There are multiple ways to solve this problem.

1. Do two loops to find the pair of numbers which match the targer. O(n^2) Solution.
2. Sort the elements and then do a front and back look up with two indexes i and j to see if they add upto the target. When A[i] + A[j] > target then, decrement j and of their sum is < target then increment i. When the sum is equal to target, we found the solution. This algorithm takes O(n \* log(n)) time. The dominating factor is the sorting. The pair finding is done in O(n) time.
3. Use MAPs to find out of the current element we are processing can sum upto a target among the elements which we’ve already seen. Just map the location of a number to its index. When going through the array elements, check if *target – A[i]* already exists in the map. If Yes, then it means that we’ve already seen a number whose *value* + *A[i]* = target. If No, then insert the element in to the map. This algorithm, with O(1) implementation of MAPs runs in *O(n) time*.