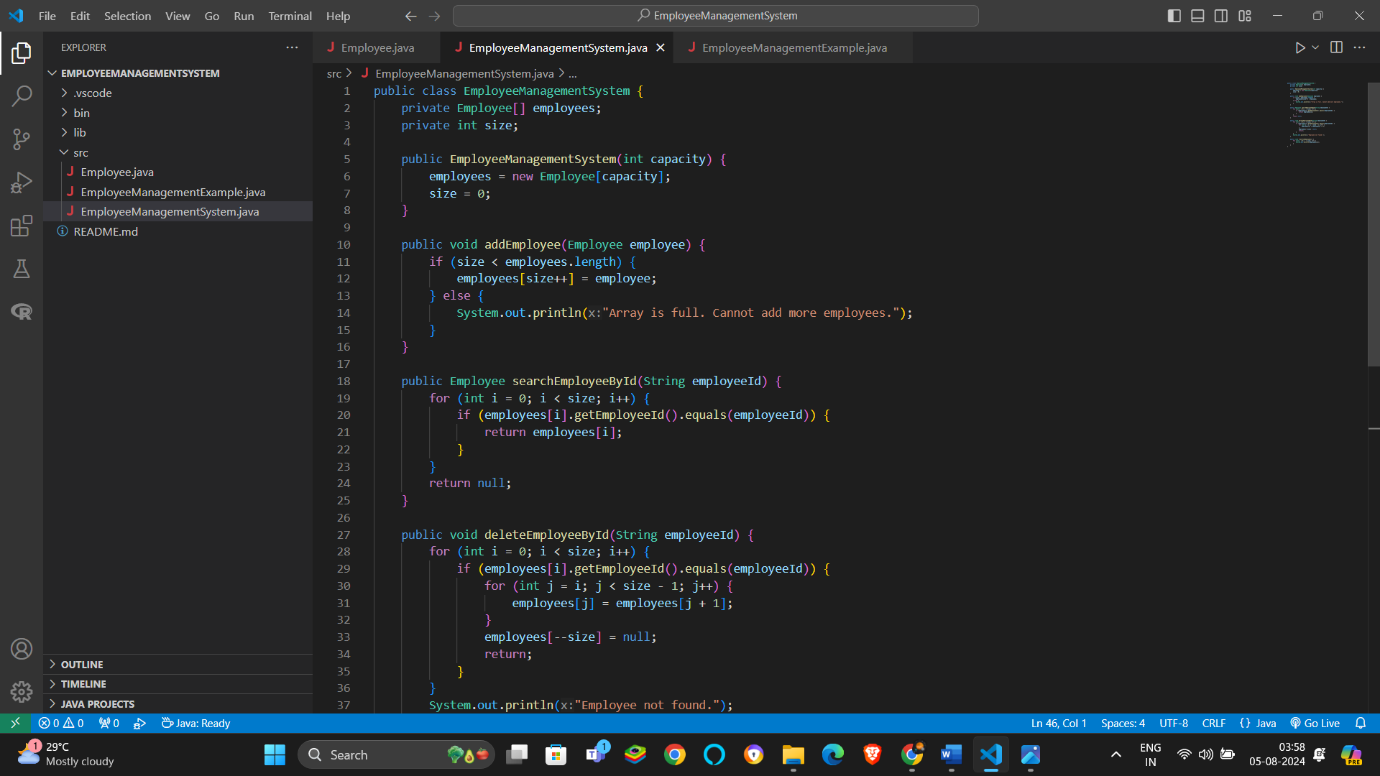
**Why Quick Sort is Generally Preferred:**

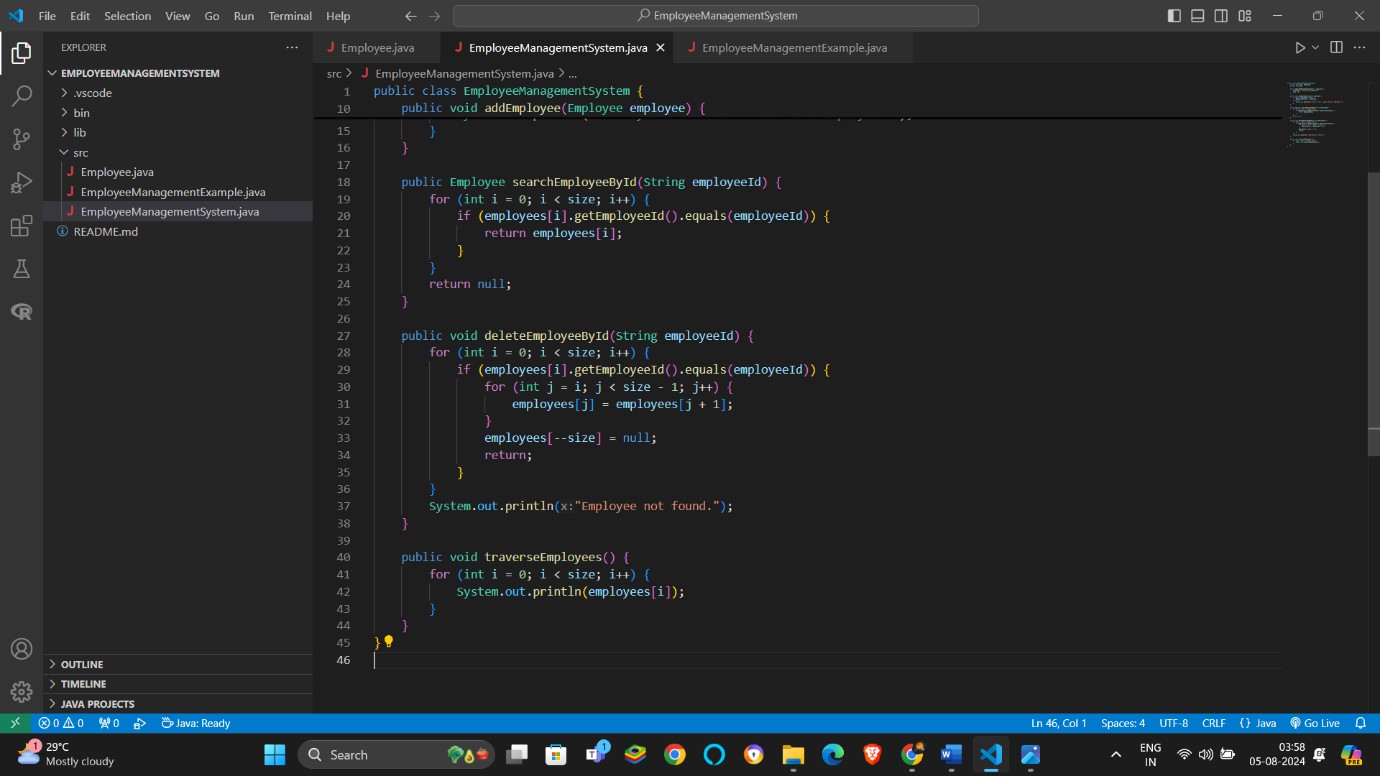
* **Efficiency**: Quick Sort has better average and best-case time complexities (O(n log n)) compared to Bubble Sort (O(n²)).
* **Divide-and-Conquer**: Quick Sort efficiently partitions the array, reducing the problem size exponentially.

