<https://leetcode.com/problems/two-sum/>

Given an array of integers nums and an integer target, return *indices of the two numbers such that they add up to target*.

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

You can return the answer in any order.

**Example 1:**

Input: nums = [2,7,11,15], target = 9

Output: [0,1]

Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

**Example 2:**

Input: nums = [3,2,4], target = 6

Output: [1,2]

**Example 3:**

Input: nums = [3,3], target = 6

Output: [0,1]

**Constraints:**

* 2 <= nums.length <= 104
* -109 <= nums[i] <= 109
* -109 <= target <= 109
* **Only one valid answer exists.**

**Follow-up:**Can you come up with an algorithm that is less than O(n2) time complexity?

**Attempt 1: 2023-02-23**

**Solution 1: Sort then two pointers (30 min)**

class Solution {

public int[] twoSum(int[] nums, int target) {

int len = nums.length;

int[][] arr = new int[nums.length][2];

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

arr[i][0] = nums[i];

arr[i][1] = i;

}

Arrays.sort(arr, (a, b) -> a[0] - b[0]);

int lo = 0;

int hi = len - 1;

while(lo < hi) {

if(arr[lo][0] + arr[hi][0] == target) {

return new int[] {arr[lo][1], arr[hi][1]};

}

if(arr[lo][0] + arr[hi][0] < target) {

lo++;

} else {

hi--;

}

}

return new int[] {};

}

}

Time Complexity: O(nlogn)

Space Complexity: O(n)

**Solution 2:  Hash Table (30 min)**

class Solution {

public int[] twoSum(int[] nums, int target) {

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

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

int a = nums[i];

int b = target - nums[i];

if(map.containsKey(b)) {

return new int[] {map.get(b), i};

}

map.put(a, i);

}

return new int[]{};

}

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to**

<https://leetcode.com/problems/two-sum/solutions/1378064/c-java-python-hashmap-two-pointers-solutions-clean-concise/>

**✔️ Solution 1: HashMap**

We need to find 2 numbers a, b so that a + b = target.

We need a HashMap data structure to store elements in the past, let name it seen.

The idea is that we iterate b as each element in nums, we check if we found a (where a = target - b) in the past.

If a exists in seen then we already found 2 numbers a and b, so that a + b = target, just output their indices.

Else add b to the seen.

class Solution {

public int[] twoSum(int[] nums, int target) {

HashMap<Integer, Integer> seen = new HashMap<>();

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

int b = nums[i], a = target - b;

if (seen.containsKey(a)) return new int[]{seen.get(a), i}; // Found pair of (a, b), so that a + b = target

seen.put(b, i);

}

return new int[]{};

}

}

**Complexity**

* Time: O(N), where N <= 10^4 is number of elements in the array nums.
* Space: O(N)

**Solution 2: Sort then Two Pointers**

* Since this problem require to output **pair of indices** instead of **pair of values**, so we need an array, let say arr to store their value with their respective indices.
* Sort array arr in increasing order by their values.
* Then use two pointer, left point to first element, right point to last element.

class Solution {

public int[] twoSum(int[] nums, int target) {

int n = nums.length;

int[][] arr = new int[n][2];

for (int i = 0; i < n; ++i) {

arr[i][0] = nums[i];

arr[i][1] = i;

}

Arrays.sort(arr, Comparator.comparingInt(o -> o[0])); // Sort arr in increasing order by their values.

int left = 0, right = n - 1;

while (left < right) {

int sum2 = arr[left][0] + arr[right][0];

if (sum2 == target)

return new int[]{arr[left][1], arr[right][1]};

if (sum2 > target)

right -= 1; // Try to decrease sum2

else

left += 1; // Try to increase sum2

}

return new int[]{};

}

}

**Complexity**

* Time: O(N \* logN), where N <= 10^4 is number of elements in the array nums.
* Space: O(N)