# Leetcode meeting link

安排：每周一到周五，美西6:30-7:30

<https://us02web.zoom.us/j/85884383545?pwd=dEY1azlXbnlMSXMyU1IvQk8vMDA0dz09>

Meeting ID: 858 8438 3545

Passcode: wmast

# TwoSum

Q1

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.

Similar problems167, 704, 33, 339

## Solution 1 (hash map, fastest solution):

* Use the **original list to minus the target value** and conduct the search if the complement of each value is in the new subtracted list.
* It only takes one for-loop if use list.
* We can also use hashable table (python implementation of hashable table is dictionary), Java create map “Map<Integer, Integer> myMap=new HashMap<>( )”. When conduct search, use “myMap.containsKey(Complement)”.

## Solution 2(two pointer):

* For the original array, need to sort it first.
* Then create a head pointer and tail pointer, index pointing to the first and last value in the sorted array.
* The summation of the two values pointed by these two pointer
  + If bigger than target, decrease the tail pointer
  + If smaller than target, increase the head pointer
  + Until the summation is the same as the target, then the two pointers (indexes) are the solution.

# Two pointer

Q1 Two sum

Q344 rev.erse\_string

Q238 product of the array except itself at each element.

# Binary search

## Q704

Given a **sorted** (in ascending order) integer array nums of n elements and a target value, write a function to search target in nums. If target exists, then return its index, otherwise return -1.

Solution1 brute force search

* One For loop search the item in the array, very slow

Solution 2(Binary search)

* Set and start\_index, and end\_index for the first and last array. The mid\_index is then the round((start\_index-end\_index)/2).
* While loop (end\_index-start\_index)>2:
  + If any of the start\_index, mid\_index and end\_index element is the target, then that is the solution.
  + If mid\_index element > target, then update the new end\_index with mid\_index, and the new mid\_index is calculated with the equation again round((start\_index-end\_index)/2).
  + If mid\_index element < target, then update the new start\_index with mid\_index, and the new mid\_index is calculated with the equation again round((start\_index-end\_index)/2).
* If the while loop is finished, that means (end\_index-start\_index)<=2, so simply, the answer right now lays in either start\_index element or end\_index element.

## Q247 find the hindex of a given citations, that is an array.

# Binary Search Tree (BST)

Q653

How to traverse a BST:

* In-order algorithm (left-root-right)
* Recursive function, inOrder; when node==null, return, means stop search.
* Otherwise, traverse left node first ; and then print out the node.val; then traverse the right node;
* Psuedo code:
  + If (node==null) {return;}
  + inOrder(node.left);
  + Printout(node.val);
  + inOrder(node.right);

Pre-order (root-left-right )

Post-order(left-right-root)

# Subarray

## Continous array has the largest sum

Q53

Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

Solution 1: starting from the first index, set a curSum, whenever curSum is smaller than 0, reset surSum=0, because negative value will not contribute to larger sum results. Thus, only need one for loop.

## Find the continuous subarray that is sum equal to a given value

Q560

Given an array of integers nums and an integer k, return the total number of continuous subarrays whose sum equals to k.

* Create a hashmap, to save the cumulative sum of the array as “key” of the map, and the number of times the value appears as the “value” of the map.
* Every time when have a new cumulative sum for this array, check if the complement value (between the new cuSum and the give value K) exist in the map. If exist, get this complement value’s times of appearance.
* By the end of the loop in the original array, the total number of all the times of appearance is the final result, the total number of continuous array that is summed equal to the given value K.

## Max product of continuous subarray

Q152, Max product subarray

Given an integer array nums, find the contiguous subarray within an array (containing at least one number) which has the largest product.

Create a current\_max, current\_min, final\_max parameter.

* Create a current\_max parameter to track the maximum of current continuous product from first to current element in an array, as each for loop iteration is continuing going, compare (current\_max\*current\_element, current\_minimum\*current\_element, current\_element) to get the new current\_max.
* Create a minimum parameter to track the minimum of the current continuous product from first to current element in array, as each for loop iteration is continuing going, (current\_max\*current\_element, current\_minimum\*current\_element, current\_element) to get the new current\_minimum.
* If current\_max > final\_max, update final\_max.
* At the end of one for loop, the final\_max is the continuous max product we are looking for.

# Duplicate element in an array

## Q277, 217 ,220

Given an array of integers, find if the array contains any duplicates.

Solution 1: use a hashmap to check if current key is contained in the map. Values in the array saved as key in the map, one for loop.

Solution 2: (maybe easier to realize in python) if the length of the array is not equal the length of the set, then there must be duplicates.

