78. Subsets

Given an integer array `nums`, return *all possible subsets (the power set)*.

The solution set must not contain duplicate subsets.

Example 1:

```
Input: nums = [1,2,3]
Output: [[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]
```

Example 2:

```
Input: nums = [0]
Output: [[],[0]]
```

Constraints:

- `1 <= nums.length <= 10`
- `-10 <= nums[i] <= 10`

```
//Bitwise solutiion
class Solution {
    public List<List<Integer>> subsets(int[] nums) {
        List<List<Integer>> ans =new ArrayList();
        int len=nums.length;
        int powlen=(int)Math.pow(2,len);
        List<Integer> temp;
        for(int i=0;i<powlen;i++)
        {
            temp=new ArrayList<Integer>();
            for(int j=0;j<len;j++)
            {
                if(((i>>j)&1)==1)
                    temp.add(nums[j]);
            }
            ans.add(temp);
        }
        return ans;
    }
}
```

136. Single Number

Given a **non-empty** array of integers `nums`, every element appears *twice* except for one. Find that single one.

Follow up: Could you implement a solution with a linear runtime complexity and without using extra memory?

Example 1:

```
Input: nums = [2,2,1]
Output: 1
```

Example 2:

```
Input: nums = [4,1,2,1,2]
Output: 4
```

Example 3:

```
Input: nums = [1]
Output: 1
```

Constraints:

- `1 <= nums.length <= 3 * 10^4`
- `-3 * 10^4 <= nums[i] <= 3 * 10^4`
- Each element in the array appears twice except for one element which appears only once.

---

https://leetcode.com/problems/single-number/ (https://leetcode.com/problems/single-number/)

```
/*
Brute force will be two loops , iterating for each element

Another solution is using hashmap or hashset but using extra space

Basically we need to know following two properties of XOR
1.  value ^ value=0
2.  value ^ 0 =value
3.  XOR follow Associative law (a1^a2^a3^a4)=(a2^a1^a4^a1)
Using above properties we can eliminate all elemnts occuring twice
Remaining will be answer
*/
class Solution {
    public int singleNumber(int[] nums) {
        int ans=nums[0];
        for(int i=1;i<nums.length;i++)
        {
            ans^=nums[i];
        }
        return ans;
    }
}
```

---

191. Number of 1 Bits

Write a function that takes an unsigned integer and returns the number of '1' bits it has (also known as the Hamming weight (http://en.wikipedia.org/wiki/Hamming_weight)).

Note:

- Note that in some languages such as Java, there is no unsigned integer type. In this case, the input will be given as a signed integer type. It should not affect your implementation, as the integer's internal binary representation is the same, whether it is signed or unsigned.
- In Java, the compiler represents the signed integers using 2's complement notation (https://en.wikipedia.org/wiki/Two%27s_complement). Therefore, in **Example 3** above, the input represents the signed integer. `-3`.

Follow up: If this function is called many times, how would you optimize it?

Example 1:

```
Input: n = 00000000000000000000000000001011
Output: 3
Explanation: The input binary string 00000000000000000000000000001011 has a total of th
```

Example 2:

```
Input: n = 00000000000000000000000010000000
Output: 1
Explanation: The input binary string 00000000000000000000000010000000 has a total of on
```

Example 3:

```
Input: n = 11111111111111111111111111111101
Output: 31
Explanation: The input binary string 11111111111111111111111111111101 has a total of th
```

Constraints:

- The input must be a **binary string** of length `32`

---

https://leetcode.com/problems/number-of-1-bits/ (https://leetcode.com/problems/number-of-1-bits/)

/* Explanation (Total 3 methoda)

1. Simple just check whether each bit is set or not using loop (num>>1)&1
2. Below method by checking only set bits using n&(n-1) it will unset all bits from LSB
3. Using look up table (Important for many queries or numbers in one go) Refer gfg notes

*/

```
public class Solution {
    // you need to treat n as an unsigned value
    public int hammingWeight(int n) {
        int res=0;
        while(n!=0)
        {
            n=n & (n-1);
            res++;
        }
        return res;
    }
}
```

---

231. Power of Two

Given an integer `n`, return *`true` if it is a power of two. Otherwise, return `false`*.

An integer `n` is a power of two, if there exists an integer `x` such that `n == 2^x`.

Example 1:

```
Input: n = 1
Output: true
Explanation: 2^0 = 1
```

Example 2:

```
Input: n = 16
Output: true
Explanation: 2^4 = 16
```

Example 3:

```
Input: n = 3
Output: false
```

Example 4:

```
Input: n = 4
Output: true
```

Example 5:

```
Input: n = 5
Output: false
```

Constraints:

- $-2^{31} <= n <= 2^{31} - 1$

---

https://leetcode.com/problems/power-of-two/ (https://leetcode.com/problems/power-of-two/) /* Explanation: Here n&(n-1) will unset rightmost bit of n to 0 for power of 2 we will required only 1 set bit in n so after performing n&(n-1) we should get 0 as answer if n is power of 2 as only 1 bit will n and that will unset because above operation */

```
class Solution {
    public boolean isPowerOfTwo(int n) {
        if(n<=0)
            return false;
        return ((n&(n-1))==0);
    }
}
```

```
class Solution {
    public boolean isPowerOfTwo(int n) {
        if(n==0 || n==Integer.MIN_VALUE)
            return false;
        return ((n&(n-1))==0);
    }
}
```

```
return n > 0 && Integer.bitCount(n) == 1;
```

```
//Recursive
return n > 0 && (n == 1 || (n%2 == 0 && isPowerOfTwo(n/2)));
```

---

260. Single Number III

Given an integer array `nums`, in which exactly two elements appear only once and all the other elements appear exactly twice. Find the two elements that appear only once. You can return the answer in **any order**.

Follow up: Your algorithm should run in linear runtime complexity. Could you implement it using only constant space complexity?

Example 1:

```
Input: nums = [1,2,1,3,2,5]
Output: [3,5]
Explanation:  [5, 3] is also a valid answer.
```

Example 2:

```
Input: nums = [-1,0]
Output: [-1,0]
```

Example 3:

```
Input: nums = [0,1]
Output: [1,0]
```

Constraints:

- `1 <= nums.length <= 30000`
- Each integer in `nums` will appear twice, only two integers will appear once.

---

https://leetcode.com/problems/single-number-iii (https://leetcode.com/problems/single-number-iii)

Refer gfg notes or single number I for more details

Once again, we need to use XOR to solve this problem. But this time, we need to do it in two passes:

In the first pass, we XOR all elements in the array, and get the XOR of the two numbers we need to find. Note that since the two numbers are distinct, so there must be a set bit (that is, the bit with value '1') in the XOR result. Find out an arbitrary set bit (for example, the rightmost set bit).

