**MSCS532 Assignment 2 – Divide and Conquer Algorithms**

Hajeera Hajeera

University of Cumberlands

MSCS-532-M80: Algorithms and Data Structures

Dr. Satish Penmatsa

09/21/2025

**MSCS532 Assignment 2 – Divide and Conquer Algorithms**

GitHub Repository: <https://github.com/>[hahajeera/MSCS532\_Assignment1](https://github.com/hahajeera/MSCS532_Assignment1)

Computer science has long relied on problem-solving strategies that break large challenges into smaller, more manageable parts. Among the most well-known strategies is the divide and conquer approach. This paradigm works by dividing a task into sub-problems, solving those independently, and then combining the results to reach a complete solution

In practice, divide and conquer provides efficiency where straightforward methods would become too slow. Examples appear in daily life: delivery companies divide regions into smaller areas to optimize routes, or tournament organizers split competitors into brackets until a champion emerges. In computing, divide and conquer powers search engines, database queries, and image compression systems.

For this assignment, two algorithms were implemented using the divide and conquer method: Binary Search (iterative version) and Merge Sort (alternative structure). These implementations highlight how the same strategy can be adapted to both searching and sorting problems.

**Part 1 – Algorithms Overview**

## Binary Search (Iterative Version)

Binary Search is a highly efficient way to locate an element in a sorted list. Instead of checking each item one by one, the algorithm repeatedly splits the list in half until the target value is found or the search space is empty. The iterative version avoids recursion by using a loop to adjust the search boundaries.

This approach is commonly seen in real life when looking up a word in a dictionary. Rather than starting at page one, a reader opens the book in the middle and narrows down the location based on alphabetical order. Similarly, in computing, binary search is widely applied in databases and memory management.

## Merge Sort (Alternative Version)

Merge Sort is a sorting algorithm that divides a list into smaller parts, sorts each individually, and then merges them into a fully sorted sequence. Unlike simpler methods such as bubble sort, merge sort maintains efficiency even with large datasets, operating with a time complexity of O(n log n).

In practical scenarios, merge sort reflects how information is organized in libraries or digital catalogs. For example, books might be separated by subject, then by author, and finally by title before being merged into an ordered collection. The alternative structure implemented here uses slightly different variable names and formatting than standard merge sort, but the logic remains consistent.

# Part 2 – Implementation and Testing

To evaluate the algorithms, Python was used to implement and execute both binary search and merge sort. The testing involved initial runs with base data as well as modified runs with altered input values. The results confirmed correctness and demonstrated how divide and conquer improves efficiency.

## Testing Results

**1. Binary Search – Initial Test**

**Input:**

Array = [2, 4, 6, 8, 10, 12, 14]