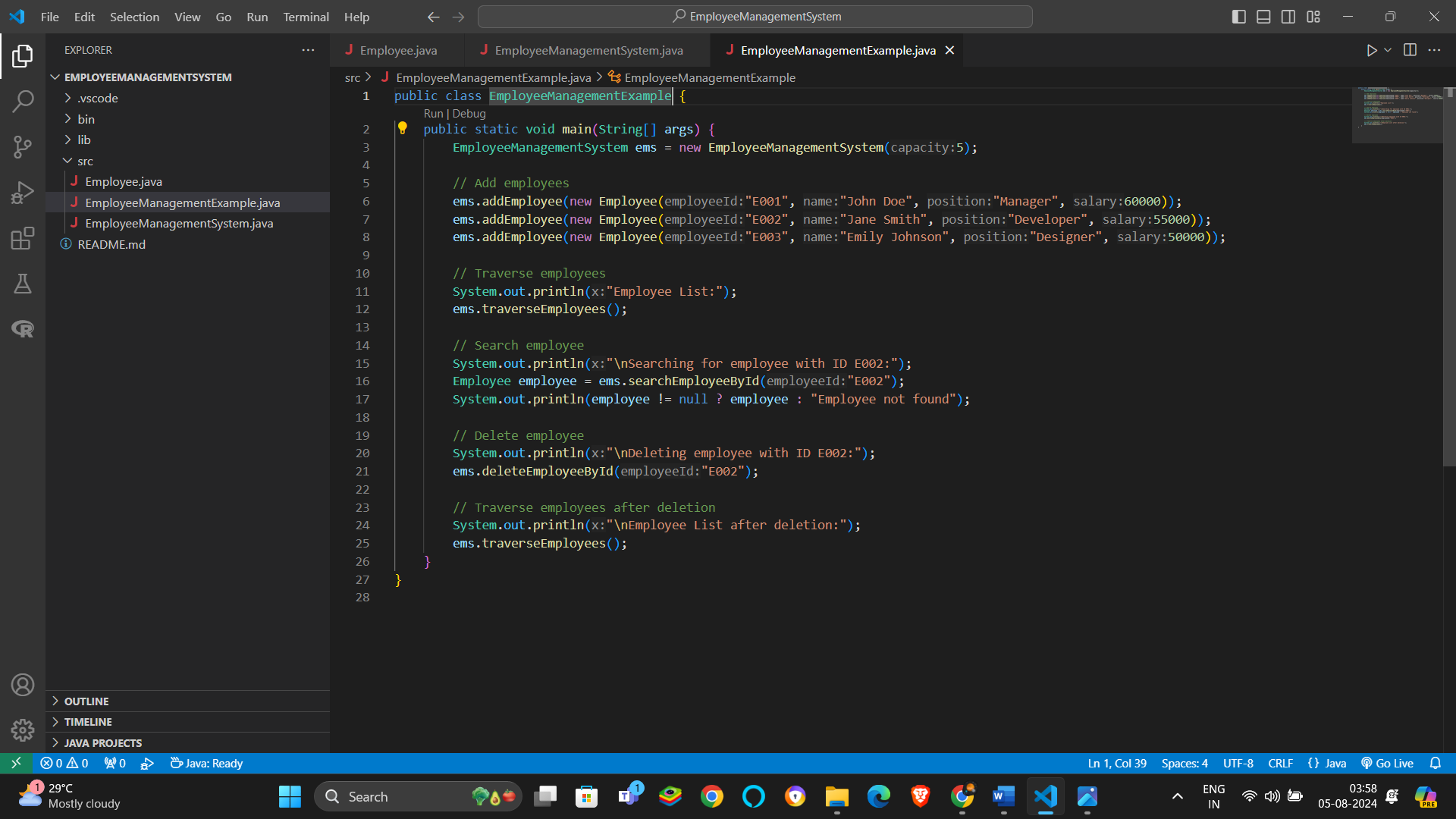
Exercise 4-

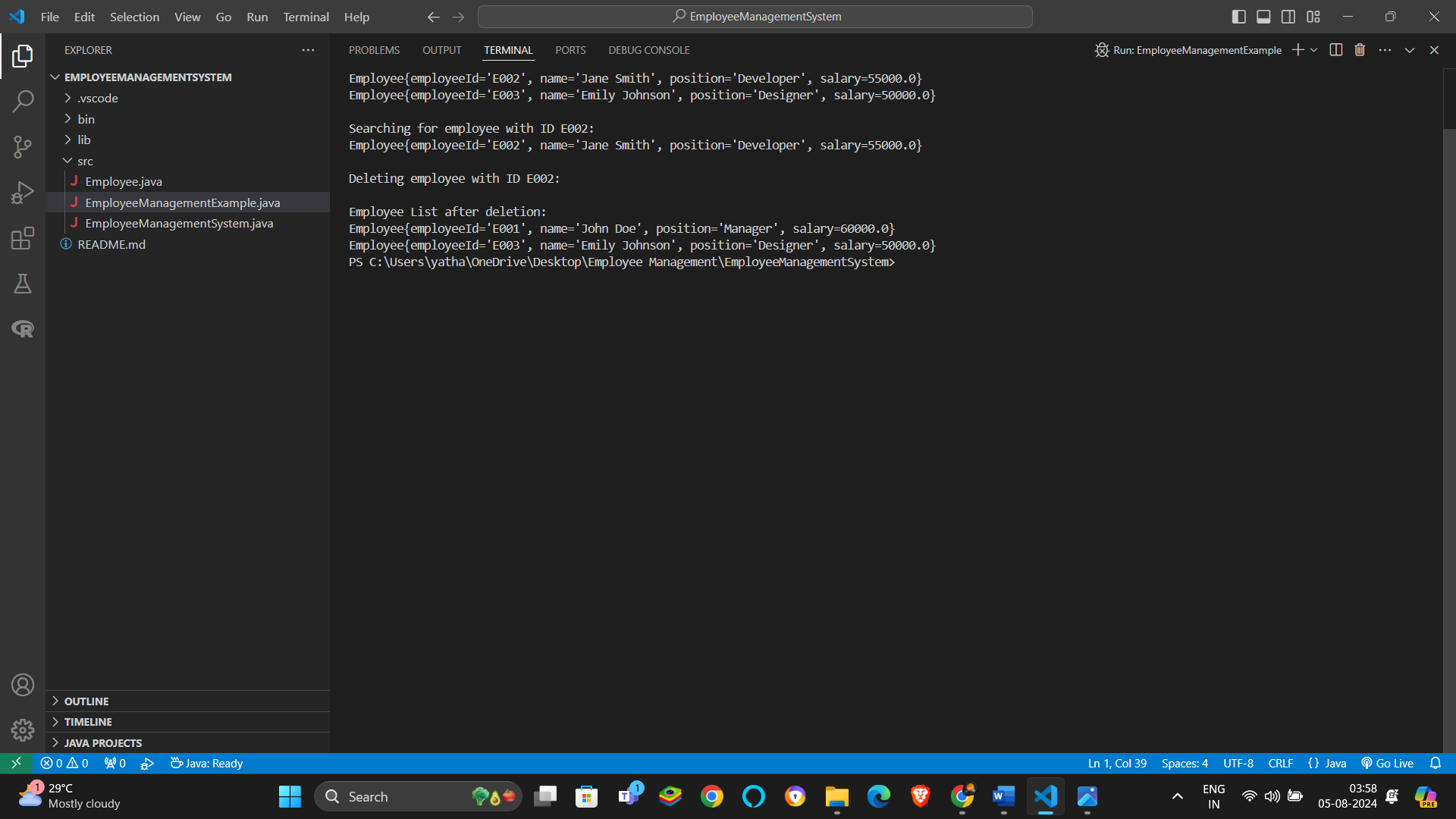












**Time Complexity Analysis:**

* **Add Task**: O(1)
* **Search Task**: O(n)
* **Delete Task**: O(n)
* **Traverse Task**: O(n)