In the second pass, we divide all numbers into two groups, one with the aforementioned bit set, another with the aforementinoed bit unset. Two different numbers we need to find must fall into thte two distrinct groups. XOR numbers in each group, we can find a number in either group.

```
class Solution {
    public int[] singleNumber(int[] nums) {
        int ans[]=new int[2];
        int n=nums.length;
        int xor=0;
        for(int i=0;i<n;i++)
        {
            xor=xor^nums[i];
        }
        int rightMostSetBit= (xor & (~(xor-1)));
        for(int i=0;i<n;i++)
        {
            int curr=nums[i];
            if((curr&rightMostSetBit)==0)
                ans[0]=ans[0]^curr;
            else
                ans[1]=ans[1]^curr;
        }
        return ans;
    }
}
```

---

268. Missing Number

Given an array `nums` containing `n` distinct numbers in the range `[0, n]`, return *the only number in the range that is missing from the array.*

Follow up: Could you implement a solution using only `O(1)` extra space complexity and `O(n)` runtime complexity?

Example 1:

```
Input: nums = [3,0,1]
Output: 2
Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2
```

Example 2:

```
Input: nums = [0,1]
Output: 2
Explanation: n = 2 since there are 2 numbers, so all numbers are in the range [0,2]. 2
```

Example 3:

```
Input: nums = [9,6,4,2,3,5,7,0,1]
Output: 8
Explanation: n = 9 since there are 9 numbers, so all numbers are in the range [0,9]. 8
```

Example 4:

```
Input: nums = [0]
Output: 1
Explanation: n = 1 since there is 1 number, so all numbers are in the range [0,1]. 1 is
```

Constraints:

- `n == nums.length`
- `1 <= n <= 10^4`
- `0 <= nums[i] <= n`
- All the numbers of `nums` are **unique**.

---

https://leetcode.com/problems/missing-number/ (https://leetcode.com/problems/missing-number/) // Use following property (1^2^3^4^5...upto n) ^(nums[i] every) =0 if no number is missing refer notes of gfg

```
class Solution {
    public int missingNumber(int[] nums) {
        int sum=0;
        int n=nums.length;
        for(int i=0;i<n;i++)
        {
            sum=sum+nums[i];
        }
        return (n*(n+1)/2)-sum;
    }
}
```

```
class Solution {
    public int missingNumber(int[] nums) {
        int ans=0;
        int n=nums.length;
        int i;
        for(i=0;i<n;i++)
        {
            ans=ans^nums[i]^i;
        }
        return ans^i;
    }
}
```

326. Power of Three

Given an integer `n`, return *`true` if it is a power of three. Otherwise, return `false`*.

An integer `n` is a power of three, if there exists an integer `x` such that `n == 3^x`.

Example 1:

```
Input: n = 27
Output: true
```

Example 2:

```
Input: n = 0
Output: false
```

Example 3:

```
Input: n = 9
Output: true
```

Example 4:

```
Input: n = 45
Output: false
```

Constraints:

- $-2^{31} <= n <= 2^{31} - 1$

Follow up: Could you do it without using any loop / recursion?

---

https://leetcode.com/problems/power-of-three/ (https://leetcode.com/problems/power-of-three/)

```
//Brute Force
class Solution {
    public boolean isPowerOfThree(int n) {
        if(n<=0)
            return false;
        while(n!=1)
        {
            if(n%3!=0)
                return false;
            n=n/3;
        }
        return true;
    }
}
```

```
/*
Explanation:
convert number into base 3 form
as we know to power of 3 only 1 should be appera once in base 3 representation
so we should use match function or regex
We will use the regular expression above for checking if the string starts with 1 ^1,
is followed by zero or more 0s 0* and contains nothing else $.
*/
class Solution {
    public boolean isPowerOfThree(int n) {
        return Integer.toString(n, 3).matches("^10*$");
    }
}
```

```
/*
Explabnation:
here 1162261467 is maximum integer which is power of 3 i.e. 3^19
so if above number is divisible by n then n is also power of 3
3^(logbase3 to Integermax)=3^19.56=3^19
*/
public class Solution {
    public boolean isPowerOfThree(int n) {
        return n > 0 && 1162261467 % n == 0;
    }
}
```

---

338. Counting Bits

Given a non negative integer number **num**. For every numbers **i** in the range **0 ≤ i ≤ num** calculate the number of 1's in their binary representation and return them as an array.

Example 1:

```
Input: 2
Output: [0,1,1]
```

Example 2:

```
Input: 5
Output: [0,1,1,2,1,2]
```

Follow up:

- It is very easy to come up with a solution with run time **O(n*sizeof(integer))**. But can you do it in linear time **O(n)** /possibly in a single pass?
- Space complexity should be **O(n)**.
- Can you do it like a boss? Do it without using any builtin function like **__builtin_popcount** in c++ or in any other language.

---

https://leetcode.com/problems/counting-bits/ (https://leetcode.com/problems/counting-bits/) This technique is look up table (dp) refer gfg notes

```
class Solution {
    public int[] countBits(int num) {
        int ans[]=new int[num+1];
        for(int i=1;i<=num;i++)
        {
            ans[i]=ans[i>>1]+(i&1);//use bracket to follow bodmass rule
                        // alternative : ans[i]=ans[i/2] + i&1
                        // i/2 will check for all bits excludeing LSB (last)
                        // i&1 check for even or odd ( 1 in LSB)
        }
        return ans;
    }
}
```

---

342. Power of Four

Given an integer `n`, return *`true` if it is a power of four. Otherwise, return `false`*.

An integer `n` is a power of four, if there exists an integer `x` such that `n == 4^x`.

Example 1:

```
Input: n = 16
Output: true
```

Example 2:

```
Input: n = 5
Output: false
```

Example 3:

```
Input: n = 1
Output: true
```

Constraints:

- $-2^{31} <= n <= 2^{31} - 1$

Follow up: Could you solve it without loops/recursion?

---

https://leetcode.com/problems/power-of-four/ (https://leetcode.com/problems/power-of-four/)

```
//Brute Force
class Solution {
    public boolean isPowerOfFour(int n) {
        if(n<=0)
            return false;
        while(n!=1)
        {
            if(n%4!=0)
                return false;
            n=n/4;
        }
        return true;
    }
}
```

```
class Solution {
    public boolean isPowerOfFour(int n) {
        int check=Integer.parseInt("10101010101010101010101010101",2);
        return ((n>0) && (n&(n-1))==0 && (n|(check))==check);
    }
}
```

```
public boolean isPowerOfFour(int num) {
        return Integer.bitCount(num) == 1 && (Integer.toBinaryString(num).length()-
1)%2==0;
    }
```

---

389. Find the Difference

You are given two strings `s` and `t`.

