Three Number Sum

Input: array = [12, 3, 1, 2, -6, 5, -8, 6]
target Sum = 0

Output: [[-8, 2, 6], [-8, 3, 5], [-6, 1, 5]]

Input: A non-empty array of distinct integers and
An integer representing a target sum

Output: All triplets in the array that sum up the target sum returned as a 2D array
Triplets should be ordered in ascending order
return empty array if no three numbers are found

```
// O(n^2) time | O(n) space
function threeNumberSum(array, targetSum) {
  array.sort((a, b) => a - b);
  const triplets = [];
  for (let i = 0; i < array.length - 2; i++) {
    let curr = i;
    let left = i + 1;
    let right = array.length - 1;
    while (left < right) {
      const currentSum = array[curr] + array[left] + array[right];
      if (currentSum === targetSum) {
        triplets.push([array[curr], array[left], array[right]]);
        left++;
        right--;
      } else if (currentSum < targetSum) {
        left++;
      } else if (currentSum > targetSum) {
        right--;
      }
    }
  }
  return triplets;
}
```

Idea: iterate through array and at each element, use pointers to compare values

Note: We go to array.length - 2 because when we are at the 3rd last element, we assign left and right pointers to 2nd last and last elements respectively. If they do not match the targetSum, we know there is no more values to check

Time: $O(n^2)$ (where n is the length of the input array) since at each element, we iterate through the remaining elements

Space: $O(n)$ since we are bounded by the # of elements in the input array. If they all meet the target sum, triplets will be at worst length n