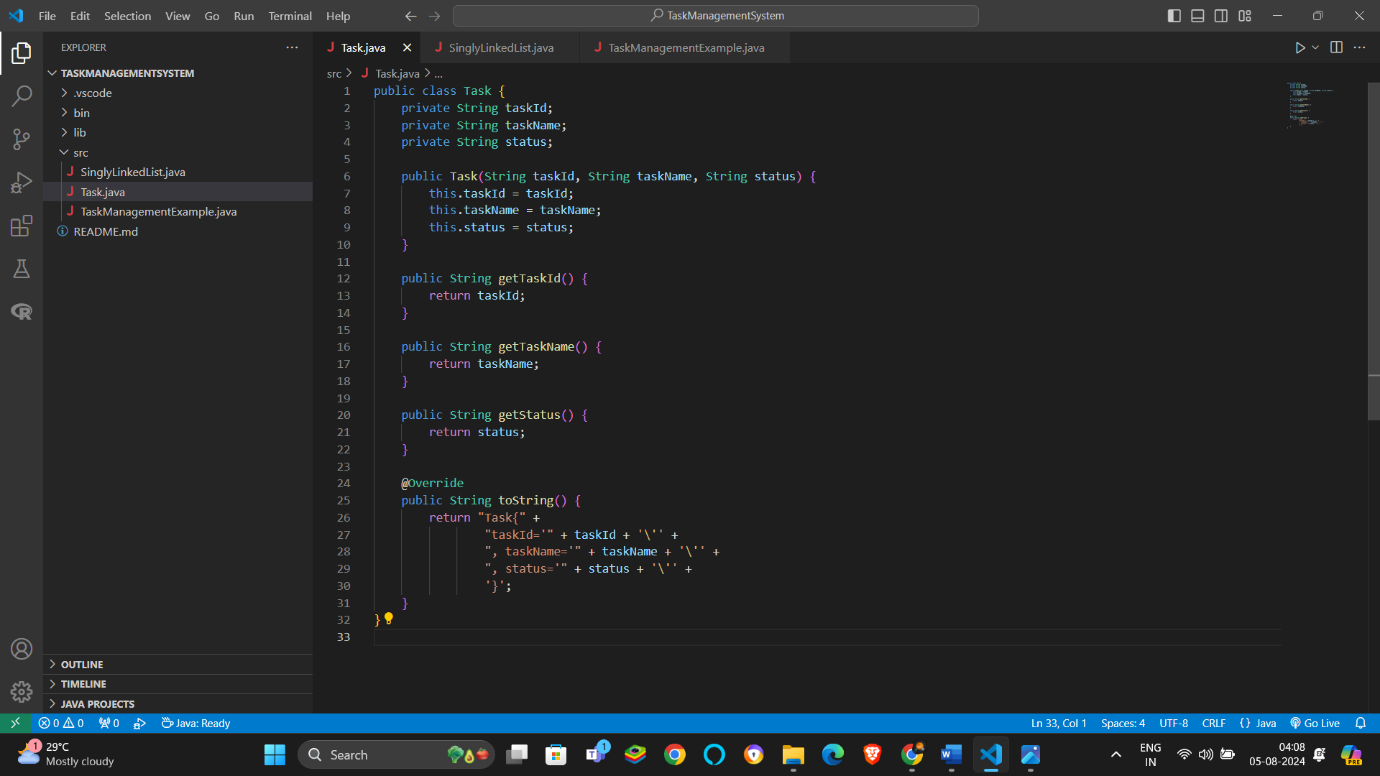
**Limitations of Arrays**:

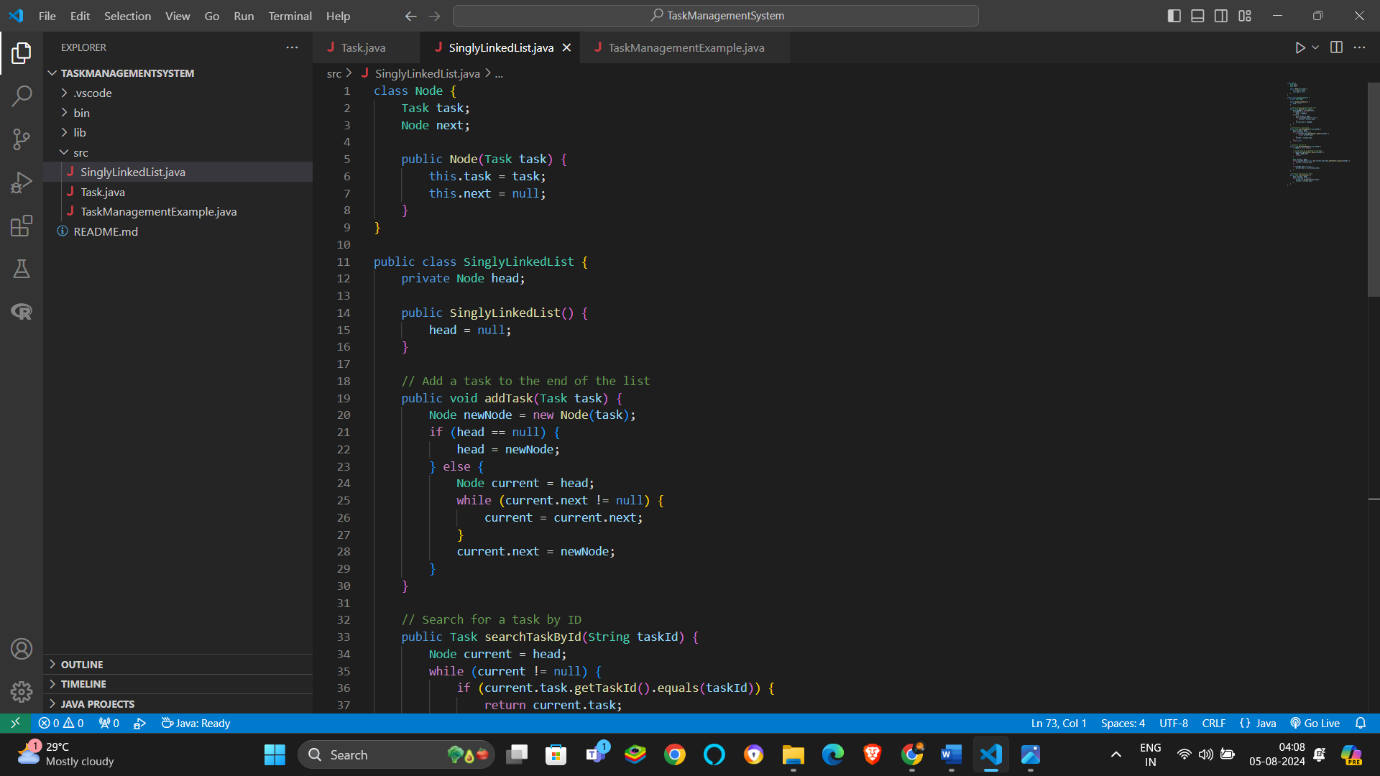
* **Fixed Size**: Arrays have a fixed size, making it challenging to add more elements once the capacity is reached.
* **Inefficient Insertions and Deletions**: Inserting or deleting elements involves shifting elements, which can be inefficient.
* **Memory Allocation**: Requires a contiguous block of memory, which might not always be available for large arrays.