String `t` is generated by random shuffling string `s` and then add one more letter at a random position.

Return the letter that was added to `t`.

Example 1:

```
Input: s = "abcd", t = "abcde"
Output: "e"
Explanation: 'e' is the letter that was added.
```

Example 2:

```
Input: s = "", t = "y"
Output: "y"
```

Example 3:

```
Input: s = "a", t = "aa"
Output: "a"
```

Example 4:

```
Input: s = "ae", t = "aea"
Output: "a"
```

Constraints:

- `0 <= s.length <= 1000`
- `t.length == s.length + 1`
- `s` and `t` consist of lower-case English letters.

---

https://leetcode.com/problems/find-the-difference (https://leetcode.com/problems/find-the-difference)

```
/*
Explanation:
Total 5 methods:
1.  Brute force (compare every element of s with t ) O(n^2)
2.  Sorting ( sort both s and t string. and check every index of s matching with t) O
(nlogn +n)
3.  Using frequency Array (store occurence of character) O(n)
4.  Using sum of all char at t-sum of all char of s (ASCII Value) O(n)
5.  Using XOR of every element of s and t (All same character will 0 due to XOR and r
emaining will be char which is not present in t)  O(n)
*/
```

```java
//Solution 5
class Solution {
    public char findTheDifference(String s, String t) {
        if(s.length()==0)
            return t.charAt(0);
        int ans=s.charAt(0)^t.charAt(0);
        int i;
        for(i=1;i<s.length();i++)
        {
            ans=ans^(s.charAt(i)^t.charAt(i));
        }
        ans^=t.charAt(i);
        return (char)ans;
    }
}
```

```java
//Solution 4
class Solution {
    public char findTheDifference(String s, String t) {
        int ans=0;
        int i;
        for(i=0;i<s.length();i++)
        {
            ans+=t.charAt(i)-s.charAt(i);
        }
        ans+=t.charAt(i);
        return (char)ans;
    }
}
```

```
//Solution 3
class Solution {
    public char findTheDifference(String s, String t) {
        int freq[]=new int[26];
        int i;
        for(i=0;i<s.length();i++)
        {
            freq[s.charAt(i)-'a']--;
            freq[t.charAt(i)-'a']++;
        }
        freq[t.charAt(i)-'a']++;
        for(int j=0;j<26;j++)
        {
            if(freq[j]==1)
                return (char)('a'+j);
        }
        return ' ';
    }
}
```

```
//Solution 2
class Solution {
    public char findTheDifference(String s, String t) {
        char temp[]=s.toCharArray();
        Arrays.sort(temp);
        s=new String(temp);
        temp=t.toCharArray();
        Arrays.sort(temp);
        t=new String(temp);
        int i;
        for(i=0;i<s.length();i++)
        {
            if(s.charAt(i)!=t.charAt(i))
                return t.charAt(i);
        }
        return t.charAt(i);
    }
}
```

---

461. Hamming Distance

The Hamming distance (https://en.wikipedia.org/wiki/Hamming_distance) between two integers is the number of positions at which the corresponding bits are different.

Given two integers `x` and `y`, calculate the Hamming distance.

Note:
$0 \leq x, y < 2^{31}$.

Example:

```
Input: x = 1, y = 4

Output: 2

Explanation:
1   (0 0 0 1)
4   (0 1 0 0)
       ↑   ↑

The above arrows point to positions where the corresponding bits are different.
```

---

https://leetcode.com/problems/hamming-distance/submissions/ (https://leetcode.com/problems/hamming-distance/submissions/)

```
/*
Explanation:
consider :  0^0=0; 1^1=0
i.e. only we will get set bit in XOR if either bit in numbers is 1 and other is 0
i.e if we XOR given two numbers let say 'ans' then set bit in ans will be our answer
as we required different bits in two numbers
Then calculate set bit in 'ans' using brain keriningam's algorithm to find set bit
*/
```

```
class Solution {
    public int hammingDistance(int x, int y) {
        int num=x^y;
        int ans=0;
        while(num!=0)
        {
            ans++;
            num=num & (num-1);
        }
        return ans;
    }
}
```

---

476. Number Complement

Given a **positive** integer `num`, output its complement number. The complement strategy is to flip the bits of its binary representation.

Example 1:

```
Input: num = 5
Output: 2
Explanation: The binary representation of 5 is 101 (no leading zero bits), and its comp
```

Example 2:

```
Input: num = 1
Output: 0
Explanation: The binary representation of 1 is 1 (no leading zero bits), and its comple
```

Constraints:

- The given integer `num` is guaranteed to fit within the range of a 32-bit signed integer.
- `num >= 1`
- You could assume no leading zero bit in the integer's binary representation.
- This question is the same as 1009: https://leetcode.com/problems/complement-of-base-10-integer/ (https://leetcode.com/problems/complement-of-base-10-integer/)

---

https://leetcode.com/problems/number-complement/ (https://leetcode.com/problems/number-complement/)

```
//brute force
/*
Explanation:
Here calculating all 1 till num
suppose num=5 (101)  then calculate number as (111)
Then Subtracting 111-101=010 as answer;
*/
class Solution {
    public int findComplement(int num) {
        int number=0;
        while(num>number)
        {
            number=(number<<1) | 1;
        //Alternative: number+=Math.pow(2,i);
        }
        return number-num;
    }
}
```

```
//optimal
/*
here (-num-1)=~num
Generally if we have to calculate complement we follow above method
Here logic is same as brute force
Instead of while loop for calculating numbers with all one
we are using Highest set bit of given number then right shift that bit
After -1 from above result we get number which have all 1
*/
class Solution {
    public int findComplement(int num) {
        if(num==0) //Explicitely handel case for 0
            return 1;
        return (-num-1) & ((Integer.highestOneBit(num)<<1)-1);
    }
}
```

---

477. Total Hamming Distance

The Hamming distance (https://en.wikipedia.org/wiki/Hamming_distance) between two integers is the number of positions at which the corresponding bits are different.

Now your job is to find the total Hamming distance between all pairs of the given numbers.