Target = 10

**Output:**

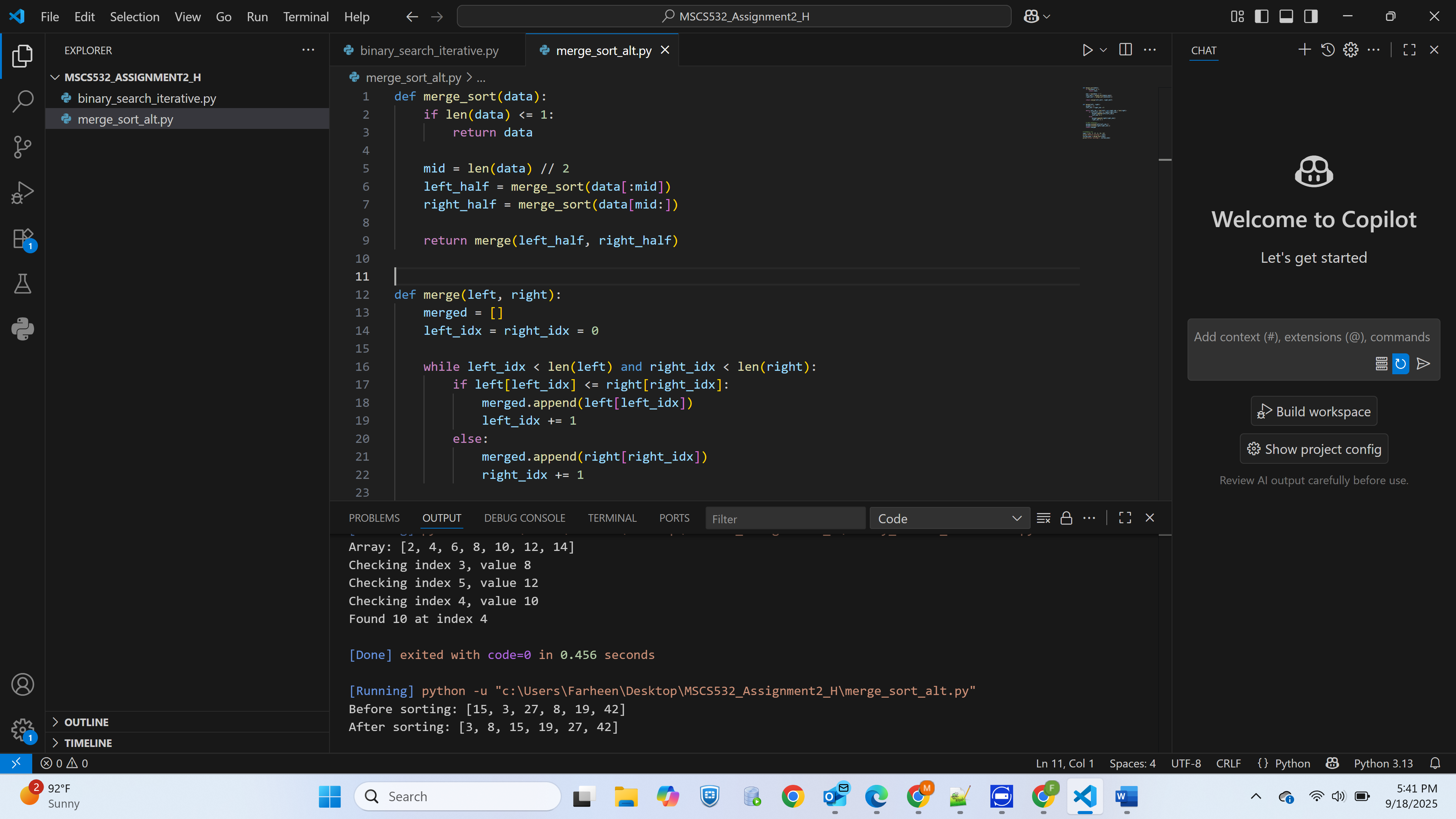
Array: [2, 4, 6, 8, 10, 12, 14]

