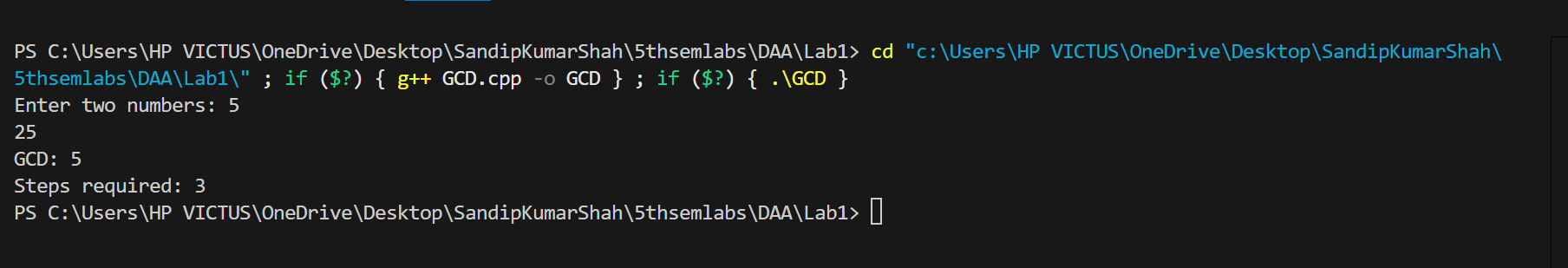
**//GCD:**

**Analysis of GCD:**

**Time Complexity:** O(log(min(a, b))) where a and b are the two numbers.

**Space Complexity:** O(log(min(a, b))) due to the recursive call stack.

**Output:**

****

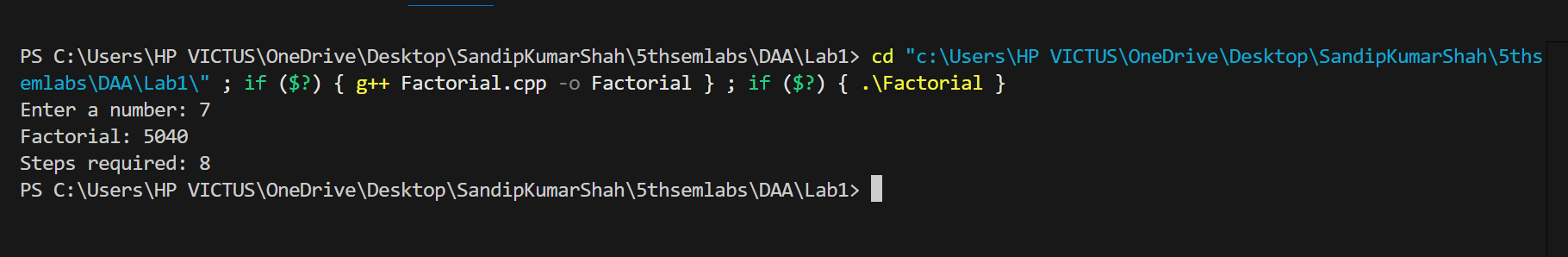
**//Factorial:**

**Analysis of Factorial:**

**Time Complexity:** O(n).

**Space Complexity:** O(n).

**Output:**

****

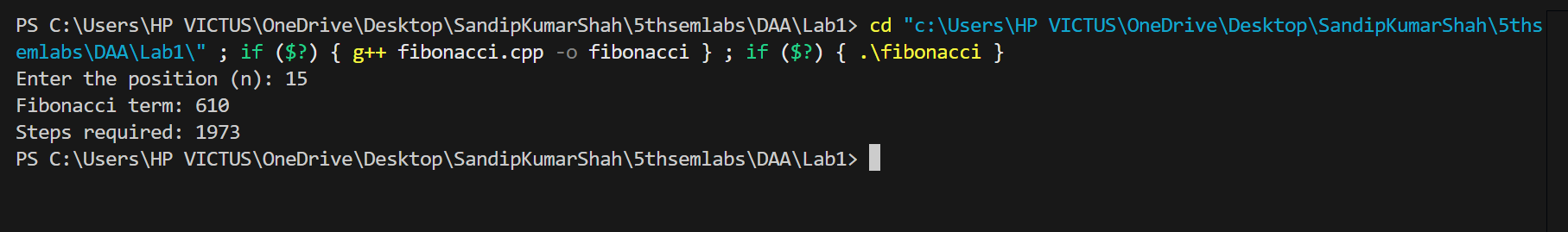
**//Fibonacci:**

**Analysis of Fibonacci:**

**Time Complexity:** O(2n).

**Space Complexity:** O(n).

**Output:**

****

**//Bubble Sort:**

**Analysis of Bubble Sort:**

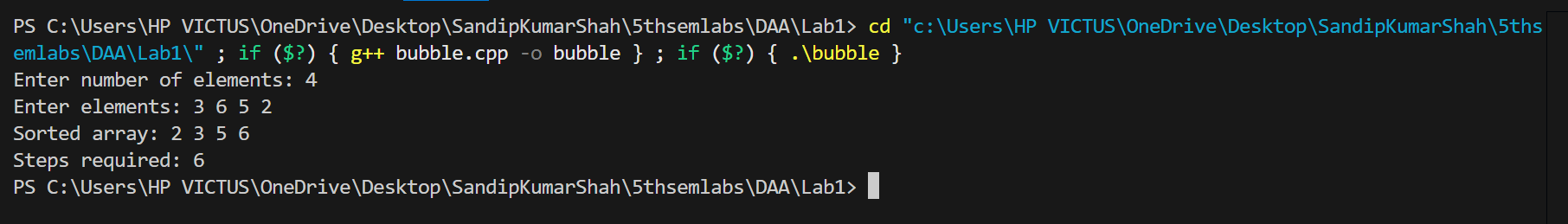
**Time Complexity:**

* Best case (already sorted): O(n)
* Worst case (reverse order): O(n^2)
* Average case: O(n^2)
  + Bubble sort compares adjacent elements in each pass and swaps them if needed.

**Space Complexity:**

* O(1) since bubble sort is an in-place sorting algorithm.

**Output:**

****

**//Insertion Sort:**

**Analysis of Insertion Sort:**

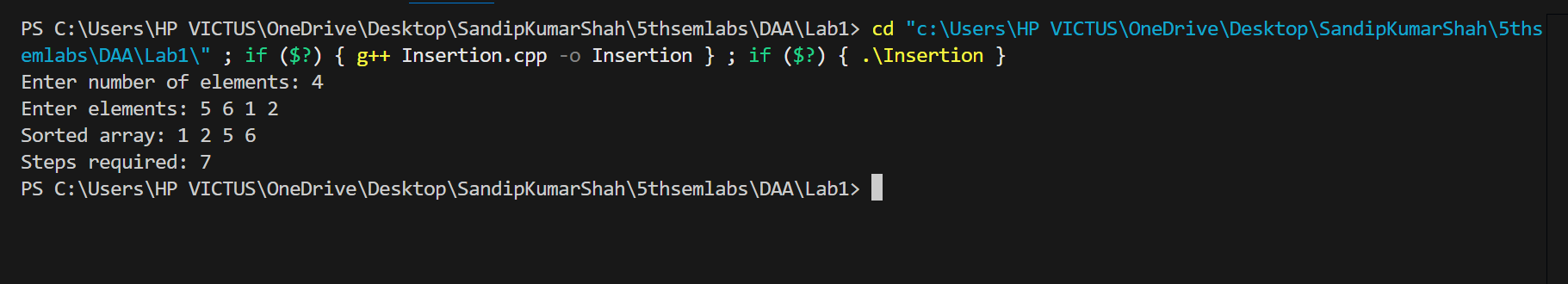