Example:

```
Input: 4, 14, 2

Output: 6

Explanation: In binary representation, the 4 is 0100, 14 is 1110, and 2 is 0010 (just
showing the four bits relevant in this case). So the answer will be:
HammingDistance(4, 14) + HammingDistance(4, 2) + HammingDistance(14, 2) = 2 + 2 + 2 = 6
```

Note:

1. Elements of the given array are in the range of `0` to `10^9`
2. Length of the array will not exceed `10^4` .

---

https://leetcode.com/problems/total-hamming-distance/ (https://leetcode.com/problems/total-hamming-distance/)

```
//Brute Force (Look up table)
class Solution {
    public int totalHammingDistance(int[] nums) {
        int bit_array[]=new int[256];
        for(int index=1;index<256;index++)
        {
            bit_array[index]=bit_array[index>>1] + (index & 1);
        }
        int len=nums.length;
        int answer=0;
        for(int index_outer=0;index_outer<len;index_outer++)
        {
            for(int index_inner=index_outer+1;index_inner<len;index_inner++)
            {
                int xor_value=nums[index_outer]^nums[index_inner];
                answer=answer+bit_array[xor_value & 0xFF];
                // xor_value=xor_value>>8;
                answer=answer+bit_array[xor_value>>8 & 0xFF];
                // xor_value=xor_value>>8;
                answer=answer+bit_array[xor_value>>16 & 0xFF];
                // xor_value=xor_value>>8;
                answer=answer+bit_array[xor_value>>24 & 0xFF];
            }
        }
        return answer;
    }
}
```

```
Explanation :
Instead of checking every pair we will consider all number bitwise
Maximum bits in every integer is 32 so
we will check set bit in every position in every integer
basically we are calculating haming distance bitwise for all numbers not numberwise
suppose 20 numbers are there, so we will consider all 32 bits in 20 numbers individua
lly
count set bits (1) for all number in LSB first
then haming distance for that particular bit for all numbers will (set bit)*(unset bi
t)
i.e. (set bit)(n-set bit)
we will add above to answer
repeat this process for all 32 bits
```

```
//optimal
class Solution {
    public int totalHammingDistance(int[] nums) {
        int answer=0;
        int len=nums.length;
        for(int index=0;index<32;index++)
        {
            int bit_count=0;
            for(int num_index=0;num_index<len;num_index++)
            {
                bit_count=bit_count+((nums[num_index]>>index) & 1);
            }
            answer=answer+(bit_count*(len-bit_count));
        }
        return answer;
    }
}
```

645. Set Mismatch

The set `S` originally contains numbers from 1 to `n`. But unfortunately, due to the data error, one of the numbers in the set got duplicated to **another** number in the set, which results in repetition of one number and loss of another number.

Given an array `nums` representing the data status of this set after the error. Your task is to firstly find the number occurs twice and then find the number that is missing. Return them in the form of an array.

Example 1:

```
Input: nums = [1,2,2,4]
Output: [2,3]
```

Note:

1. The given array size will in the range [2, 10000].
2. The given array's numbers won't have any order.

---

https://leetcode.com/problems/set-mismatch/ (https://leetcode.com/problems/set-mismatch/)

```
/*
Explanation:
Other than following methods:
1.Brute Force (comapare each index and check duplicate) O(n^2)
2.Sorting ( Sort array and check which consecutive index are same) O(nlogn)
3.Using HashMap O(n) / Or frequency array
4.Using XOR ( Same as sum where we want to find 2 odd times repeating number)
   (By XORING all array and natural number, checking set bit from right) O(n)
*/
```

```
/*
Explanation:
Using Following Equations:
x^2 -y^2= (x+y)(x-y)
Consider x as repeatiting number and y as missing number
we know , sum of natural number till n let say val=(n*(n+1)/2)
and sum of square of natural number till n let say sval=(n*(n+1)*(2*n+1)/6)
consider sum as sum of all elements in array
Here sum will be val(sum of natural) +x (Extra repeating number) -y(missing number)
i.e. sum=val+(x-y)
and ssum as square of all elements in array
similarly ssum=sval + x^2 -y^2;
so, ssum=sval+(x+y)(x-y);
     ssum=sval +(x+y)(sum-val)
         in above we already find ssum,sval,sum,val
now we can calculate x+y from above equations
no we have x+y and x-y so we can calculate x and y i.e our answer
converting above logic in program:
```

```
class Solution {
    public int[] findErrorNums(int[] nums) {
        long sum=0,ssum=0;
        long n=nums.length;
        for(int i=0;i<(int)n;i++)
        {
            sum+=nums[i];
            ssum+=(nums[i]*nums[i]);
        }
        long val=sum-(n*(n+1)/2);
        long sval=ssum-(n*(n+1)*(2*n+1)/6);
        long sadd=sval/val;
        int ans[]=new int[2];
        ans[0]=(int)((val+sadd)/2);
        ans[1]=(int)(sadd-ans[0]);
        return ans;
    }
}
```

check else if condition if you not understand

```
class Solution {
    public int[] findErrorNums(int[] nums) {
        int ans[]=new int[2];
        for(int i=0;i<nums.length;i++)
        {
            int temp=Math.abs(nums[i]);
            ans[1]=ans[1]^(i+1)^(temp);
            if(nums[temp-1]<0)
                ans[0]=temp;
            else if(nums[temp-1]>0)
                nums[temp-1]=-nums[temp-1];
        }
        ans[1]=ans[1]^ans[0];
        return ans;
    }
}
```

---

693. Binary Number with Alternating Bits

Given a positive integer, check whether it has alternating bits: namely, if two adjacent bits will always have different values.

Example 1:

```
Input: n = 5
Output: true
Explanation: The binary representation of 5 is: 101
```

Example 2:

```
Input: n = 7
Output: false
Explanation: The binary representation of 7 is: 111.
```

Example 3:

```
Input: n = 11
Output: false
Explanation: The binary representation of 11 is: 1011.
```

Example 4:

```
Input: n = 10
Output: true
Explanation: The binary representation of 10 is: 1010.
```

Example 5:

```
Input: n = 3
Output: false
```

Constraints:

- $1 <= n <= 2^{31} - 1$

---

https://leetcode.com/problems/binary-number-with-alternating-bits/ (https://leetcode.com/problems/binary-number-with-alternating-bits/)

```
/*
Explanation:
start from 1 ansd 2
we want to check either number is within  ....01010101 or ....10101010
so we will right shift two bits and OR with 1 or 2 so
in iteration
check1                   check2
00000001              00000010
00000101              00001010
00010101              00101010
01010101              10101010

for every iteration we will check whether given number is equal to check1 or check2
*/
```

```
class Solution {
    public boolean hasAlternatingBits(int n) {
        if(n==1)
            return true;
        if(n==2)
            return true;
        int check1=1;
        int check2=2;
        for(int i=0;i<15;i++)
        {
            check1=check1<<2|1;
            check2=check2<<2|2;
            if(n==check1 ||  n==check2)
                return true;
        }
        return false;
    }
}
```

```
//Second Solution
/*
n & n>>1 will ensure that all bits are alternative
*/
class Solution {
    public boolean hasAlternatingBits(int n) {
        return ((n & (n>>1))==0 && (n & (n>>2))==(n>>2));
    }
}
```

762. Prime Number of Set Bits in Binary Representation

Given two integers `L` and `R`, find the count of numbers in the range `[L, R]` (inclusive) having a prime number of set bits in their binary representation.