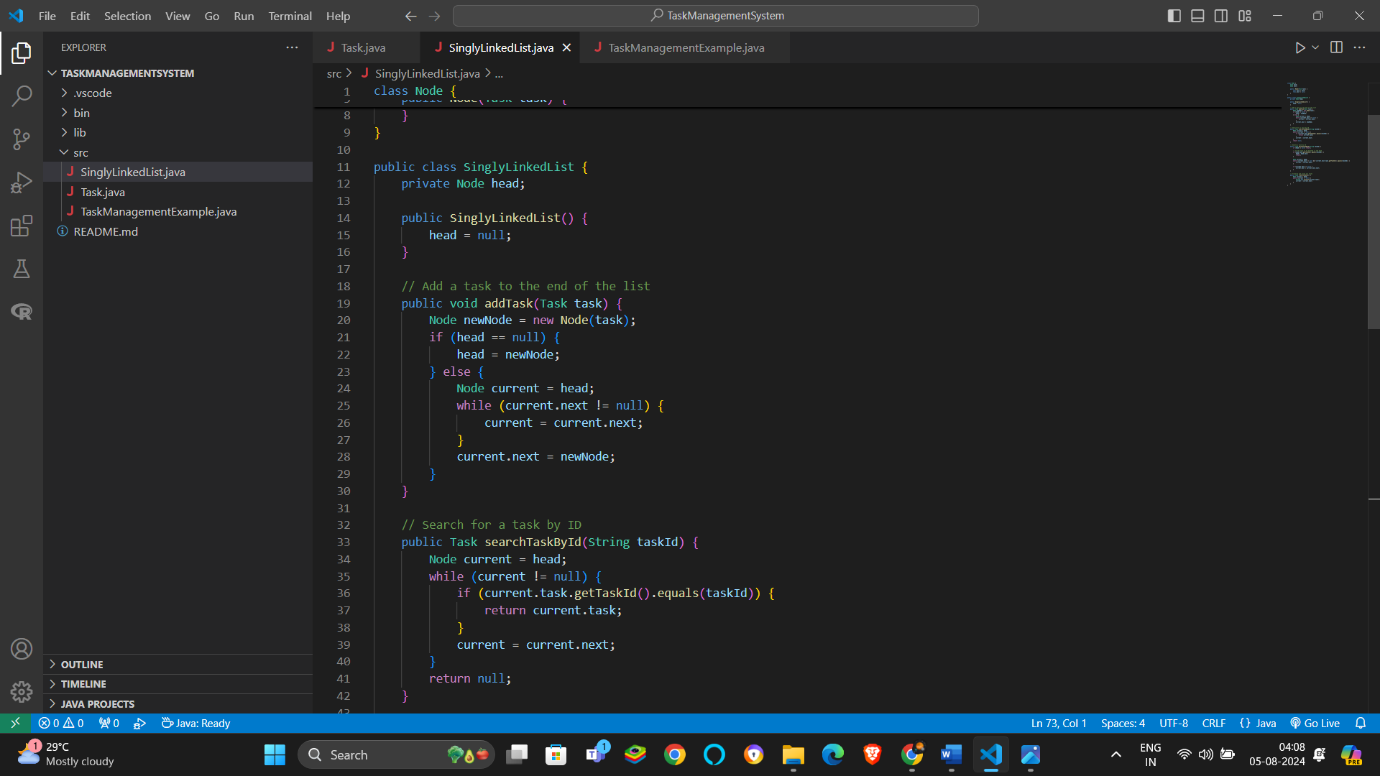
**When to Use Arrays**:

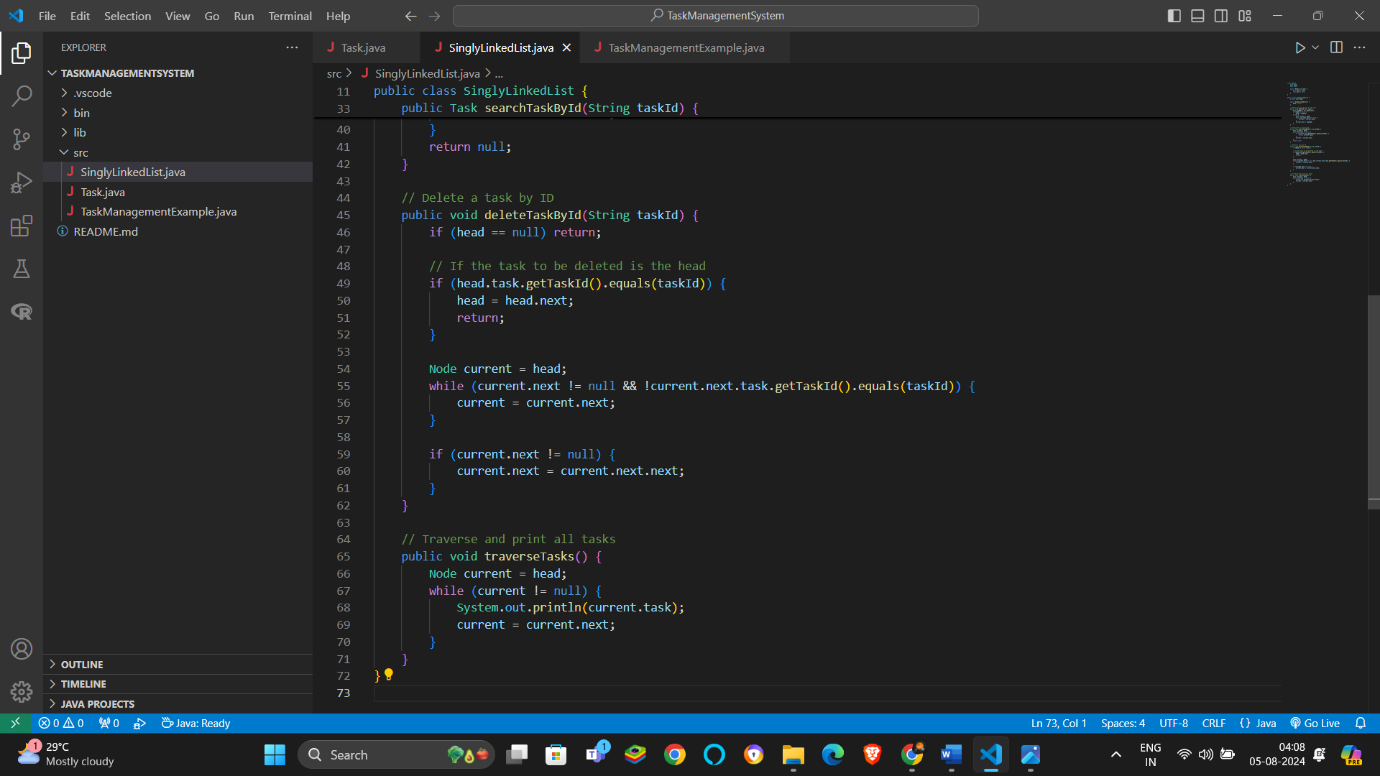
* **Static Datasets**: Use arrays when the size of the dataset is known and does not change frequently.
* **Simple Data Storage**: Use arrays when simple and fast access is required and the operations are straightforward.

