# Is Subsequence - Leetcode 392

## My solution:

var isSubsequence = function(s, t) {

if(s === t || s=== '') return true

let start = 0

let end = t.length-1

let idx = 0

let last = s.length-1

let res = 0

while (start <= end){

if(s[idx] === t[start]){

idx++

start++

res++

if(idx === last){

break

}

if(res === s.length){

return true

}

}

if(s[last] === t[end]){

last--

res++

end--

if(res === s.length){

return true

}

}

else {

end--

start++

}

}

return false

};

## Notes:

* Two pointers: significa que vamos a tener dos punteros (i, j) en los dos strings (s y t).
* Me compliqué porque entendí que two pointers consistía en señalar el inicio y final de UN SOLO string.
* La idea mejorada sería iterar t y sumar j siempre, mientras que i solo se suma si hay coincidencia.
* Texto

  Descripción generada automáticamente

# Algorithm I - Binary Search

# 704. Binary Search

* Dos formas: iterative y recursive
* Algoritmo tipo Divide y Vencerás
* Requisito: complejidad O(log n)

## My Solution:

Recursivo: Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Iterativo:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

## Notes:

* La idea es que el vector esté ordenado. Después comprobar el target con el valor del medio e ir descartando mitades del vector, dependiendo si el target es menor o mayor.
* Al principio no me di cuenta de que necesitaba two pointers, el index inferior y superior, significando que consideramos el vector completo. Además, si no usaba esos punteros después no podía recuperar el index correcto (ya que el tamaño variaba).
* En el iterativo notar que el while tiene que ser ‘<=’ porque si se encontraba el target los dos punteros necesariamente tienen que ser iguales.
* Análisis de complejidad: en cada iteración se descarta la mitad del vector, es decir, n -> n/2 -> n/4… Puesto de otra forma, la función que nos dice cuántas veces podemos dividir n entre 2 es ‘***Log2 n = x’***.
* Si el vector tiene muchísimos elementos, en algunos lenguajes puede pasar un overflow cuando se calcula el valor medio. Para arreglarlo calcular el **medio = lower + Math.ceil((upper-lower)/2)**. La división es la distancia entre el inferior y el superior. Imagen de la pantalla de un celular con letras

  Descripción generada automáticamente con confianza media

# 35. Search Insert Position

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

## My solution:

Texto

Descripción generada automáticamente

## Notes:

* La solución funciona pero hay otra forma de pensarlo. Retornar ‘lower’ en lugar de ‘idx’. Ya que al final lower=upper=medio.
* Si target es menor a nums[medio], upper decrementa y pasa a ser menor a lower entonces el valor lógico sería que el número no encontrado este en la posición lower=medio e impone un límite inferior.
* Si target es mayor a nums[medio], lower incrementa y sale del ciclo con un valor mayor a upper que resulta ser la posición donde debería estar el número no encontrado.
* El truco está en entender que si es target es menor, debe estar en la posición del medio (reemplazando), mientras que si es mayor simplemente se suma una posición a medio (no reemplaza).

# Neetcode roadmap problems to get started

# 217. Contains Duplicate

Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.

***Array, Easy***

## My solution:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

## Notes:

* Problemas fáciles para empezar con Python y Leetcode en general. Aquí simplemente lo resolví de la forma más directa y fácil que se me ocurrió aunque sea O(n logn) time complexity y O(1) space. Extra tiempo porque se ordena primero.
* Mejor solución usa un hashset para preguntar si existe el numero o no y logra un time complexity de O(n) pero se sacrifica un poco de space porque hay que crear el hashset y se obtiene un O(n) space compelxity.
  + Texto

    Descripción generada automáticamente
  + **Hashsets son útiles porque se pueden buscar e insertar elementos en tiempo constante.**

# 242. Valid Anagram

Given two strings s and t, return true if t is an anagram of s, and false otherwise.

