Day 10 / 100 :

Topic - Array, string

1 Problem statement: Single Number II (Medium)

Given an integer array nums where every element appears three times except for one, which appears exactly once. Find the single element and return it.
You must implement a solution with a linear runtime complexity and use only constant extra space.

Example 1:
Input: nums = [2,2,3,2]
Output: 3
Example 2:
Input: nums = [0,1,0,1,0,1,99]
Output: 99

Solutions :

Approach 1 - brute force

```
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int c=1;
        for(int i=0;i<nums.size();i++){
            c=1;
            for(int j=0;j<nums.size();j++){
                if(nums[i]==nums[j] && i!=j){
                    c++;
                    break;
                }
            }
            if(c==1)
                return nums[i];
        }
        return 0;
    }
};
```

1. Initialize the variable c to 1. This variable is used to count the number of occurrences of each element.
2. Start a loop to iterate over each element in the nums array. The loop variable i represents the current index.
3. Within the outer loop, initialize c to 1 for each new element. This is necessary because we need to count the occurrences of the current element.
4. Start an inner loop to compare the current element at index i with all other elements in the array. The loop variable j represents the index of the other elements.
5. Check if the element at index i is equal to the element at index j (excluding the same index). If they are equal, it means we have found a duplicate occurrence of the current element. In that case, increment c and break out of the inner loop.
6. After the inner loop completes, check if c is still equal to 1. If it is, it means we have found the single element, as it has no duplicates in the array. In that case, return the single element (nums[i]).
7. If the outer loop completes without finding the single element, return 0 as a default value. This line is reached only if all elements have duplicates and the single element is not present.

Solutions :

Approach 2 - Optimized Solution

1. Initialize ans as 0, which will store the single element that appears once.
2. Iterate over each bit position from 0 to 31.
3. Create a bitmask by left-shifting 1 by i bits.
4. Initialize a counter c to 0.
5. Iterate over each element j in nums and increment c if the ith bit of j is set.
6. If c is not divisible by 3, set the ith bit of ans using a bitwise OR operation with the bitmask.
7. Repeat the above steps for all bit positions.
8. Return ans as the final result.

This solution has a linear runtime complexity of O(N) and uses constant extra space.

```
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int ans=0;
        for(int i=0;i<32;i++)
```

```
        {
            int bit=1<<i;
            int c=0;
            for(int j:nums)
            {
                if(j&bit)
                c++;
            }
            if(c%3)
            ans|=bit;
        }
        return ans;
    }
};
```

2 Problem statement: Search in Rotated sorted 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 - Brute Force

The variable ans is initialized to -1. This variable will store the index of the found target value or -1 if the target is not present in the vector.

1. The for loop iterates over each element of the nums vector using the index variable i.
2. Inside the loop, the code checks if the element at the current index i is equal to the target value using the condition if (nums[i] == target).
3. If the condition is true, it means the target value has been found. The index i is assigned to the ans variable.
4. The loop continues until either the target value is found or all elements in the vector have been checked.
5. After the loop finishes, the function returns the value of ans. If the target value was found, it contains the index of the first occurrence of the target; otherwise, it remains -1.

```
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int ans=-1;
        for(int i=0; i<nums.size(); i++){
            if(nums[i] == target){
                ans =i;
            }
        }
        return ans;
    }
};
```

Solutions :

Approach 2 - Optimized Solution

Explanation:

1. Find middle point mid = (l + h)/2
2. If key is present at middle point, return mid.

3. Else If arr[l..mid] is sorted
   a) If key to be searched lies in range from arr[l]
   to arr[mid], recur for arr[l..mid].
   b) Else recur for arr[mid+1..h]
4. Else (arr[mid+1..h] must be sorted)
   a) If key to be searched lies in range from arr[mid+1]
   to arr[h], recur for arr[mid+1..h].
   b) Else recur for arr[l..mid]

```
class Solution {
public:
    int search(vector<int>& nums, int target) {
      int beg=0,end=nums.size()-1,mid;
        while(beg<=end)
        {
            mid=(beg+end)/2;
            if(nums[mid]==target)
                return mid;
            if(nums[beg]<=nums[mid])
            {
                if(target<=nums[mid] && target>=nums[beg])
                    end=mid-1;
                else
                    beg=mid+1;
            }

            else
            {
                if(target>=nums[mid] && target<=nums[end])
                   beg=mid+1;
                else
                    end=mid-1;
            }

        }
        return -1;
    }
};
```