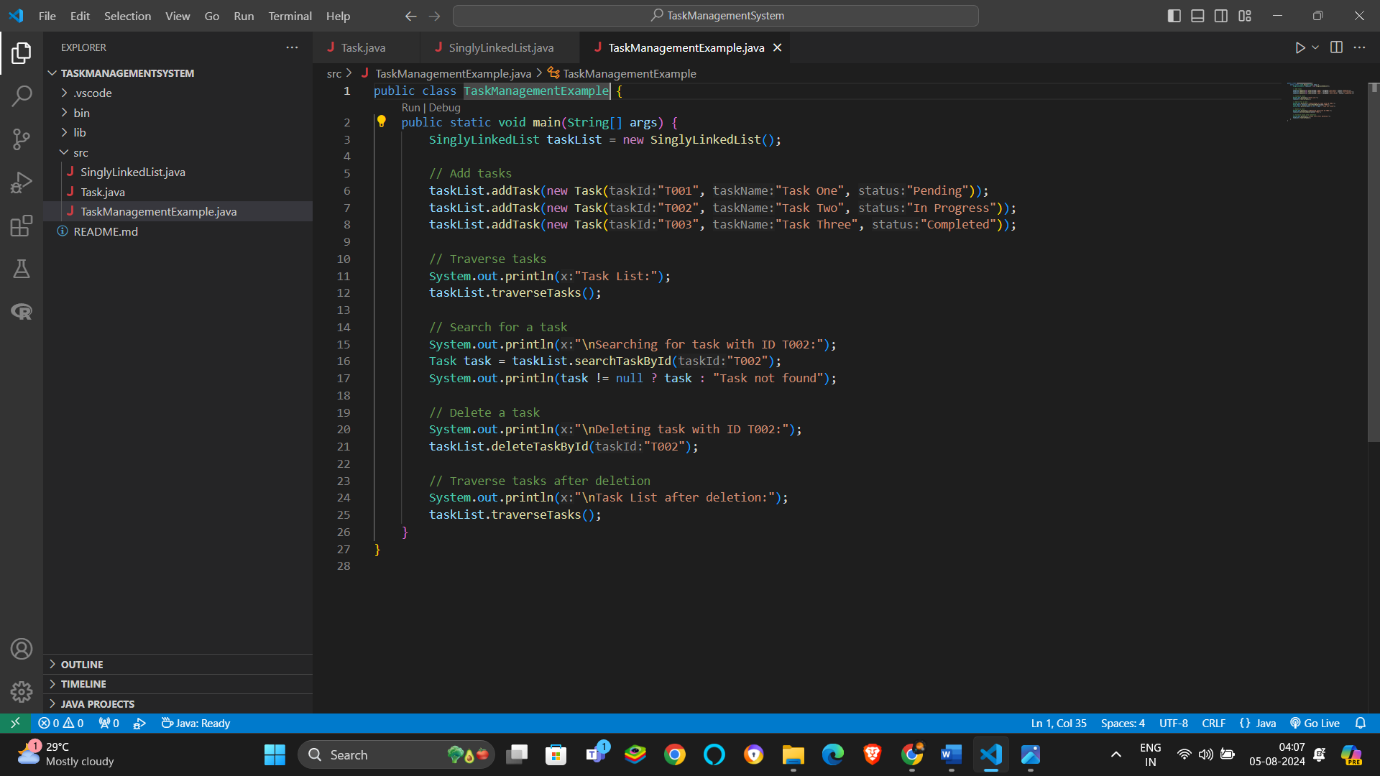
Exercise 5-

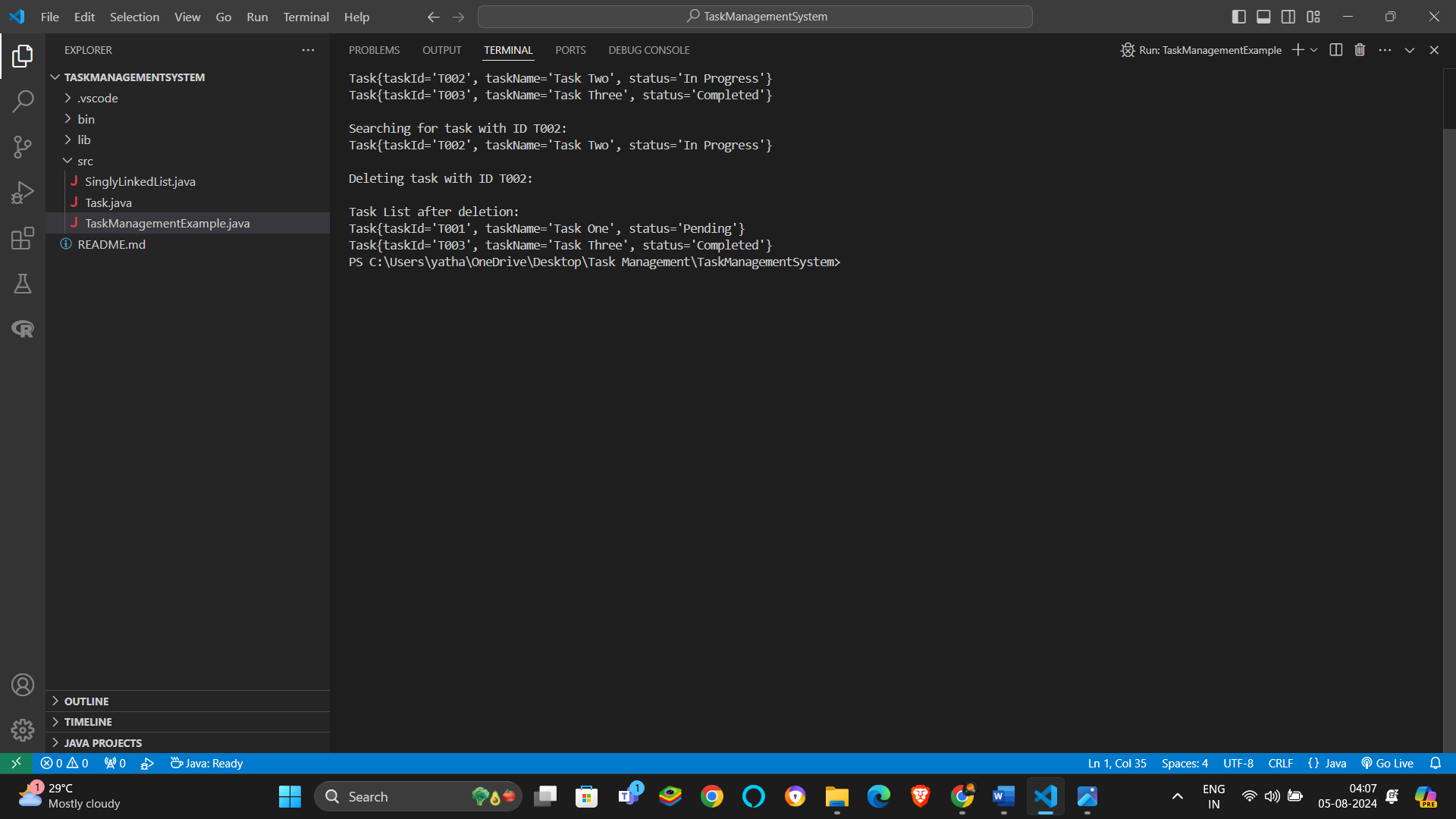












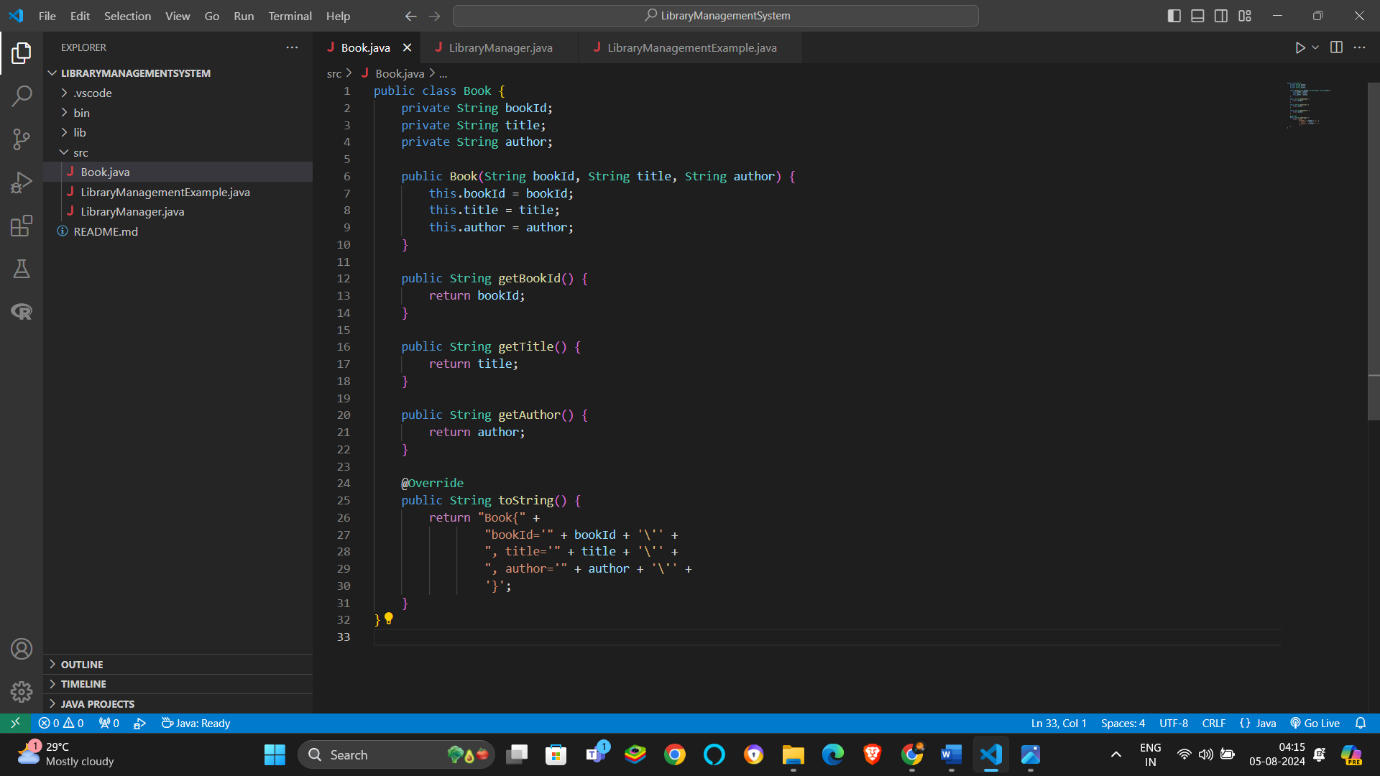