(Recall that the number of set bits an integer has is the number of `1`s present when written in binary. For example, `21` written in binary is `10101` which has 3 set bits. Also, 1 is not a prime.)

Example 1:

```
Input: L = 6, R = 10
Output: 4
Explanation:
6 -> 110 (2 set bits, 2 is prime)
7 -> 111 (3 set bits, 3 is prime)
9 -> 1001 (2 set bits , 2 is prime)
10->1010 (2 set bits , 2 is prime)
```

Example 2:

```
Input: L = 10, R = 15
Output: 5
Explanation:
10 -> 1010 (2 set bits, 2 is prime)
11 -> 1011 (3 set bits, 3 is prime)
12 -> 1100 (2 set bits, 2 is prime)
13 -> 1101 (3 set bits, 3 is prime)
14 -> 1110 (3 set bits, 3 is prime)
15 -> 1111 (4 set bits, 4 is not prime)
```

Note:

1. `L, R` will be integers `L <= R` in the range `[1, 10^6]`.
2. `R - L` will be at most 10000.

---

https://leetcode.com/problems/prime-number-of-set-bits-in-binary-representation/
(https://leetcode.com/problems/prime-number-of-set-bits-in-binary-representation/)

```
class Solution {
    boolean prime[];
    int arr[];
    Solution()
    {
        arr=new int[256];
        for(int i=1;i<256;i++)
        {
            arr[i]=arr[i>>1] + (i&1);
        }
        prime=new boolean[33];
        prime[0]=true;
        prime[1]=true;
        for(int i=2;i<=5;i++)
        {
            if(prime[i]==false)
            {
                for(int j=i*i;j<=32;j+=i)
                {
                    prime[j]=true;
                }
            }
        }
    }

    int isprime(int n)
    {
        if(prime[n]==false)
            return 1;
        return 0;
    }

    public int countPrimeSetBits(int L, int R) {
        int ans=0;
        for(int i=L;i<=R;i++)
        {
            int res=0;
            res+=arr[i & 0xFF];
            res+=arr[(i>>8) & 0xFF];
            res+=arr[(i>>16) & 0xFF];
            res+=arr[(i>>24) & 0xFF];
            ans+=isprime(res);
        }
        return ans;
    }
}
```

```
/*
0b10100010100010101100 is the bit wise representation of 665772.
Here 2nd,3rd,5th,7th,11th,13th,17th,19th,23rd and 29th bits are 1 and rest are 0s.
What do all these positions have in common? they are prime numbers
Integer.bitCount() is method to calculate set bits of number;
*/

Java stream:
public int countPrimeSetBits(int L, int R) {
    return IntStream.range(L, R+1).map(i -> 665772 >> Integer.bitCount(i) & 1).sum();
}


Java:
public int countPrimeSetBits(int L, int R) {
    int count = 0;
    while (L <= R)
        count += 665772 >> Integer.bitCount(L++) & 1;
    return count;
}
```

---

852. Peak Index in a Mountain Array

Let's call an array `arr` a **mountain** if the following properties hold:

- `arr.length >= 3`
- There exists some `i` with `0 < i < arr.length - 1` such that:
    - `arr[0] < arr[1] < ... arr[i-1] < arr[i]`
    - `arr[i] > arr[i+1] > ... > arr[arr.length - 1]`

Given an integer array arr that is **guaranteed** to be a mountain, return any `i` such that `arr[0] < arr[1] < ... arr[i - 1] < arr[i] > arr[i + 1] > ... > arr[arr.length - 1]`.

Example 1:

```
Input: arr = [0,1,0]
Output: 1
```

Example 2:

```
Input: arr = [0,2,1,0]
Output: 1
```

Example 3:

```
Input: arr = [0,10,5,2]
Output: 1
```

Example 4:

```
Input: arr = [3,4,5,1]
Output: 2
```

Example 5:

```
Input: arr = [24,69,100,99,79,78,67,36,26,19]
Output: 2
```

Constraints:

- `3 <= arr.length <= 10^4`
- `0 <= arr[i] <= 10^6`
- `arr` is **guaranteed** to be a mountain array.

---

876. Middle of the Linked List

Given a non-empty, singly linked list with head node `head`, return a middle node of linked list.

If there are two middle nodes, return the second middle node.

Example 1:

```
Input: [1,2,3,4,5]
Output: Node 3 from this list (Serialization: [3,4,5])
The returned node has value 3.  (The judge's serialization of this node is [3,4,5]).
Note that we returned a ListNode object ans, such that:
ans.val = 3, ans.next.val = 4, ans.next.next.val = 5, and ans.next.next.next = NULL.
```

Example 2:

```
Input: [1,2,3,4,5,6]
Output: Node 4 from this list (Serialization: [4,5,6])
Since the list has two middle nodes with values 3 and 4, we return the second one.
```

Note:

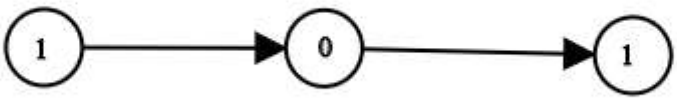
- The number of nodes in the given list will be between `1` and `100`.

1290. Convert Binary Number in a Linked List to Integer

Given `head` which is a reference node to a singly-linked list. The value of each node in the linked list is either 0 or 1. The linked list holds the binary representation of a number.

Return the *decimal value* of the number in the linked list.

Example 1:




```
Input: head = [1,0,1]
Output: 5
Explanation: (101) in base 2 = (5) in base 10
```

Example 2:

```
Input: head = [0]
Output: 0
```

Example 3:

```
Input: head = [1]
Output: 1
```

Example 4:

```
Input: head = [1,0,0,1,0,0,1,1,1,0,0,0,0,0,0]
Output: 18880
```

Example 5:

```
Input: head = [0,0]
Output: 0
```

Constraints:

- The Linked List is not empty.
- Number of nodes will not exceed `30`.
- Each node's value is either `0` or `1`.

