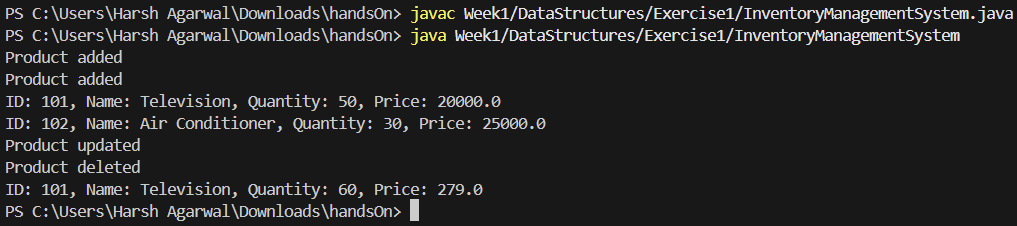
Output for Data Structures And Algorithms

Exercise 1:

Why data structures and algorithms are essential? They enable efficient storage, quick retrieval, fast updates, and optimized searching within a large inventory, which directly impacts system performance and user experience.

Discuss the types of data structures suitable for this problem. ArrayList and HashMap can be used for this problem.

Time Complexities: Add: O(1), Update: O(1), Delete: O(1)

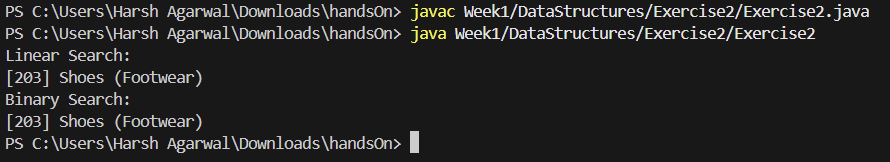


Exercise 2:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Best Case | Average Case | Worst Case |
| Linear Search | O(1) | O(n) | O(n) |
| Binary Search | O(1) | O(log N) | O(log N) |

Big O Notation: Big O notation describes the upper bound of an algorithm's running time in terms of input size n.

Binary search is more suitable for the e-commerce platform if products are sorted (e.g., alphabetically or by ID).For dynamic or unsorted product data, linear search might be necessary unless real-time sorting or indexing is applied



Exercise 3:

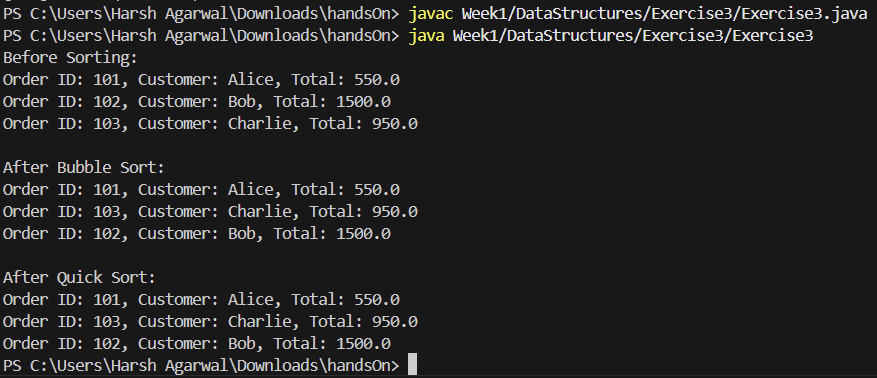
Bubble Sort: Repeatedly swaps adjacent elements if they are in the wrong order. Simple but inefficient for large datasets.

Insertion Sort: Builds the sorted array one element at a time by inserting elements into their correct position. Efficient for small or nearly sorted arrays.

Quick Sort: A divide-and-conquer algorithm. Chooses a pivot, partitions the array into elements less than and greater than the pivot, then recursively sorts the partitions.

Merge Sort: Another divide-and-conquer algorithm. Recursively splits the array and merges sorted halves. Guarantees O(n log n) time but uses extra memory.