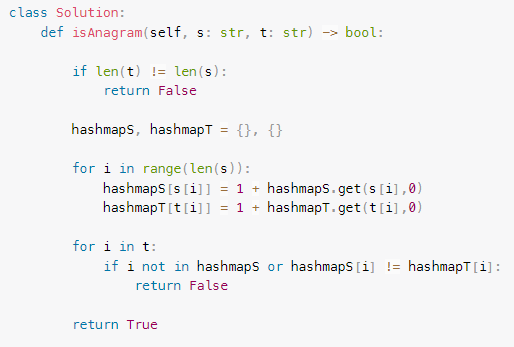
***Hashing, Easy***

## My solution:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

## Notes:

* Problemas fáciles para empezar con Python y Leetcode en general. Aquí lo resolví usando hashmaps contando los caracteres. O(s+t) time complexity, space complexity es lo mismo porque creamos hashmaps de tamaño de s y t. Necesita extra memory.
* Mejor solución es mas concisa pero tiene mejores tiempos y usa mejor el espacio porque yo cree variables innecesarias y use comparaciones de más.
  + 
* Python tiene una estructura de datos que simplifica todo esto llamada **Counter()** que es un hashmap que cuenta todo automáticamente -> **return Counter(s) == Counter(t)**. Tiene la misma complejidad.

## ***Follow up question:*** ¿Puedes hacer una solución que no necesite memoria extra, es decir O(1)?

Se puede evitar crear hashmaps ordenando los strings y simplemente comparándolos. Lo malo es que el time complexity es distinto dependiendo del algoritmo (los buenos tienen O(n logn) time) pero es menester tener en cuenta que incluso los buenos sorting algoritmos usan extra memoria O(n) pero a veces pueden que no si están bien optimizados.

# 1. 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.

***Array, Hashing, Easy***

## My solution:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* Como siempre lo resolví de la forma más directa, fácil y rápida que se me ocurrió, y obviamente no es muy eficiente ya que tiene un time complexity de O(n^2) y space complexity de O(n)

## Better solution:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* Primero podemos notar que usamos Hashmap, la cual empieza vacía y se va llenando a medida que iteramos sobre el array. Además las keys son los valores del array y los values son los índices del array.
* Se usa una técnica muy inteligente que consiste en solo hacer una pasada en el array, en lugar de llenar el hashmap desde un vamos.
* La lógica es que primero se agarra un número, luego se hace la diferencia con el target y el resultado es lo que luego se busca en el hashmap. Si no se encuentra en el hashmap, se actualiza agregándolo al mismo.
* El hecho de que se va llenando de a poco el hashmap y no se llena desde un principio es que, al llegar al segundo número (que estará más delante del primero) estará garantizado que el número restante ya estará en el hashmap.
* Time and space complexity O(n), ya que agregar y buscar en el hashmap es constante.

# 49. Group Anagrams

Given an array of strings strs, group the anagrams together. You can return the answer in any order.

***Array, Hashing, Medium, Sorting, String***

## My solution:

Texto

Descripción generada automáticamente

* Primero recorrí el array y ordené cada string y luego los fui agregando al hashmap.
* Al tener que ordenar se tiene un time complexity de O(m \* n logn), donde n es la longitud media de los strings y es lo que lleva ordenarlos. A esto se lo multiplica por m (cantidad de strings en array).

## Better solution:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* La idea es parecida a lo que se me ocurrió primero, que es contar cuantas veces se repiten los caracteres. Pero, en este caso va mas allá, ya que se tiene un array contador de longitud igual a la cantidad de letras del alfabeto.
* Una cosa muy copada es que a este array contador se lo convierte en tupla y se lo usa como key del hashmap.
* Para acceder al índice correcto se hace la diferencia del ASCII de la letra actual menos ‘a’.
* Time complexity O(m\*n), donde m es la cantidad de strings y n es la longitud promedio de los strings. También se multiplica por 26 (longitud del array contador) pero es despreciable.

# 347. Top K Frequent Elements

Given an integer array nums and an integer k, return the k most frequent elements. You may return the answer in any order.

***Array, Hashing, Divide and Conquer, Sorting, Heap (Priority Queue), Bucket Sort, Counting, Quickselect***

## My solution:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* Se me ocurrió contar las veces que se repiten los números y guardarlos en el hashmap donde key=num y val=veces, luego ordernar los valores y preguntar si están en el hashmap para guardar la key en un array.
* Toma demasiado tiempo. O(n logn) por el sorting.
* Se obtenía mas o menos lo misms O(k logn) is se usa max heaps así se evita el sorting.

## Better solution:

Texto

Descripción generada automáticamente

* Se usa la misma idea que yo pensé, que es almacenar la frecuencia de ocurrencia en un hashmap.
* Utilizamos **Bucket Sort**: la idea básica es cuando tenemos un input array con números uniformemente distribuidos, creamos un array donde cada elemento son buckets. Tomamos cada elemento arr[n] del input array y vamos contando la frecuencia y guardamos el contador en el índice=arr[n], es decir bucket[arr[n]]. Luego ser ordena cada bucket con otro algoritmo.

