Analysis and Approach:

For the Two Sum problem, my approach is based on using a hash map (or dictionary) to efficiently find pairs of numbers that add up to the target. Instead of using a nested loop to check every pair (which would have a higher time complexity), I opted to leverage the hash map's fast lookup capabilities. This allows us to store each number's index as we iterate through the list. By checking if the complement (the number needed to reach the target) is already in the map, we can immediately find the required pair when it exists.

Code Explanation:

1. **Initialize a hash map:** I start by creating an empty dictionary (hash_map = {}) to store each number and its index as I process the list.
2. **Iterate through the list:** For each element in nums, I calculate the "match pair" or the number needed to reach the target when added to the current element.
3. **Check the hash map for the match pair:** If the match pair is already in hash_map, this means we have previously encountered the complement of the current number, and they add up to the target. I then return the indices of these two numbers.
4. **Store the current number and index in the hash map:** If the match pair is not found, I add the current number and its index to hash_map and proceed to the next iteration. This allows me to build a reference of each number's position for future checks.

This approach allows the solution to be achieved in a single pass through the list.

Time and Space Complexity:

- **Time Complexity:** O(n) – In this approach, I only iterate through the list once. For each element, checking if its complement exists in the hash map takes constant time, O(1), since dictionary lookups are O(1) on average. Thus, the overall time complexity is O(n), where n is the length of the input list.
- **Space Complexity:** O(n) – I use a hash map to store each unique number and its index. In the worst case, where no two numbers sum up to the target, the hash map will store every element in the list. Hence, the space complexity is O(n), as the dictionary grows with the size of the input list.