Checking index 3, value 8

Checking index 5, value 12

Checking index 4, value 10

Found 10 at index 4



**2. Merge Sort – Initial Test**

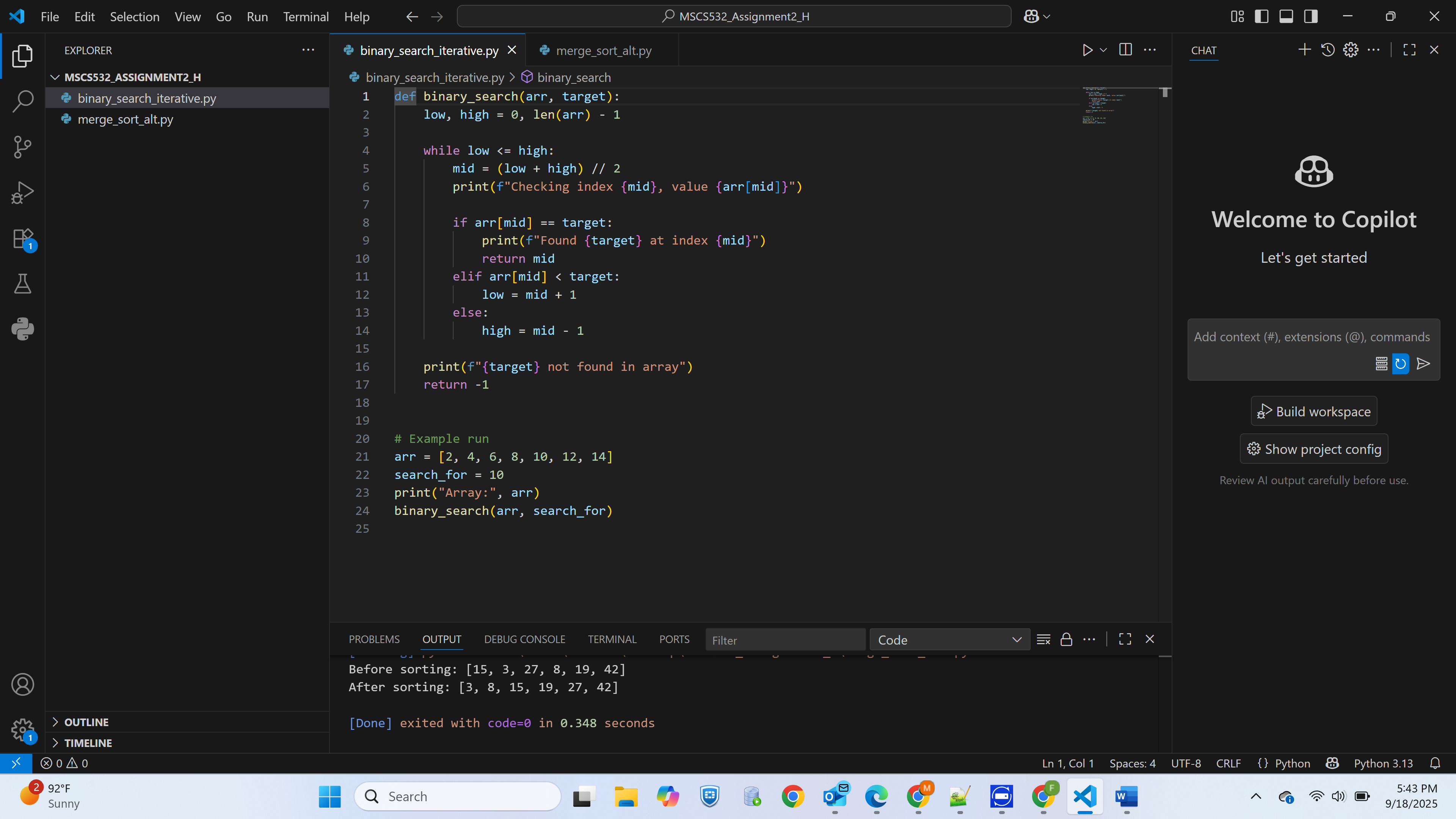
**Input:**

[15, 3, 27, 8, 19, 42]

**Output:**

Before sorting: [15, 3, 27, 8, 19, 42]

After sorting: [3, 8, 15, 19, 27, 42]



**3. Binary Search – Modified Test**

**Input:**

Array = [2, 4, 6, 8, 10, 12, 14]