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza media

Sería **O(n) time complexity** si los valores del input array fuesen bounded (que sepamos que va de 1-10 por ejemplo). Pero si hay un solo valor muy grande (1millon), entonces nuestro bucket array será de longitud muy grande.

* Entonces, se hace lo mismo pero se invierten el **índice** con el **count**. Cada índice representará el conteo y en los valores vamos a tener un array que indica los números en sí que se repiten ‘índice’ veces. De esta forma podemos también determinar los ‘k’ valores que mas se repiten ya que solo tenemos que iterar de atrás para adelante.

Un reloj digital

Descripción generada automáticamente con confianza media

Vamos a tener un bucket array de longitud igual al tamaño del input array, es decir que es bounded a len(input array) + 1.

* **Time and space complexity O(n),** ya que en el peor de los casos vamos a tener que iterar hasta el índice 1 (todos los números del input array se repetirían 1 sola vez). Y como dijimos, el tamaño del array es igual tamaño del input array.

# 238. Product of Array Except Self

Given an integer array nums, return an array answer such that answer[i] is equal to the product of all the elements of nums except nums[i].

You must write an algorithm that runs in O(n) time and without using the division operation.

***Array, Prefix Sum, Medium***

## My solution: **No lo pude resolver**

## Solution:

Como lo dice en los tags, se podía usar “Prefix Sum” (que desconocía) y básicamente consiste en ir multiplicando (en este caso porque se puede sumar, etc) los prefijos de cada número, donde los prefijos son los números anteriores. Después se multiplican los sufijos que son los números que vienen después. Finalmente se multiplican prefijos por sufijos y dan el resultado.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**Es O(n) time and space complexity**.

Pero una **mejor forma** para ahorrar espacio es hacer dos pasadas, primero de izquierda a derecha y después al revés e ir almacenando primero los prefijos y después se los multiplica por los sufijos en el mismo output:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

## Código:

Texto

Descripción generada automáticamente

* Necesitamos inicializar variables postfix y prefix con un valor de 1, ya que el prefix del primer elemento es 1 (idem último elemento). En la primera pasada se almacena en la posición correspondiente a cada elemento **SU prefix** y luego se actualiza el prefix para la siguiente iteración. Yo pensé en hacer al revés, donde se actualiza el prefix con el valor anterior y recién se almacena.
* La primera pasada quedaría [-1,1,0,-3,3] -> [1,-1,-1,0,0]. Cada posición corresponde con el prefix del elemento en dicha posición.
* La segunda pasada simplemente se multiplica el postfix por lo que está almacenado en la posición actual y se actualiza el postfix.
* **O(n) time y O(1) space** porque el problema dice que el output array no cuenta como memoria extra.

# 36. Valid Sudoku

Determine if a 9 x 9 Sudoku board is valid. Only the filled cells need to be validated according to the following rules:

* Each row must contain the digits 1-9 without repetition.
* Each column must contain the digits 1-9 without repetition.
* Each of the nine 3 x 3 sub-boxes of the grid must contain the digits 1-9 without repetition.

Note:

A Sudoku board (partially filled) could be valid but is not necessarily solvable.

Only the filled cells need to be validated according to the mentioned rules.

***Array, Matrix, Medium, Hashing***

## My solution:

No lo pude resolver pero mi idea era para las filas y columnas iterar y comparar, lo cual me pareció que no era buena idea porque parecía costosa.

* Lo que no se me ocurrió es usar hashset en lugar de arrays. *Recordemos que para buscar e insertar en un hashset, se hace en tiempo constante.*

Para el tema de los sub cuadros se me ocurrió guardarlos en un hashmap pero no pude descifrar qué keys usar para identificar cada matriz 3x3.

* En la solución se explica cómo hacer esto…

## Solution:

Para las filas y columnas tenemos variables que guardar un hashmap de hashset donde las keys son las posiciones de cada elemento en la fila/columna y los valores son los números.

Para los sub cuadros, la solución es super ingeniosa. Resulta que suponemos a cada cuadro le corresponde un índice 0, 1, 2 y para que cada par [fila][columna] entre dentro de ya sea 0, 1 o 2, se divide entre 3 -> Ej: [7][4] = [7/3][4/3] = [2][1]. Estos pares van a ser las keys del hashmap en forma de tupla (r/3, c/3)

Imagen que contiene Diagrama

Descripción generada automáticamente

Texto

Descripción generada automáticamente

* **O(n=9^2) time and space complexity**