---

https://leetcode.com/problems/convert-binary-number-in-a-linked-list-to-integer (https://leetcode.com/problems/convert-binary-number-in-a-linked-list-to-integer)

```
//Optimal
class Solution {
    public int getDecimalValue(ListNode head) {
        int num = head.val;
        while (head.next != null) {
            num = (num << 1) | head.next.val;
                        // Above is alternative for (num *2) +head.next.val;
            head = head.next;
        }
        return num;
    }
}
```

```
//bruteForce
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */
class Solution {
    public int getDecimalValue(ListNode head) {
        int ans=0;
        ArrayList <Integer> a=new ArrayList<Integer>();
        ListNode temp=head;
        while(head!=null)
        {
            a.add(head.val);
            head=head.next;
        }
        for(int i=a.size()-1;i>=0;i--)
        {
            ans=ans+(int)Math.pow(2,a.size()-i-1)*a.get(i);
        }
        return ans;
    }
}
```

//Optimal one loop

```
//Calculating from MSB
class Solution {
    public int getDecimalValue(ListNode head) {
        int ans=0;
        int raise=30; // maximum size of int is 30 given
        while(head!=null)
        {
            ans=ans+(int)Math.pow(2,raise)*head.val;
            raise--;
            head=head.next;
        }
        ans=ans>>(raise+1); (right shift so that we get actual integer)
        return ans;
    }
}
```

1310. XOR Queries of a Subarray

Given the array `arr` of positive integers and the array `queries` where `queries[i] = [Li, Ri]`, for each query `i` compute the **XOR** of elements from `Li` to `Ri` (that is, `arr[Li] xor arr[Li+1] xor ... xor arr[Ri]`). Return an array containing the result for the given `queries`.

Example 1:

```
Input: arr = [1,3,4,8], queries = [[0,1],[1,2],[0,3],[3,3]]
Output: [2,7,14,8]
Explanation:
The binary representation of the elements in the array are:
1 = 0001
3 = 0011
4 = 0100
8 = 1000
The XOR values for queries are:
[0,1] = 1 xor 3 = 2
[1,2] = 3 xor 4 = 7
[0,3] = 1 xor 3 xor 4 xor 8 = 14
[3,3] = 8
```

Example 2:

```
Input: arr = [4,8,2,10], queries = [[2,3],[1,3],[0,0],[0,3]]
Output: [8,0,4,4]
```

Constraints:

- `1 <= arr.length <= 3 * 10^4`
- `1 <= arr[i] <= 10^9`
- `1 <= queries.length <= 3 * 10^4`
- `queries[i].length == 2`
- `0 <= queries[i][0] <= queries[i][1] < arr.length`

---

https://leetcode.com/problems/xor-queries-of-a-subarray (https://leetcode.com/problems/xor-queries-of-a-subarray)

```
/*
Explanation:
we know following properties of XOR:
1. A ^ A =0
2. A ^ 0 =A
3. A^B^C=C^A^B (Associative)
We we use prefix array Tecnique to solve above problem
Here we need some pre computation
We will store Xor of all contiguous subarray starting from index 0 (prefix array)
i.e prefix[2] contain XOR of all elements form arr[0] ...arr[2]
prefix[5] =arr[0] ^arr[1] ^arr[1] ^ arr[2] ^ arr[3] ^arr[4] ^arr[5]
now we have to perform queries considering above array
Let say query is from 2 to 5
now in answer we required arr[2]^arr[3]^arr[4] ^arr[5]           ......eqn(1)
consider following operation:
prefix[5] ^ prefix[2]=(arr[0]^arr[1]^arr[2]^arr[3]^arr[4]^arr[5]) ^ (arr[0]^arr[1]^ar
r[2])
According to mention above XOR property same elements will be 0
prefix[5] ^ prefix[2]=arr[3] ^arr[4] ^ arr[5]                    .....eqn(2)
Now look our needed answer in eqn 1 and what we got in eqn 2
So we need to just XOR leftpoaition of query in above eqn 2
prefix[5] ^prefix[2] ^arr[2]=eqn(1)   = answre
so form above we can conclude that
answer= prefix[R] ^prefix[L] ^arr[L]

*/
```

```
class Solution {
    public int[] xorQueries(int[] arr, int[][] queries) {
        int arrayLength=arr.length;
        int queryLength=queries.length;
        int prefix[]=new int[arrayLength];
        int ans[]=new int[queryLength];
        prefix[0]=arr[0];
        for(int index=1;index<arrayLength;index++)
        {
            prefix[index]=prefix[index-1]^arr[index];
        }

        for(int index=0;index<queryLength;index++)
        {
            int leftPosition=queries[index][0];
            int rightPosition=queries[index][1];
            ans[index]= prefix[leftPosition] ^ prefix[rightPosition] ^ arr[leftPositi
on];
        }
        return ans;
    }
}
```

1318. Minimum Flips to Make a OR b Equal to c

Given 3 positives numbers `a`, `b` and `c`. Return the minimum flips required in some bits of `a` and `b` to make (`a` OR `b` == `c`). (bitwise OR operation).
Flip operation consists of change **any** single bit 1 to 0 or change the bit 0 to 1 in their binary representation.

Example 1:


```
Input: a = 2, b = 6, c = 5
Output: 3
Explanation: After flips a = 1 , b = 4 , c = 5 such that (a OR b == c)
```

Example 2:

```
Input: a = 4, b = 2, c = 7
Output: 1
```

Example 3:

```
Input: a = 1, b = 2, c = 3
Output: 0
```

Constraints:

- 1 <= a <= 10^9
- 1 <= b <= 10^9
- 1 <= c <= 10^9

---

https://leetcode.com/problems/minimum-flips-to-make-a-or-b-equal-to-c/
(https://leetcode.com/problems/minimum-flips-to-make-a-or-b-equal-to-c/)

```
/*
Explanation:
1) find OR of a,b let say orvalue=a|b;
2)Now we Have to compare result of orvalue with given c
3)Iterate for 32 times(32 bit int in java)
4)There are two case if mismatch is found
  1.if in given c bit is 1 and orvalue has bit 0
    for above case we need to change bit of orvalue to 1
    we don't care about which number among a,b to set 1
    but we know that if bit of either one bit is set 1 then orvalue bit will be 1
    basically we have to change bit of one number only (OR logic property)
    then ans will be increment by 1
  2.if given c bit is 0 and orvalue has bit 1
    In above there are two possibilities
      a)If bit of both a and b are set so we have to unset both bits to change orvalu
e to 0
        so we will increase ans by 2
      b)If bit of either bit is set so we have to change only one bit(among a,b) to 0
        as other number bit is already 0
        so we will increase ans by 1.
5)if mismatch not found do nothing.
*/
```

```
class Solution {
    public int minFlips(int a, int b, int c) {
        int ans=0;
        int or=a | b;
        for(int i=0;i<32;i++)
        {
            int inctwo=(c>>i)&1;
            int incone=(or>>i)&1;
            if((incone)!=(inctwo))
            {
                if(inctwo==0)
                {
                    ans+=((a>>i)&1)^0;
                    ans+=((b>>i)&1)^0;
                    //Instaed of above two statements you can use following:
                    //Read explanation point 4.2
                    /*
                    if( ((a>>i)&1) == 1)
                        ans++;
                    if( ((b>>i)&1) == 1)
                        ans++;
                    */
                }
                else
                    ans++;
            }
        }
        return ans;
    }
}
```