Target = 7

**Output:**

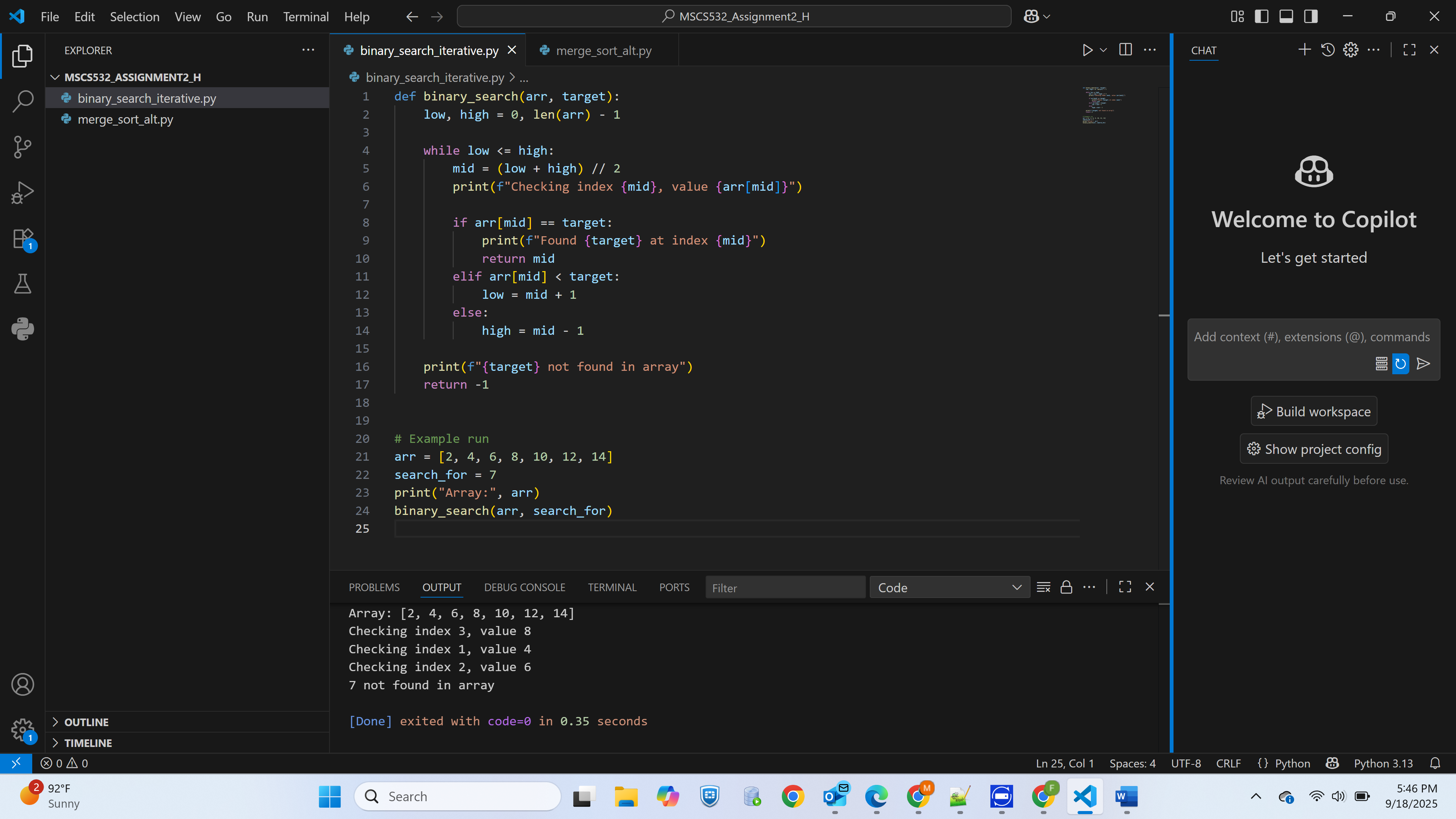
Array: [2, 4, 6, 8, 10, 12, 14]

