**LeetCode Problems for Coding Interviews**

**1. Arrays & Hashing**

**Q.1. Two Sum (LeetCode Q.1)**

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

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

Output: [0,1]

Brute-force: Her eleman için diğer tüm elemanlarla toplamlarını kontrol ederiz. Yani tüm sayı çiftleri denenir. Bu yöntem küçük diziler için çalışır ancak büyük veri setlerinde performans sorunu yaşanır.

Time Complexity: **O(n2)**

Space Complexity: **O(1)**

**Optimal:** Bir “HashMap” yani bir anahtar-değer tablosu kullanarak sayıları işleriz. Her sayının hedefe ulaşmak için ihtiyaç duyduğu tamamlayıcı sayıyı daha önce görüp görmediğimizi kontrol ederiz. Eğer görmüşsek çözümü bulmuş oluruz.

Time Complexity: **O(n)**

Space Complexity: **O(n)**

Optimal Çözüm Java Kodu:

public int[] twoSum(int[] nums, int target) {  
 Map<Integer, Integer> map = new HashMap<>();  
  
 for (int i = 0; i < nums.length; i++) {  
 int complement = target - nums[i];  
  
 if (map.containsKey(complement)) {  
 return new int[] { map.get(complement), i };  
 }  
 map.put(nums[i], i);  
 }  
  
 return new int[] {};  
}

**Q.2. Contains Duplicate (LeetCode Q.217)**

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

Input: nums = [1,2,3,1]

Output: true

Brute-force: Her elemanı diğer tüm elemanlarla karşılaştırarak tekrar eden bir değer olup olmadığını kontrol ederiz.

Time Complexity: **O(n2)**

Space Complexity: **O(1)**

**Optimal:** Daha önce gördüğümüz tüm elemanları bir “HashSet” içinde tutarız. Eğer yeni bir eleman zaten sette varsa, tekrarlı demektir ve “true” döneriz.

Time Complexity: **O(n)**

Space Complexity: **O(n)**

Optimal Çözüm Java Kodu:

static boolean containsDuplicate(int[] nums) {  
 Set<Integer> set = new HashSet<>();  
  
 for (int num : nums) {  
 if (set.contains(num)) {  
 return true;  
 }  
 set.add(num);  
 }

return false;  
}

**Q.3. Valid Anagram (LeetCode Q.242)**

Description: An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, using **all the original letters exactly once**. Given two strings “s” and “t”, return “true” if “t” is an anagram of “s”.

Input: s = "anagram", t = "nagaram"

Output: true

Brute-force: Her iki string’i karakterlerine ayırıp sıralar ve sonra karşılaştırırız. Sıralama işlemi sayesinde anagram olup olmadıklarını anlayabiliriz.

Time Complexity: **O(n log(n))**

Space Complexity: **O(n)**

**Optimal:** Her karakterin kaç kez geçtiğini sayarız. “s” içindeki harfleri “+1”, “t” içindeki harfleri “-1” şeklinde işleriz. Sonuçta tüm sayımlar “0” olmalı.

Time Complexity: **O(n)**

Space Complexity: **O(1)** // Sadece 26 harf için

Optimal Çözüm Java Kodu:

static boolean isAnagram(String s, String t) {  
 if (s.length() != t.length()) {  
 return false;  
 }  
  
 int[] count = new int[26];  
  
 for (int i = 0; i < s.length(); i++) {  
 count[s.charAt(i) - 'a']++;  
 count[t.charAt(i) - 'a']--;  
 }  
  
 for (int c : count) {  
 if (c != 0) {  
 return false;  
 }  
 }  
  
 return true;  
}