Day 45 / 100 :

Topic - Array, Binary search

1 Problem statement: Search in Rotated array (medium)

There is an integer array nums sorted in ascending order (with distinct values).
Prior to being passed to your function, nums is possibly rotated at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (0-indexed). For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become [4,5,6,7,0,1,2].
Given the array nums after the possible rotation and an integer target, return the index of target if it is in nums, or -1 if it is not in nums.
You must write an algorithm with O(log n) runtime complexity.

Example 1:
Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4

Example 2:
Input: nums = [4,5,6,7,0,1,2], target = 3
Output: -1

Example 3:
Input: nums = [1], target = 0
Output: -1

Solutions :

Approach 1 - Binary search

- The function search takes in a vector nums representing the rotated sorted array and an integer target that we need to find.
- It initializes the variables left and right to the start and end indices of the array, respectively.

- The variable mid is calculated as the middle index of the array using the formula (left + right) / 2.
- The while loop is executed as long as the left pointer is less than or equal to the right pointer. This ensures that the search space is not empty.
- Inside the while loop, we check if the middle element nums[mid] is equal to the target. If it is, we return the index mid.
- If the middle element is greater than or equal to the left element, it means the left part of the array is non-rotated.
- We then check if the target is within the range of the left non-rotated part. If it is, we update the right pointer to mid - 1 to search in the left part.
- If the target is not within the range, we update the left pointer to mid + 1 to search in the right part.
- If the middle element is less than the left element, it means the right part of the array is rotated.
- We then check if the target is within the range of the right rotated part. If it is, we update the left pointer to mid + 1 to search in the right part.
- If the target is not within the range, we update the right pointer to mid - 1 to search in the left part.

Complexity:

- Time complexity:
  O(log n)
- Space complexity:
  O(1)

```
Class Solution {
public:
    int search(vector<int>& nums, int target) {
     int n = nums.size();
      int left = 0;
      int right = n-1;
      int mid= left + (right - left) / 2;
      while(left <= right){
        if(nums[mid] == target)
         return mid;
        if(nums[mid] >= nums[left]) {
          if(target >= nums[left] && target <= nums[mid])
            {
                right = mid - 1;
```

```
            }
           else left = mid + 1;
        }
        else {
          if(target >= nums[mid] && target <= nums[right])
            left = mid + 1;
          else right = mid - 1;
        }
          mid = left + (right - left) / 2;
      }
      return -1;
    }
};
```

2 Problem statement: [Roman to Integer](#) (medium)

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.
Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

- I can be placed before V (5) and X (10) to make 4 and 9.
- X can be placed before L (50) and C (100) to make 40 and 90.
- C can be placed before D (500) and M (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer.

Example 1:
Input: s = "III"

Output: 3
Explanation: III = 3.

Example 2:
Input: s = "LVIII"
Output: 58
Explanation: L = 50, V= 5, III = 3.

Example 3:
Input: s = "MCMXCIV"
Output: 1994
Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.

Solutions :

Approach 1 - Hash table

If there are Numbers such as XL, IV, etc.
Simply subtract the smaller number and add the larger number in next step.
For example, if there is XL, ans =-10 in first step, ans=-10+50=40 in next step.
Otherwise, just add the numbers.

Complexity:

- Time complexity: O(n)

```
class Solution {
public:
    int romanToInt(string s) {
        int ans=0;
        unordered_map <char,int> mp{
        {'I',1},{'V',5},{'X',10},{'L',50},{'C',100},{'D',500},{'M',1000}};

    for(int i=0;i<s.size();i++){
        if(mp[s[i]]<mp[s[i+1]]){
            //for cases such as IV,CM, XL, etc...
            ans=ans-mp[s[i]];
        }
        else{
            ans=ans+mp[s[i]];
        }
    }
        return ans;
    }
};
```