Checking index 3, value 8

Checking index 1, value 4

Checking index 2, value 6

7 not found in array



**4. Merge Sort – Modified Test**

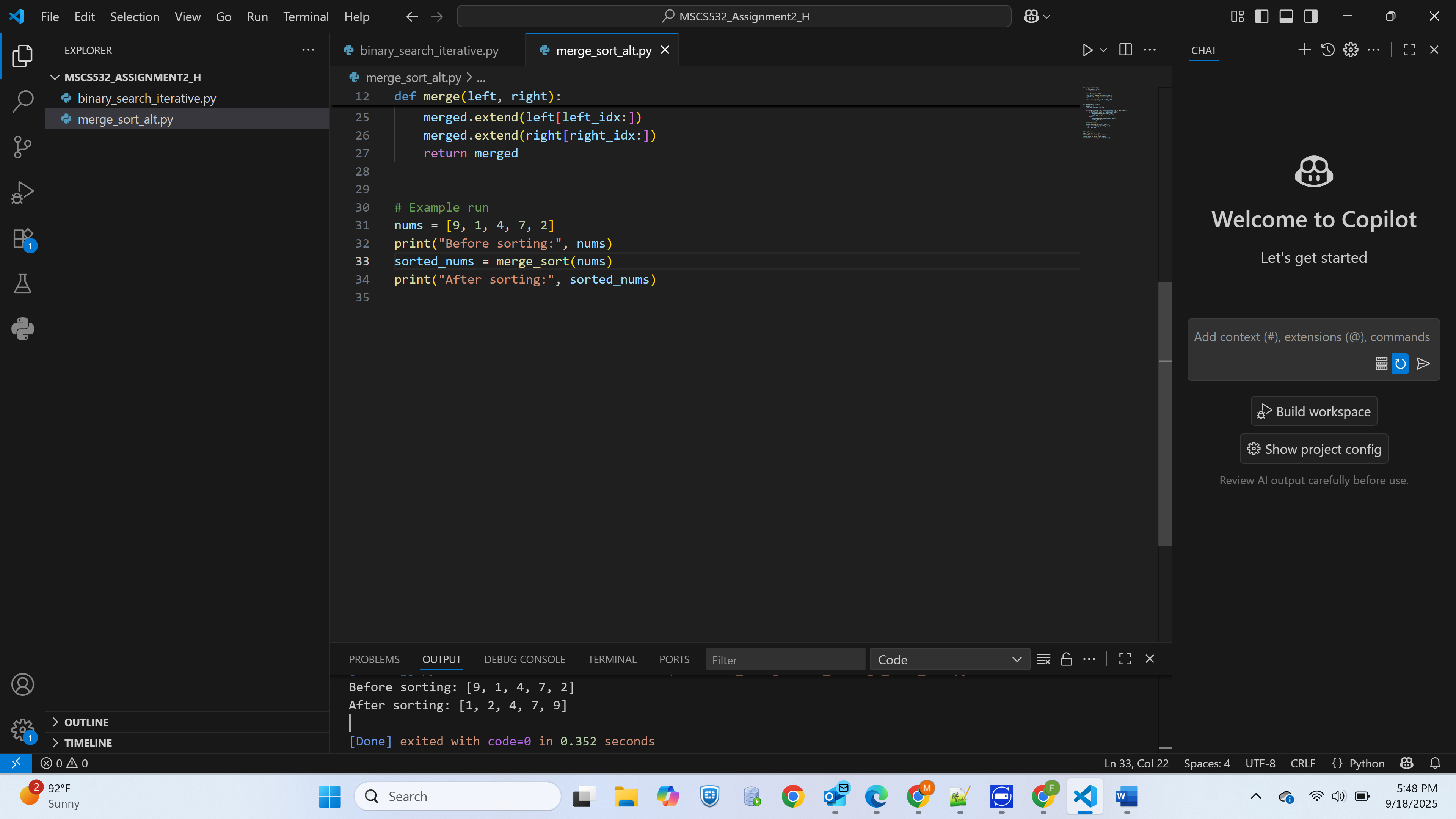
**Input:**

[9, 1, 4, 7, 2]

**Output:**

Before sorting: [9, 1, 4, 7, 2]

After sorting: [1, 2, 4, 7, 9]



## Conclusion

This assignment highlighted the versatility of the divide and conquer strategy by applying it to both searching and sorting problems. The iterative binary search showcased how breaking down a search space into halves leads to quick results, while the alternative merge sort demonstrated efficient organization of unordered data. Together, these algorithms reinforce the principle that dividing large challenges into smaller pieces not only simplifies the process but also enhances performance. Beyond academic study, these methods mirror strategies used in logistics, decision-making, and information organization in daily life.

By exploring different implementations, it becomes clear that divide and conquer is not a single algorithm but a guiding philosophy for solving complex problems. Its origin in mathematical reasoning has grown into a powerful tool that continues to drive progress in computing today.

# Reference

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to algorithms* (4th ed.). The MIT Press.

Goodrich, M. T., & Tamassia, R. (2015). *Algorithm design and applications*. Wiley.

Kleinberg, J., & Tardos, É. (2006). *Algorithm design*. Pearson Education.

Knuth, D. E. (1998). *The art of computer programming, Volume 3: Sorting and searching* (2nd ed.). Addison-Wesley.

Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley Professional.

GeeksforGeeks. (2023, June 15). *Divide and conquer algorithm*. GeeksforGeeks. https://www.geeksforgeeks.org/divide-and-conquer-algorithm/