**Time Complexity:**

* Best case (already sorted): O(n)
* Worst case (reverse order): O(n^2)
* Average case: O(n^2)
  + Insertion sort builds the sorted array one element at a time.

**Space Complexity:**

* O(1) since insertion sort is an in-place sorting algorithm.

**Output:**

****

**//Selection Sort:**

**Analysis of Selection Sort:**

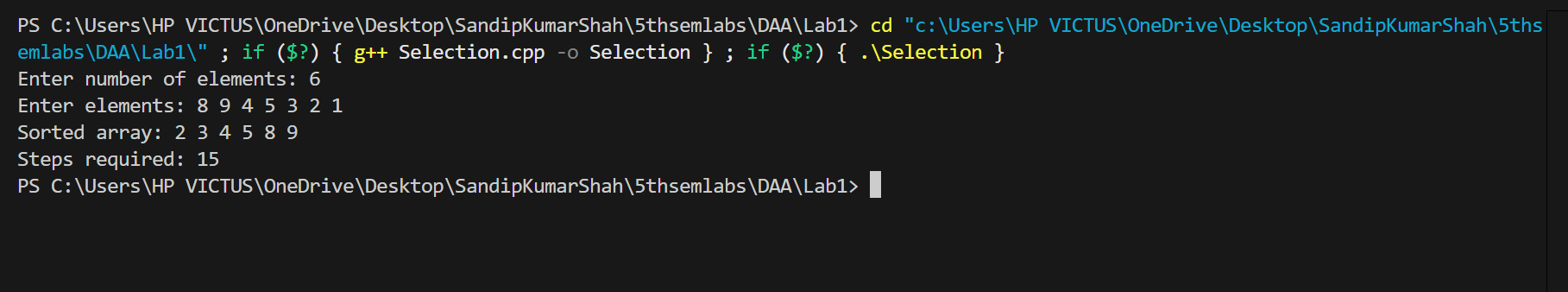
**Time Complexity:**

* Best case: O(n^2)
* Worst case: O(n^2)
* Average case: O(n^2)
  + Selection sort iterates through the array multiple times, selecting the minimum (or maximum) and swapping.

**Space Complexity:**

* O(1) since selection sort is an in-place sorting algorithm.

**Output:**

****

**Conclusion:**

Each algorithm demonstrates its logic and execution steps effectively. Recursive algorithms like GCD, Factorial, and Fibonacci involve repeated function calls, with Fibonacci having the highest step count due to overlapping subproblems. Sorting algorithms differ in efficiency, with insertion and selection sorts showing linear and quadratic time complexities in small datasets.