# Leetcode Notes and Practice

Sebastian Pucher

May 25, 2025

## Contents

**1 Hashing** **2**
- 1.1 Valid Anagram . . . 2
- 1.2 Twosum . . . 2

**2 Two Pointers** **2**
- 2.1 Valid Palindrome . . . 2

**3 Sliding Window** **3**
- 3.1 Best Time to Buy and Sell Stocks . . . 3

**4 Stack** **3**
- 4.1 Valid Parentheses . . . 3

**5 Binary Search** **4**
- 5.1 Binary Search Recursive . . . 4
- 5.2 Binary Search Iterative . . . 4
- 5.3 Search 2D Matrix . . . 5

# 1 Hashing

## 1.1 Valid Anagram

**Question:** Given two strings s and t, return true if the two strings are anagrams of each other, otherwise return false

- Create two separate dictionaries
- Loop through one of the input strings, add key letter or letter freq.
- If dict are the same, return true

## 1.2 Twosum

**Question:** Given an array of integers nums and an integer target, return the indices i and j such that nums[i] + nums[j] == target and i != j

- Use a hashmap to store the index of each number in the array as the *value*
- On each iteration, check first to see if the difference between the target val and the current num is already stored in the hashmap
- If it is, then return the value at that key (the index), as well as the current index i
- If it's not, then add the current number and index to the hashmap
- **Key Idea:** Always check the existence between the target and the current number as a key in the hashmap first!

# 2 Two Pointers

## 2.1 Valid Palindrome

**Question:** Given a string s, return true if it is a palindrome, otherwise return false.

A palindrome is a string that reads the same forward and backward. It is also case-insensitive and ignores all non-alphanumeric characters.

- First change the string to all lowercase with .lower() function, this ensures case sensitive args are taken care of
- Init left and right pointers to first and last characters
- Do a check using **.isalnum()**. If it is *not* alpha numeric, then increment or decrement the pointer and *continue* through the loop
- While left is less than or equal two right, compare the letters, if not the same, return false
- increment / decrement left and right at bottom of loop

# 3 Sliding Window

## 3.1 Best Time to Buy and Sell Stocks

**Question:** You are given an integer array prices where prices[i] is the price of NeetCoin on the ith day.

You may choose a single day to buy one NeetCoin and choose a different day in the future to sell it.

Return the maximum profit you can achieve. You may choose to not make any transactions, in which case the profit would be 0.

- **Buy low... sell high**: To achieve this, init two pointers, left wanting to find the low value in the arr, and right wanting to find the high value. Init left to first index, and right to second index in array
- Begin looping through, so long as the right pointer is less than the length of the array
- If the right value is ever less than the left value, then you have found a new low to sell at. *So assign left equal to right, and increment right by one: continue the while loop*
- Upon each iteration check if the difference between right and left values is greater than global profit (which is init to 0). If so, update profit

# 4 Stack

## 4.1 Valid Parentheses

**Question:**You are given a string s consisting of the following characters: '(', ')', '{', '}', '[' and ']'.

The input string s is valid if and only if: – Every open bracket is closed by the same type of close bracket. – Open brackets are closed in the correct order. – Every close bracket has a corresponding open bracket of the same type.

Return true if s is a valid string, and false otherwise.

- **Main idea:** Use a Stack to push open brackets, and pop closed brackets
- Before iterating through each char of the string, create an array of valid open brackets, and an array of valid close brackets (same positioning in each)
- For each char, if the char is neither in each bracket array, return false.
- If the char is an open bracket, push to stack
- If the char is an close bracket, *first check if the stack is non empty* then check to see if the last value on the stack is the corresponding open bracket to the current close bracket
- Continue looping if it is, return false if it isn't
- After the loop completes, ensure that the stack is empty, if it is, return true

# 5 Binary Search

## 5.1 Binary Search Recursive

**Question:** You are given an array of distinct integers nums, sorted in ascending order, and an integer target.

Implement a function to search for target within nums. If it exists, then return its index, otherwise, return -1.

Your solution must run in O(logn) time.

- **Main idea:** To solve this problem recursively, you must keep track of left and right indices upon each recursive call
- Initialize left to be index zero, and right to be last index (len(A) -1)
- *Base Case:* If left index equals right index, check if the value is in only position, if it is, return index, else return -1
- If left doesn't equal right, find the mid index, and compare target value to value at middle index
- If it equals the middle index, return
- If it's greater than the middle index, recurse, left = middle index + 1
- If it's less than the middle index, recurse, right = middle index - 1
- **If parameters don't use left and right, you'll have to add them initially as none as defaults, then set them before recursing**

## 5.2 Binary Search Iterative

**Question:** You are given an array of distinct integers nums, sorted in ascending order, and an integer target.

Implement a function to search for target within nums. If it exists, then return its index, otherwise, return -1.

Your solution must run in O(logn) time.

- **Main idea:** To solve this problem iteratively, you must keep track of left and right indices and use a while loop
- *While loop Condition:* Loop so long as right is greater than or equal to the left index
- *Base Case:* Within the while loop, calculate the mid index:
- If left doesn't equal right, find the mid index, and compare target value to value at middle index
- If it equals the middle index, return
- If it's greater than the middle index, recurse, left = middle index + 1

- If it's less than the middle index, recurse, right = middle index - 1
- **If parameters don't use left and right, you'll have to add them initially as none as defaults, then set them before recursing**

## 5.3 Search 2D Matrix

**Question:**

- **Main idea:**