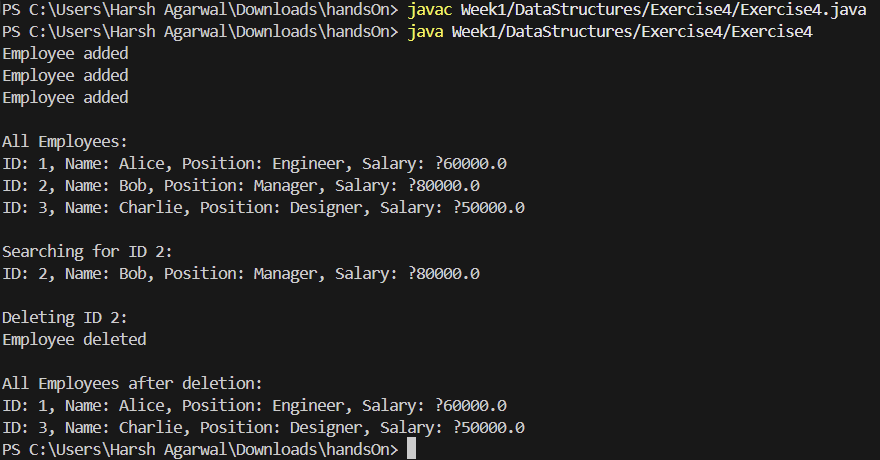
|  |  |  |  |
| --- | --- | --- | --- |
|  | Best Case | Average Case | Worst Case |
| Bubble Sort | O(N) | O(N2) | O(N2) |
| Quick Sort | O(N logN) | O(N logN) | O(N2) |



Exercise 4:

Time Complexity of Operations (in an array):

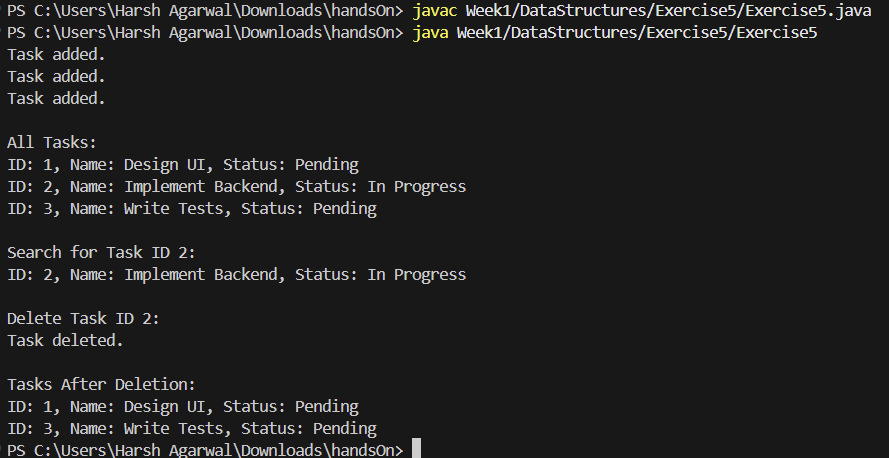
* Add (at end): O(1)
* Search: O(n)
* Traverse: O(n)
* Delete (by value or index): O(n) — due to shifting elements after deletion.



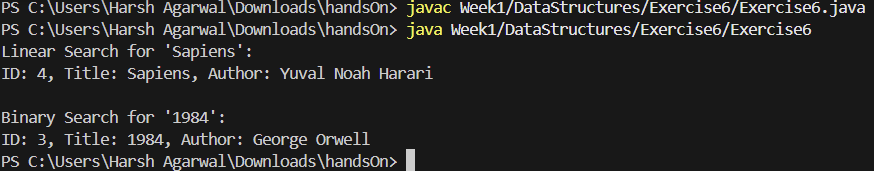
Exercise 5:

Singly Linked List: Each node contains data and a reference to the next node. Can only be traversed forward. Simpler and uses less memory than doubly linked lists.

Doubly Linked List**:** Each node contains data, a reference to the **next** node, and a reference to the previous node. Allows bidirectional traversal. Requires more memory due to extra pointer but offers more flexibility.



Exercise 6:



Exercise 7:

Recursion: Recursion is a technique where a function calls itself to solve a smaller instance of the same problem. It simplifies problems that have a repetitive or self-similar structure. A recursive method typically has:

* A base case to stop recursion.
* A **rec**ursive case that reduces the problem size.

Time Complexity of Recursive Algorithm:

futureValue(n) = growthRate \* futureValue(n - 1)

Optimization Techniques:

* Memoization: Store previously computed results to avoid redundant calculations.
* Tabulation: Use an iterative, bottom-up approach to fill values in a table.
* Convert to Iterative: In many cases, recursion can be converted to a loop to save memory and avoid stack overflow.

