## Time Complexity Analysis

The reverseArray function operates with a while loop that runs approximately n/2 times, where n is the size of the input array. Each iteration involves a constant number of steps—specifically, five assignments and operations—resulting in a total complexity of Treverse = 5n/2 . On the other hand, the bubbleSort function consists of a nested loop structure where the outer loop runs n−1n times and the inner loop varies from n−1 down to 1, leading to a total of ( n – 1 ) / 2 comparisons in the worst-case. Each comparison may involve up to four steps (one comparison and three assignments in case of a swap), culminating in a complexity of Tbubble = 2( n2 – n ). Therefore, the overall time complexity of the program is dominated by the bubble sort, resulting in O(n2).

## Space Complexity Analysis

In terms of space complexity, both functions utilize a constant amount of memory that does not scale with the input size. The reverseArray function requires space for three integer variables: start, end, and temp. Similarly, the bubbleSort function requires space for the loop control variables i, j, and the Boolean flag swapped. Thus, both functions exhibit a space complexity of O(1).

## Summary

In summary, the time complexity of the program is O(n2) in the worst case due to the bubble sort algorithm, while the space complexity remains O(1) as the memory usage is constant, irrespective of the input size. This analysis highlights the efficiency limitations of bubble sort compared to more advanced sorting algorithms.