**Time Complexity Analysis:**

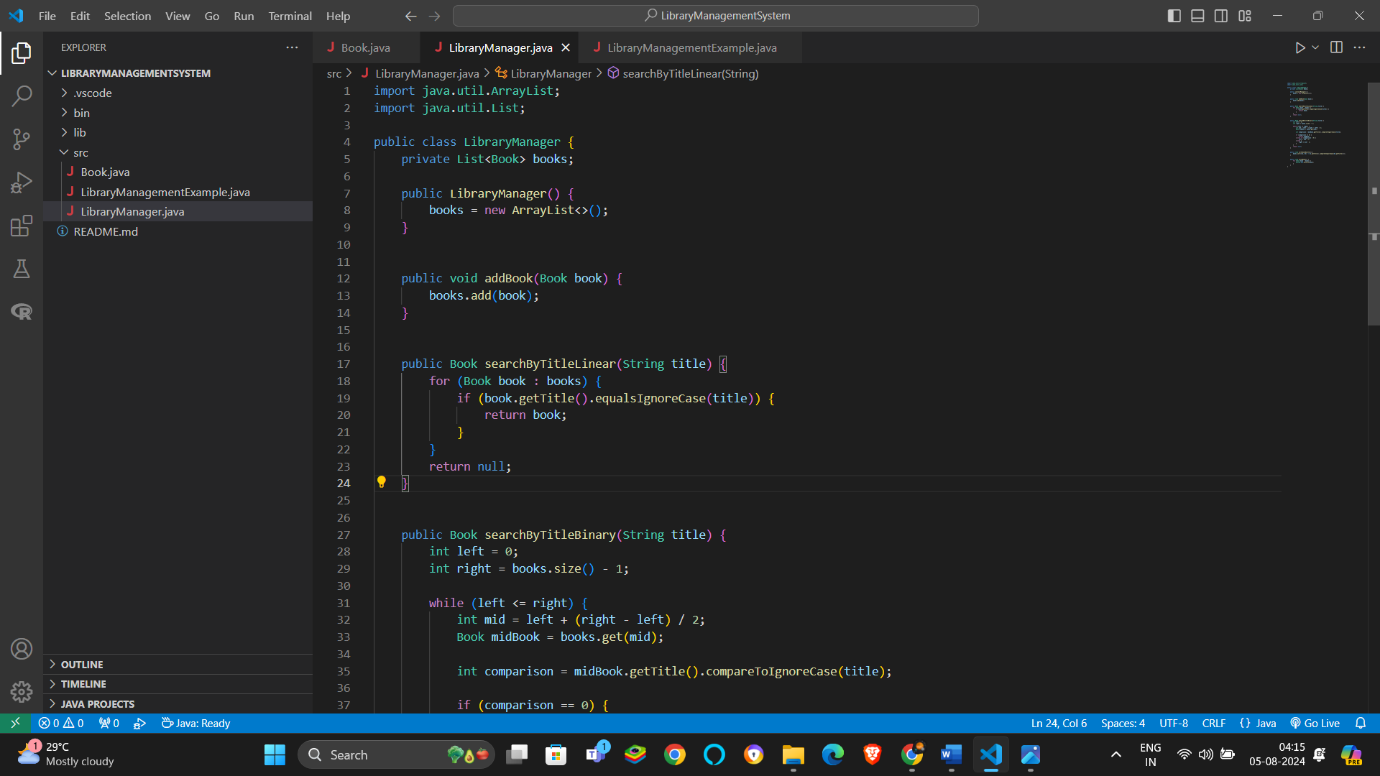
* **Add Task**: O(n)
* **Search Task**: O(n)
* **Delete Task**: O(n)
* **Traverse Task**: O(n)