## Q27 , 26, 80 Remove element or duplicate element

Given an array nums and a value val, remove all instances of that value in-place and return the new length. (similar Q27, Q26, 80)

**Solution 1: Tracking the useful element and keep piling them at the beginning of the array**

* Initialize a pointer at the beginning index, to track the value needs to be preserve,and iterate through all the elements
* If the element is the one to preserve, than put it at the beginning, and move the pointer forward once. Swap the early element into the current iteration position.
* By the end of iteration, how many times the pointer has increment to, that is the number of elements of the new array. The returned array with preserved elements at the beginning up to the index of the pointer, after that, it junk elements.

**Solution 2: replace the element that is need to be delete with the last value in the array, shrink the size of the array every time when there is an element found to be deleted.**

* Initialize a counter to track the preserved elements. Count=0
* With a while(i< array\_length), if current element is the one we want to delete, then replace this element with the last element in the array, and array\_length-1 becomes the new array\_length in the while loop.
* If the current\_element is the value we want to preserve, increment the counter by 1, count++.
* By the end of the loop, counter indicates the length of preserved array, and the useful elements is at the beginning of the array, up to the index of the counter.

**Q80 is allowing element to repeat twice, so that the third repeated element will get removed.**

* In this question, beside initiate a pointer to track the location of element,
* we also need to create a counter to track if the repeated element is more than 2 or not.

# Rotate array

Q189, Given an array, rotate the array to the right by k steps, where k is non-negative.

* First, if k is x number of times of the length of the array, then we just need to rotate the array to the steps where it is the **residual of k divided by the length of the array.**
* **Starting with index 0, the first element, we will jump forward k steps. When the current index+k is bigger than the length of the array, we return back to (index+k)-length location**, where this element should be rotate into.
* **When we come back to meet the starting index again, we have finished a cycle for this rotation**. However, we might not cover all the element in this array, as some of the rotation could come back to zero and skip some of the element in the array.
* Thus, we **will use the starting index plus one, and start the rotation process again**.
* At the same time, we will **have a counter** to count all the elements we have rotated. When the **counter has reached to the total number of length, then we finished rotating** for all the elements.

# H-index of an array

## Q247

given an array, calculate its h-index

* Solution 1:
  + First sort the array, then iterate from high value to low value
  + Each loop compare the current index and the value, if the value is higher than current index, then h-index plus 1. (If current value is higher than the current index, that means all the previous values are higher than current index, because it is sorted. )
* Solution 2 :
  + The idea is track all the possible hindex, for a given citation, we add one credit to all the possible hindex in an array or list.

I.e.  create a variable hindex\_list, the length is the length of the the given citations parameters.

When we have a given citation, we add one credit for all the locations in the hindex\_list. For example, if the citation we are given is 4, then we add one credit for location 0, 1, 2, 3 (be aware index stating from 0).

* By the end of iteration, we check the scores in the hindex\_list, if the score is higher than its location then we get all the locations that match this criteria. The highest locations that matches the condition the score is higher that its location, that is the hindex we are looking for.
* Solution 3, binary search
  + First sort this array, from low to high, the index is from 0 to n -1, can the possible\_hindex is from n to 1
  + The goal is to find an index at where, the citation\_value < possible\_hindex at the left of that index, and citation\_value > possible\_hindex at the right side of the hindex
  + Treat special case, when citations is with length 0 and 1.
  + Use binary search, have L and R stands for the left and right end, M is middle.
    - When the citations has length >=2. first treat special case, if the index at 0 and n is non-zero citation, then the hindex=1.
    - When citations[M] == possible\_hindex, job is done, this possible\_hindex is the hindex, because the array is sorted, Moving left the possible\_hindex is increasing, and the array is sorted from low to high from left to right. So, left side of M has to be smaller citation value than possible\_hindex, and right side of M has to be bigger citation value than possible\_hindex.
    - When citations[M] > possible\_hindex, update the hindex-possible\_hindex, make the new\_right=M, because the hindex might be higher than current hindex.
    - When citations[M] < possible\_hindex, make the new\_left=M, because the possible\_hindex is higher than the current citation value.
    - Stopping criteria is when L=M or R=M.

# Hash Map to track elements

## Q41 Tracking the first missing positive number.

Q383 Q387, using for loops to track individual character appearance in a string.

Q205, isomorphic strings. Put one char in a string as key, and the char of the corresponding index in another string as the value.

# Find sub string from larger string

## Q28 strStr

For a given substring, iterate through the larger string, and each time of iteration, compare the substring to the same length of strings in the larger string.