1. Shuffle a string

<https://assets.leetcode.com/uploads/2020/07/09/q1.jpg>

**Input:** s = "codeleet", indices = [4,5,6,7,0,2,1,3]

**Output:** "leetcode"

**Explanation:** As shown, "codeleet" becomes "leetcode" after shuffling.

class Solution {

    public String restoreString(String s, int[] indices) {

        char[] ans = new char[indices.length];

        for (int i = 0; i < indices.length; i++) {

            ans[indices[i]] = s.charAt(i);

        }

        return new String(ans);

    }

}

1. Score of a string
2. **Example 1:**
3. **Input:** s = "hello"
4. **Output:** 13
5. **Explanation:**
6. The **ASCII** values of the characters in s are: 'h' = 104, 'e' = 101, 'l' = 108, 'o' = 111. So, the score of s would be |104 - 101| + |101 - 108| + |108 - 108| + |108 - 111| = 3 + 7 + 0 + 3 = 13.

class Solution {

    public int scoreOfString(String s) {

        int sum = 0;

        for(int i=0; i<s.length()-1; i++)

sum += Math.abs(s.charAt(i)-s.charAt(i+1));

        return sum;

    }

}

3. Number of 1 bits

**Input:** n = 11

**Output:** 3

**Explanation:**

The input binary string **1011** has a total of three set bits.

class Solution {

    public int hammingWeight(int n) {

        return Integer.bitCount(n);

    }

}

4. Reverse a bit in binary(Reverse bits of a given 32 bits unsigned integer.)

**Input:** n = 11111111111111111111111111111101

**Output:** 3221225471 (10111111111111111111111111111111)

**Explanation:** The input binary string **11111111111111111111111111111101** represents the unsigned integer 4294967293, so return 3221225471 which its binary representation is **10111111111111111111111111111111**.

public class Solution {

    // you need treat n as an unsigned value

    public int reverseBits(int n) {

        int result = 0;

        for (int i = 0; i < 32; i++) {

            result <<= 1;  // Shift result left by 1 bit

            result |= (n & 1);  // Add the least significant bit of n to result

            n >>= 1;  // Shift n right by 1 bit

        }

        return result;

    }

}

5. Single number

**Input:** nums = [2,2,1]

**Output:** 1

class Solution {

    public int singleNumber(int[] nums) {

        int unique=0;

        for(int i=0;i<nums.length;i++){

            unique^=nums[i];

        }

        return unique;

    }

}

 **Iteration 1:** unique ^= 4 (unique becomes 4).

 **Iteration 2:** unique ^= 1 (unique becomes 4 ^ 1 = 5).(4 in binary is 100 + 001=101=5)

 **Iteration 3:** unique ^= 2 (unique becomes 5 ^ 2 = 7).

 **Iteration 4:** unique ^= 1 (unique becomes 7 ^ 1 = 6).

 **Iteration 5:** unique ^= 2 (unique becomes 6 ^ 2 = 4).

6. First missing positive (HARD)

**Input:** nums = [1,2,0]

**Output:** 3

**Explanation:** The numbers in the range [1,2] are all in the array.

class Solution {

    public int firstMissingPositive(int[] nums) {

        for(int i =0; i<nums.length;i++){

            if(nums[i] <= 0 || nums[i]> nums.length)

                    continue;

            while(nums[i] != i+1 && nums[i] != nums[nums[i]-1]){

                int temp = nums[nums[i]-1];

                nums[nums[i]-1] = nums[i];

                nums[i] = temp;

                if(nums[i] <= 0 || nums[i]> nums.length)

                    break;

            }

        }

        for(int i =0 ; i<nums.length;i++){ for the test case nums=[1,2,0]

            if(nums[i] != i+1){

                return i+1;

            }

        }

        return nums.length+1; for the test case nums=[1]

    }

}

7. Check if array is good (HARD)

**Input:** nums = [12,5,7,23]

**Output:** true

**Explanation:** Pick numbers 5 and 7.

5\*3 + 7\*(-2) = 1

class Solution {

    public boolean isGoodArray(int[] nums) {

        int temp = nums[0];

        for(int i=0;i<nums.length;i++){

            temp = func(temp, nums[i]);

            if(temp == 1){

                return true;

            }

        }

        return false;

    }

    int func(int x, int y){

        if(y == 0){

            return x;

        }

        else{

            return func(y, (x%y));

        }

    }

}

1. Find in mountain array

**Input:** array = [1,2,3,4,5,3,1], target = 3

**Output:** 2

**Explanation:** 3 exists in the array, at index=2 and index=5. Return the minimum index, which is 2.

class Solution {

    public int findInMountainArray(int target, MountainArray mountainArr) {

        int peak=findPeak(mountainArr);

        int firstTry=orderBs(mountainArr,target,0,peak);

        if(firstTry != -1){

            return firstTry;

        }

        return orderBs(mountainArr,target,peak+1,mountainArr.length()-1);

    }

    public static int findPeak(MountainArray mountainArr){

        int start=0;

        int end=mountainArr.length()-1;

        while(start<end){

            int mid=start+(end-start)/2;

            if(mountainArr.get(mid)>mountainArr.get(mid+1)){

                end=mid;

            }

            else{

                start=mid+1;

            }

        }

        return start;

    }

    public static int orderBs(MountainArray mountainArr,int target,int start,int end){

        boolean isAsc=mountainArr.get(start)<mountainArr.get(end);

        while(start<=end){

            int mid=start+(end-start)/2;

            if(mountainArr.get(mid)==target){

                return mid;

            }

            if(isAsc){

                if(mountainArr.get(mid)>target)

                {

                    end=mid-1;

                }

                else{

                    start=mid+1;

                }

            }

            else{

                if(mountainArr.get(mid)<target)

                {

                    end=mid-1;

                }

                else{

                    start=mid+1;

                }

            }

        }

        return -1;

    }

}