1342. Number of Steps to Reduce a Number to Zero

Given a non-negative integer `num`, return the number of steps to reduce it to zero. If the current number is even, you have to divide it by 2, otherwise, you have to subtract 1 from it.

Example 1:

```
Input: num = 14
Output: 6
Explanation:
Step 1) 14 is even; divide by 2 and obtain 7.
Step 2) 7 is odd; subtract 1 and obtain 6.
Step 3) 6 is even; divide by 2 and obtain 3.
Step 4) 3 is odd; subtract 1 and obtain 2.
Step 5) 2 is even; divide by 2 and obtain 1.
Step 6) 1 is odd; subtract 1 and obtain 0.
```

Example 2:

```
Input: num = 8
Output: 4
Explanation:
Step 1) 8 is even; divide by 2 and obtain 4.
Step 2) 4 is even; divide by 2 and obtain 2.
Step 3) 2 is even; divide by 2 and obtain 1.
Step 4) 1 is odd; subtract 1 and obtain 0.
```

Example 3:

```
Input: num = 123
Output: 12
```

Constraints:

- `0 <= num <= 10^6`

---

https://leetcode.com/problems/number-of-steps-to-reduce-a-number-to-zero/ (https://leetcode.com/problems/number-of-steps-to-reduce-a-number-to-zero/)

```
class Solution {
    public int numberOfSteps (int num) {
        int ans=0;
        while(num!=0)
        {
            if((num & 1)==0)
                num=num>>1;
            else
                num=num-1;
            ans++;
        }
        return ans;
    }
}
```

1356. Sort Integers by The Number of 1 Bits

Given an integer array `arr` . You have to sort the integers in the array in ascending order by the number of **1's** in their binary representation and in case of two or more integers have the same number of **1's** you have to sort them in ascending order.

Return *the sorted array*.

Example 1:

```
Input: arr = [0,1,2,3,4,5,6,7,8]
Output: [0,1,2,4,8,3,5,6,7]
Explantion: [0] is the only integer with 0 bits.
[1,2,4,8] all have 1 bit.
[3,5,6] have 2 bits.
[7] has 3 bits.
The sorted array by bits is [0,1,2,4,8,3,5,6,7]
```

Example 2:

```
Input: arr = [1024,512,256,128,64,32,16,8,4,2,1]
Output: [1,2,4,8,16,32,64,128,256,512,1024]
Explantion: All integers have 1 bit in the binary representation, you should just sort
```

Example 3:

```
Input: arr = [10000,10000]
Output: [10000,10000]
```

Example 4:

```
Input: arr = [2,3,5,7,11,13,17,19]
Output: [2,3,5,17,7,11,13,19]
```

Example 5:

```
Input: arr = [10,100,1000,10000]
Output: [10,100,10000,1000]
```

Constraints:

- `1 <= arr.length <= 500`
- `0 <= arr[i] <= 10^4`

---

https://leetcode.com/problems/sort-integers-by-the-number-of-1-bits/ (https://leetcode.com/problems/sort-integers-by-the-number-of-1-bits/)

```
/*
Explanation:
1. Create Look up table for given length.
2. Now we know that we have maximum 32 bits in integer
   so number of 1 in given number will be in 0 to 32
3. Iterate over number for 0 to 32 let say i
    1) Check if given no of bits equal to i
        then sort numbers according to number who have equal no. of bits(equal to i)
*/
```

```
class Solution {
    public int[] sortByBits(int[] arr) {
        int len=arr.length;
        int ans[]=new int[len];
        int bit[]=new int[10001];
        for(int i=1;i<10001;i++)
        {
            bit[i]=bit[i>>1]+(i&1);
        }
        int index=0;
        for(int i=0;i<=32;i++)
        {
            int start=index;
            for(int j=0;j<len;j++)
            {
                if(bit[arr[j]]==i)
                {
                    ans[index]=arr[j];
                    index++;
                }
            }
            Arrays.sort(ans,start,index);
        }
        return ans;
    }
}
```

1394. Find Lucky Integer in an Array

Given an array of integers `arr`, a lucky integer is an integer which has a frequency in the array equal to its value.

Return *a lucky integer* in the array. If there are multiple lucky integers return the **largest** of them. If there is no lucky integer return **-1**.

Example 1:

```
Input: arr = [2,2,3,4]
Output: 2
Explanation: The only lucky number in the array is 2 because frequency[2] == 2.
```

Example 2:

```
Input: arr = [1,2,2,3,3,3]
Output: 3
Explanation: 1, 2 and 3 are all lucky numbers, return the largest of them.
```

Example 3:

```
Input: arr = [2,2,2,3,3]
Output: -1
Explanation: There are no lucky numbers in the array.
```

Example 4:

```
Input: arr = [5]
Output: -1
```

Example 5:

```
Input: arr = [7,7,7,7,7,7,7]
Output: 7
```

Constraints:

- `1 <= arr.length <= 500`
- `1 <= arr[i] <= 500`

1404. Number of Steps to Reduce a Number in Binary Representation to One

Given a number `s` in their binary representation. Return the number of steps to reduce it to 1 under the following rules:

- If the current number is even, you have to divide it by 2.
- If the current number is odd, you have to add 1 to it.

It's guaranteed that you can always reach to one for all testcases.