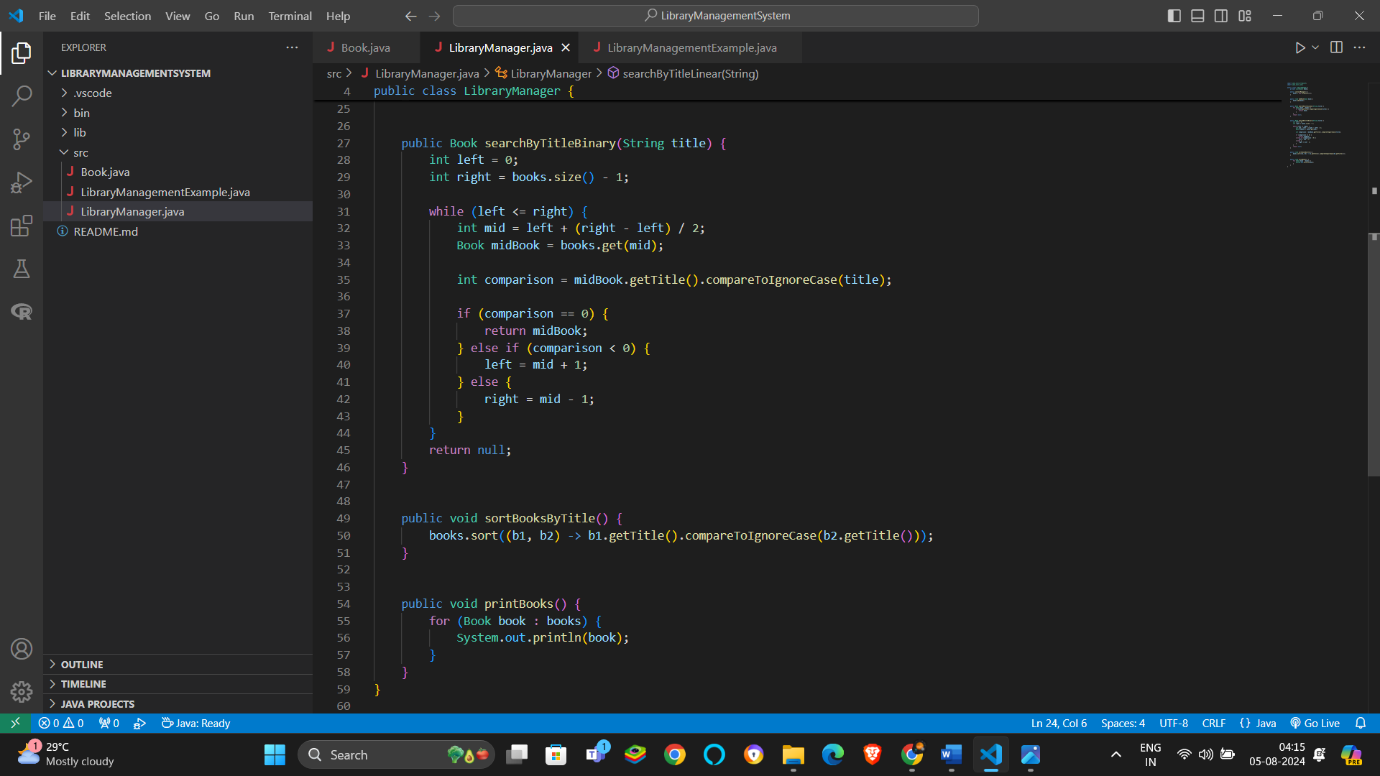
**Advantages of Linked Lists Over Arrays**:

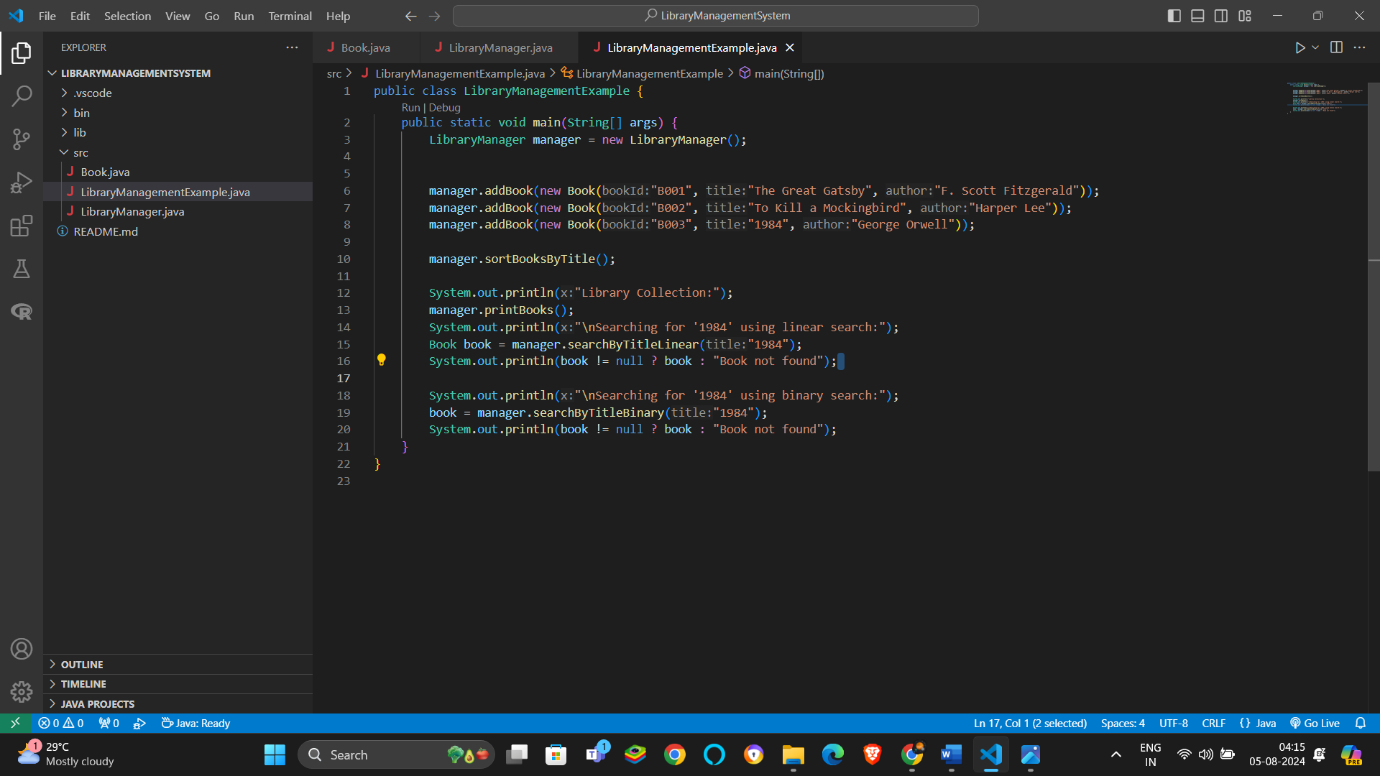
* **Dynamic Size**: Linked lists can easily grow or shrink without reallocating or resizing.
* **Efficient Insertions/Deletions**: Linked lists allow efficient insertions and deletions at the head or middle of the list.
* **Memory Utilization**: No need to allocate a large block of contiguous memory as with arrays.

