**//BST:**

**Analysis of BST:**

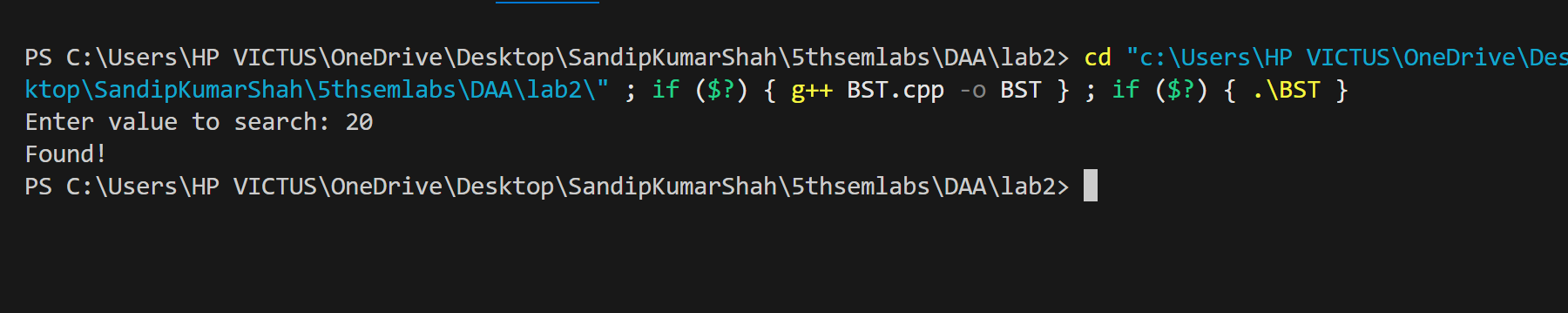
**Time Complexity:**

* In the best case (balanced BST), the time complexity is O(log n).
* In the worst case (unbalanced, resembling a linked list), the time complexity is O(n).

**Space Complexity:**

* O(log n) in the best case (balanced tree) due to the recursive stack.
* O(n) in the worst case (unbalanced tree).

**Output:**

****

**//MINMAX:**

**Analysis of MINMAX:**

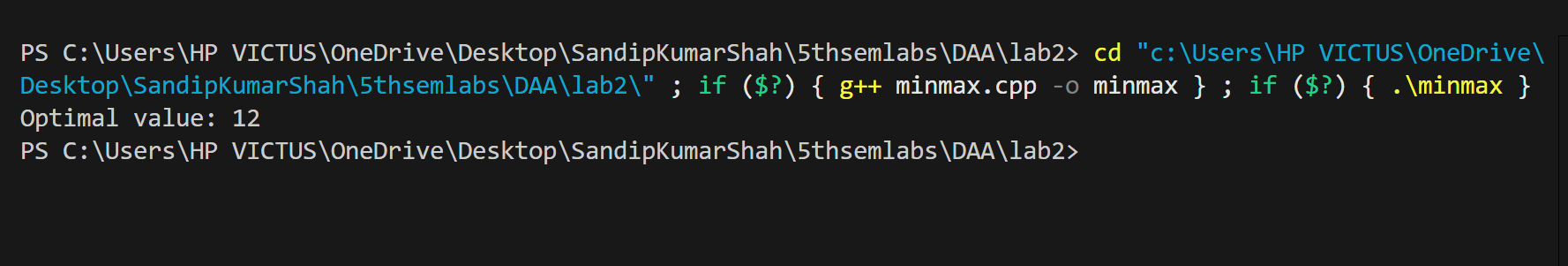
**Time Complexity:**

* The time complexity is O(n) because each element of the array is visited once to find the minimum and maximum.

**Space Complexity:**

* O(1) since only a few variables are used to store the min and max values.

**Output:**

****

**//Merge Sort:**

**Analysis of Merge Sort:**

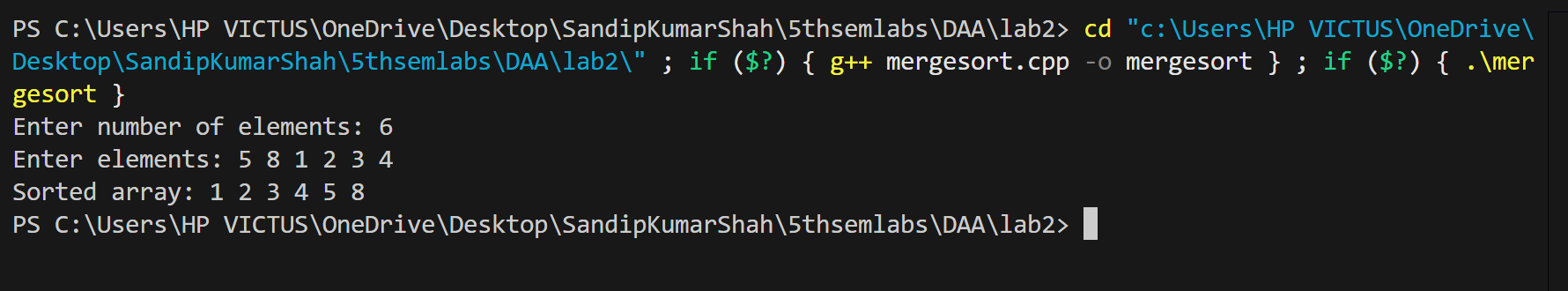
**Time Complexity:**

* Best case, worst case, and average case all have a time complexity of O(n log n), where n is the number of elements in the array.
  + This is because the array is divided in half at each recursive step (log n), and then each element is processed during the merge step (n).

**Space Complexity:**

* O(n) because of the temporary arrays used in the merge process.

**Output:**

****

**Conclusion:**

The Binary Search Tree (BST) is an efficient data structure for dynamic searching, insertion, and deletion, offering O(log n) time complexity in the best case but potentially degrading to O(n) in unbalanced trees. Merge Sort is a stable and efficient sorting algorithm with a time complexity of O(n log n), ideal for large datasets, but requires O(n) extra space. The Min-Max Algorithm efficiently finds the minimum and maximum elements in an array in O(n) time and O(1) space, making it a simple and space-efficient solution for determining bounds in datasets. Each algorithm serves distinct purposes, balancing time and space efficiency depending on the application.