Example 1:

```
Input: s = "1101"
Output: 6
Explanation: "1101" corressponds to number 13 in their decimal representation.
Step 1) 13 is odd, add 1 and obtain 14.
Step 2) 14 is even, divide by 2 and obtain 7.
Step 3) 7 is odd, add 1 and obtain 8.
Step 4) 8 is even, divide by 2 and obtain 4.
Step 5) 4 is even, divide by 2 and obtain 2.
Step 6) 2 is even, divide by 2 and obtain 1.
```

Example 2:

```
Input: s = "10"
Output: 1
Explanation: "10" corressponds to number 2 in their decimal representation.
Step 1) 2 is even, divide by 2 and obtain 1.
```

Example 3:

```
Input: s = "1"
Output: 0
```

Constraints:

- `1 <= s.length <= 500`
- `s` consists of characters '0' or '1'
- `s[0] == '1'`

---

https://leetcode.com/problems/number-of-steps-to-reduce-a-number-in-binary-representation-to-one/ (https://leetcode.com/problems/number-of-steps-to-reduce-a-number-in-binary-representation-to-one/)

/* Explanation: First we have to consider extra 0 at MSB as it may possible that it will used when case '111' We have to perform 2 operation: if odd then LSB will always 1 so we have to add 1 Result after adding 1 will be all bits from LSB to till we find 0 will be 0 and first 0 from LSB will be 1 (Step++) (Do not decrement index as it will not reduce bit) Now for even we have to just remove LSB (i.e steps++ and index--)

*/

```
class Solution {
    public int numSteps(String s) {
        char arr[]=('0'+s).toCharArray();
        int steps=0;
        int len=s.length()+1;
        for(int i=len-1;i>=1;)
        {
            if(i!=1 && arr[i]=='1')
            {
                steps++;
                while(i>=0 && arr[i]!='0')
                {
                    steps++;
                    i--;
                }
                if(i>=0)
                    arr[i]='1';
            }
            else
            {
                steps++;
                i--;

            }
        }
        if(arr[0]=='0')
            return steps-1;
        return steps;
    }
}
```

1486. XOR Operation in an Array

Given an integer `n` and an integer `start`.

Define an array `nums` where `nums[i] = start + 2*i` (0-indexed) and `n == nums.length`.

Return the bitwise XOR of all elements of `nums`.

Example 1:

```
Input: n = 5, start = 0
Output: 8
Explanation: Array nums is equal to [0, 2, 4, 6, 8] where (0 ^ 2 ^ 4 ^ 6 ^ 8) = 8.
Where "^" corresponds to bitwise XOR operator.
```

Example 2:

```
Input: n = 4, start = 3
Output: 8
Explanation: Array nums is equal to [3, 5, 7, 9] where (3 ^ 5 ^ 7 ^ 9) = 8.
```

Example 3:

```
Input: n = 1, start = 7
Output: 7
```

Example 4:

```
Input: n = 10, start = 5
Output: 2
```

Constraints:

- `1 <= n <= 1000`
- `0 <= start <= 1000`
- `n == nums.length`

---

https://leetcode.com/problems/xor-operation-in-an-array (https://leetcode.com/problems/xor-operation-in-an-array)

```
//optimal
class Solution {
    public int xorOperation(int n, int start) {
        /**
            First of all, we need to know (2^4^6^8^10) == 2^(4^6^8^10) == 2^((4^6)^(8
^10)) = 2^(2^2) = 2

            (1) start >> 1, to match add 2*n as add n;
            (2) then &1 to check if start >> 1 is odd or even, so we will know last d
igit start with 0 or 1.
            (3) then &1 to figure out n is odd or even
            (4) start >> 1 is odd, then n is odd, result is (^pairs of 1)^ first one
                                   then n is even, result is (^pairs of 1)^ last one
^ first one
            (5) start >> 1 is even, then n is even, the result is (^pairs of 1)
                                    then n is odd, the result is (^pairs of 1)^last o
ne.
            (6) the final result: (result from step 4, 5) * 2 + ss;  ss is the last d
igit XOR result.
        **/
            int ss = 0;
            if( start % 2 == 1 && n % 2 == 1){
                ss = 1;
            }
            start = start>>1;
            if( (start & 1) == 1){
                if( (n & 1) == 1){
                    return (((n-1)/2 & 1) ^ start) * 2 + ss;
                }
                else{
                    return (((n-2)/2 & 1) ^ start ^ (start + n-1)) * 2 + ss;
                }
            }
            else{
                if( (n & 1) == 1){
                    return (((n-1)/2 & 1) ^ (start + n - 1)) * 2 + ss;
                }
                else{
                    return (n/2 & 1) * 2 + ss;
                }
            }
    }
}
```

```
//Brute Force
class Solution {
    public int xorOperation(int n, int start) {
        int answer=0;
        for(int  i=0;i<n;i++)
        {
            answer=answer^(start+2*i);
        }
        return answer;
    }
}
```

1525. Number of Good Ways to Split a String

You are given a string `s`, a split is called *good* if you can split `s` into 2 non-empty strings `p` and `q` where its concatenation is equal to `s` and the number of distinct letters in `p` and `q` are the same.

Return the number of *good* splits you can make in `s`.

Example 1:

```
Input: s = "aacaba"
Output: 2
Explanation: There are 5 ways to split "aacaba" and 2 of them are good.
("a", "acaba") Left string and right string contains 1 and 3 different letters respecti
("aa", "caba") Left string and right string contains 1 and 3 different letters respecti
("aac", "aba") Left string and right string contains 2 and 2 different letters respecti
("aaca", "ba") Left string and right string contains 2 and 2 different letters respecti
("aacab", "a") Left string and right string contains 3 and 1 different letters respecti
```

Example 2:

```
Input: s = "abcd"
Output: 1
Explanation: Split the string as follows ("ab", "cd").
```

Example 3:

```
Input: s = "aaaaa"
Output: 4
Explanation: All possible splits are good.
```

Example 4:

```
Input: s = "acbadbaada"
Output: 2
```

Constraints:

- `s` contains only lowercase English letters.
- `1 <= s.length <= 10^5`

---

https://leetcode.com/problems/number-of-good-ways-to-split-a-string/
(https://leetcode.com/problems/number-of-good-ways-to-split-a-string/)

```
class Solution {
    public int numSplits(String s) {
        int freq[]=new int[26];
        int temp[]=new int[26];
        int n=s.length();
        int ans=0;
        int len1=0,len2=0;
        for(int i=0;i<n;i++)
        {
            if(freq[s.charAt(i)-'a']==0)
                len2++;
            freq[s.charAt(i)-'a']++;
        }
        for(int i=0;i<n;i++)
        {
            char curr=(s.charAt(i));
            if(temp[curr-'a']<1)
            {
                len1++;
            }
            temp[curr-'a']++;
            if(freq[curr-'a']>0)
            {
                freq[curr-'a']--;
            }
            if(freq[curr-'a']==0)
                len2--;
            if(len1==len2)
                ans++;
        }
        return ans;
    }
}
```