**Answers:**

* Explain Big O notation and how it helps in analyzing algorithms.

Big O notation describes the **upper bound** of an algorithm's time or space complexity in terms of input size n. It helps analyse how the algorithm performs as the input grows, ignoring constant factors.

* Describe the best, average, and worst-case scenarios for search operations.

Best Case:

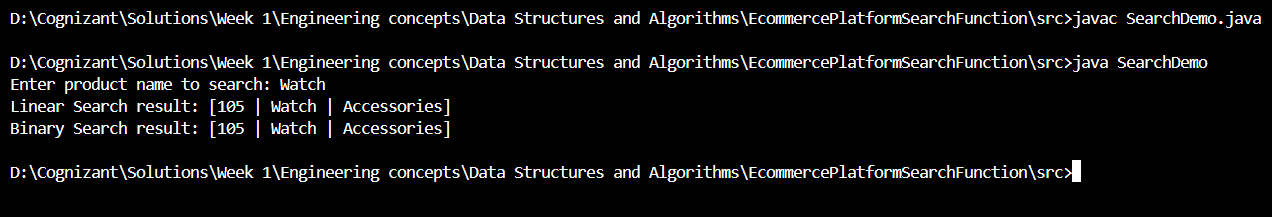
* The target element is found in the first few operations.
* Minimal computation is required.
* Time Complexity: Often O(1).

Average Case:

* The target element is located somewhere in the middle of the dataset.
* Represents a typical or expected scenario.
* Time Complexity: Depends on the algorithm, commonly O(n) or O(log n).

Worst Case:

* The target element is not present or is found after checking the entire/maximum portion of the dataset.
* Represents the most time-consuming situation.
* Time Complexity: Usually O(n) for linear approaches, O(log n) for optimized algorithms like binary search.
* Output of Code:



* Compare the time complexity of linear and binary search algorithms.

**Linear Search:**

* Best Case:
* Element is at the first position
* Time Complexity: O(1)
* Average Case:
* Element is somewhere in the middle
  + Time Complexity: O(n/2) → Simplified to O(n)
* Worst Case:
  + Element is at the last position or not present
  + Time Complexity: O(n)

**Binary Search *(on sorted data)*:**

* Best Case:
  + Element is at the middle of the array
  + Time Complexity: O(1)
* Average Case:
  + Element is somewhere in the array
  + Time Complexity: O(log n)
* Worst Case:
  + Element is not present, or found after maximum splits
  + Time Complexity: O(log n)
* Discuss which algorithm is more suitable for your platform and why.

**Binary search** is more suitable for an e-commerce platform because it is **faster (O(log n))**, **scales well with large datasets**, and works efficiently on **sorted product lists**, which are common in such systems.