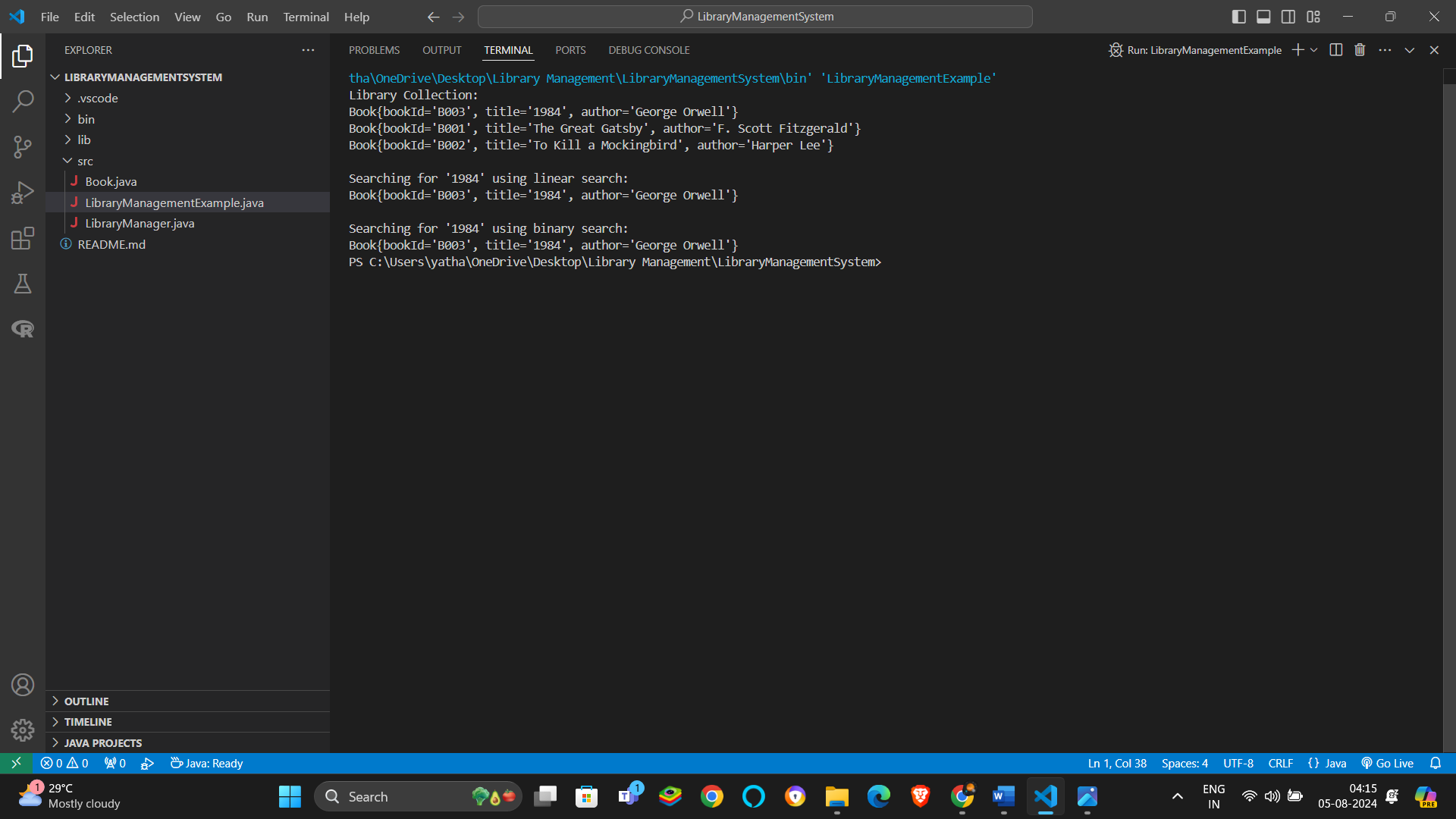
Exercise 6-







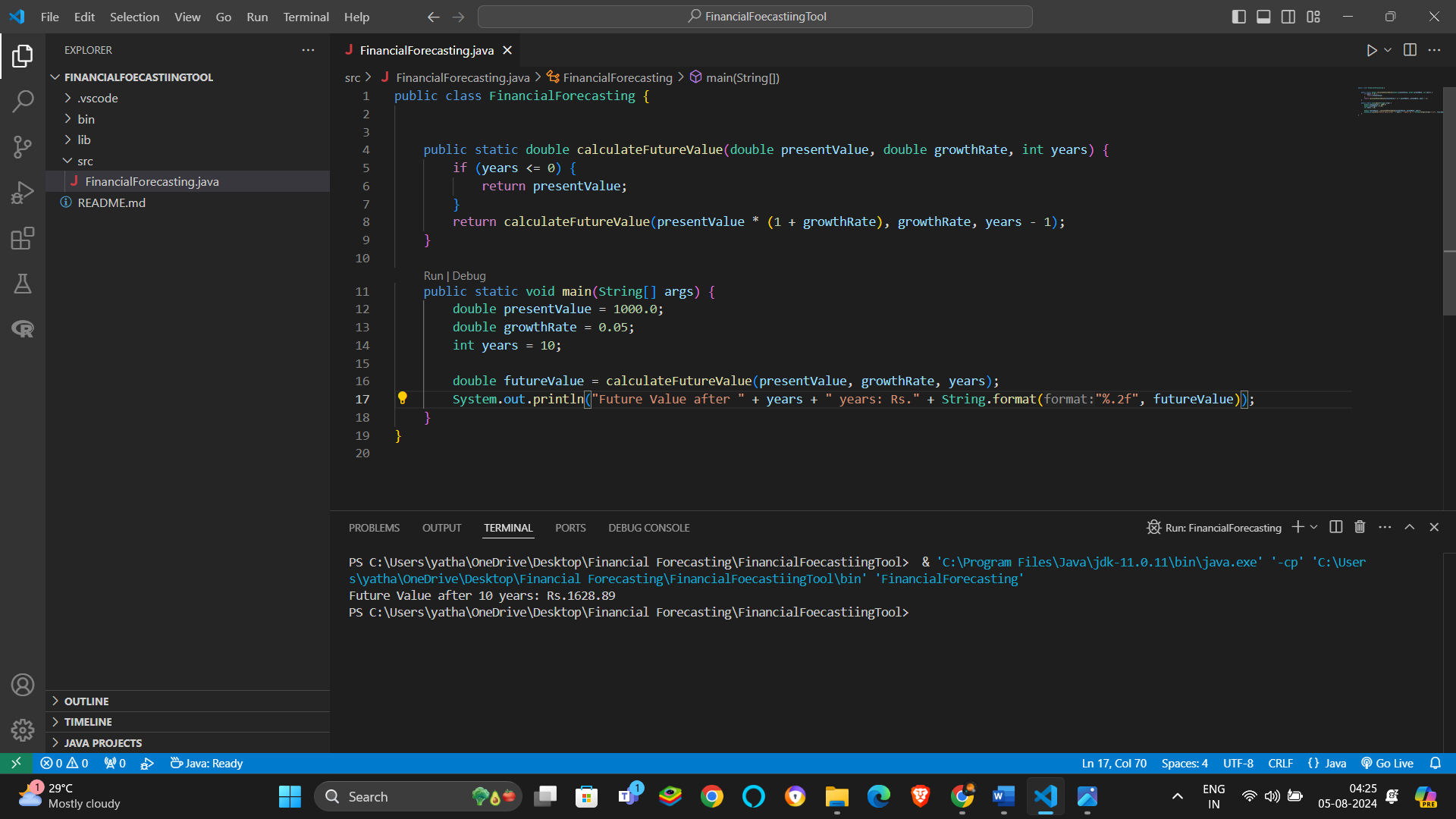




The worst case Time Complexity of Linear Search in O(n) and that of Binary Search is O(log n).

Linear Search is beneficial in use when the data set is samll and unsorted while Binary Search is useful for large and sorted data sets.

Exercise 7-



The Time Complexity of this recursive algorithm will be O(n) where n is the number of years.

To Optimize this solution we can follow an iterative approach instead of using a recursive function.