1. Two Sum

Given an array of integers, return indices of the two numbers such that they add up to a specific target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

Example 1:

```
Given nums = [2, 7, 11, 15], target = 9,

Because nums[0] + nums[1] = 2 + 7 = 9,
return [0, 1].
```

```
class Solution {
    public int[] twoSum(int[] nums, int target) {

        // two pointers
        int[] sorted = new int[nums.length];
        for (int i = 0; i < nums.length; i++) {
            sorted[i] = nums[i];
        }
        Arrays.sort(sorted); // O(Nlog(N))

        int start = 0;
        int end = nums.length-1;

        while(start < end) {
            if (sorted[start] + sorted[end] < target) {
                start++;
            } else if (sorted[start] + sorted[end] > target) {
                end--;
            } else {
                break;
            }
        }

        int index1 = -1;
        int index2 = -1;

        for (int i = 0; i < nums.length; i++) {
            if (nums[i] == sorted[start] || nums[i] == sorted[end]) {
                if (index1 == -1) {
                    index1 = i;
                } else {
                    index2 = i;
                }
            }
        }

        int[] res = {index1, index2};
        Arrays.sort(res);
        return res;
    }
}
```

```
class Solution {
    public int[] twoSum(int[] nums, int target) {

        // hashmap
        Map<Integer, Integer> map = new HashMap<>();

        for (int i = 0; i < nums.length; i++) {
            int num = nums[i];

            if (map.containsKey(target - num)) {
                return new int[]{map.get(target - num), i};
            }

            map.put(num, i);
        }

        return null;
    }
}
```