**BIT MANIPULATION**

Table of Contents

[**Important Concepts** 2](#_Toc203856981)

[Binary Representation: 2](#_Toc203856982)

[Bitwise operators: 3](#_Toc203856983)

[Codes on bits: 3](#_Toc203856984)

[LEVEL 1: **EASY** 5](#_Toc203856985)

[1. Hamming Distance 5](#_Toc203856986)

[2. Counting Bits 5](#_Toc203856987)

[**LEVEL** 2: **Medium** 6](#_Toc203856988)

[1. Array Removal 6](#_Toc203856989)

[LEVEL 3: **Difficult** 7](#_Toc203856990)

[**SOLUTIONS:** 8](#_Toc203856991)

[**LEVEL 1:** 8](#_Toc203856992)

[**LEVEL 2:** 9](#_Toc203856993)

# **Important Concepts**

## Binary Representation:

############ convert decimal to binary #################

#Method 1: using bin

n = 12

bn = bin(n)[2:]

print(bn)  #1100

#Method 2: using recursion

def convert\_to\_bn(n):

    if n==0:

        return ""

    else:

        return convert\_to\_bn(n//2)+ str(n%2)

n = 12

ans=convert\_to\_bn(n)

print(ans)

############  convert decimal to binary  #################

#Method 1: using bin

binary\_str = "1100"

dc = int(binary\_str, 2)

print(dc)  #12

#Method 2: manual

def convert\_to\_decimal(binary\_str):

    ans = 0

    two\_power = 1

    for i in binary\_str[::-1]:

        ans += int(i)\*two\_power

        two\_power \*= 2

    return ans

binary\_str = "1100"

ans=convert\_to\_decimal(binary\_str)

print(ans)

## Bitwise operators:

| **Operator** | **Symbol** | **Example** | **Description** |
| --- | --- | --- | --- |
| AND | & | a & b | 1 if both bits are 1 |
| OR | | | a | b | 1 if either bit is 1 |
| XOR | ^ | a ^ b | 1 if bits are different |
| NOT | ~ | ~a | Inverts bits |
| Left shift | << | a << 1 | Multiply by 2 |
| Right shift | >> | a >> 1 | Divide by 2 |

a, b = 5, 3

print(a & b)  # 1

print(a | b)  # 7

print(a ^ b)  # 6

print(~a)     # -6 (2's complement)

#last bit is reserved to show sign of number(+(0),-(1))

#shifts

a = 5 #101

print(a<<1)  #5\*2 10

print(a<<2)  #5\*2\*2 20

print(a>>1) #5/2 2

print(a>>2) #5/(2\*2) 1

## Codes on bits:

############  check if kth bit is set  ############

def is\_kth\_bit\_set(n, k):

    return (n & (1 << k)) != 0

print(is\_kth\_bit\_set(5, 0))  # True, 5 = 101

print(is\_kth\_bit\_set(5, 1))  # False

print(is\_kth\_bit\_set(5, 2))  # True

############  Set / Clear / Toggle a Bit  ############

def set\_bit(n, k):     return n | (1 << k)

def clear\_bit(n, k):   return n & ~(1 << k)

def toggle\_bit(n, k):  return n ^ (1 << k)

print(set\_bit(5, 1))     # 7

print(clear\_bit(7, 1))   # 5

print(toggle\_bit(5, 0))  # 4

############  Check if Number is Power of 2  ############

def is\_power\_of\_two(n):

    return n > 0 and (n & (n - 1)) == 0

print(is\_power\_of\_two(8))  # True

print(is\_power\_of\_two(10)) # False

#concept: if num is power of 2, it has only 1 set bit

#eg: 8 = 1000, so n-1 = 0111, doing AND of both gives 0

############  XOR tricks  ############

# a ^ a = 0

# a ^ 0 = a

# a ^ b ^ a = b (XOR is commutative and associative)

############  Count set and unset bits  ############

n = 13 #1101

setbits=0

non\_setbits=0

while(n>0):

    if n&1>0:

        setbits+=1

    else:

        non\_setbits+=1

    n = n>>1

print(setbits,non\_setbits)

**Note**: Since binary operations work on bits, they are normally faster than other operations.

# LEVEL 1: **EASY**

### Hamming Distance

Link: <https://leetcode.com/problems/hamming-distance>

### Counting Bits

Link: <https://leetcode.com/problems/counting-bits/>

### Number of bit to make two integers equal

Link: <https://leetcode.com/problems/number-of-bit-changes-to-make-two-integers-equal/>

### Find XOR of numbers from L to R

Link: <https://www.geeksforgeeks.org/problems/find-xor-of-numbers-from-l-to-r/1>

### Sort Integers by the number of 1 bits

Link: <https://leetcode.com/problems/sort-integers-by-the-number-of-1-bits/>

### Single Number

Link: <https://leetcode.com/problems/single-number/>

### Missing Number

Link: <https://leetcode.com/problems/missing-number/>

### Find the XOR of Numbers Which Appear Twice

Link: <https://leetcode.com/problems/find-the-xor-of-numbers-which-appear-twice/>

### Convert a Number to Hexadecimal

Link: <https://leetcode.com/problems/convert-a-number-to-hexadecimal/>

# **LEVEL** 2: **Medium**

# LEVEL 3: **Difficult**

# **SOLUTIONS:**

## **LEVEL 1:**

1. Hamming Distance

#can also say this as, number of bits we need to flip too covert x to y

class Solution:

    def hammingDistance(self, x: int, y: int) -> int:

        xv = x^y           #xor sets bits that are different in x and y

        ans=0              #count of set bits

        for i in range(31):

            if (xv & (1<<i))!=0:

                ans+=1

        return ans

#to check if given bit is set, do and with (1<<i)

#xv & (1<<i), for i=2, 1<<2 = 100(4)

# i=5, 1<<5 = 100000(32)

1. Hamming Distance

class Solution:

    def countBits(self, n: int) -> List[int]:

        dp = [0]\*(n+1)

        if n==0: return [0]

        dp[1]=1

        p2=1

        for i in range(2,n+1):

            if i & (i-1) ==0:

                p2  = p2\*2

            dp[i] = 1 + dp[i-p2]

        return dp

#Analysis

# n=0[0] => 0     n=1[1] =>1

# n=2[10]=> 1     n=3[11] =>2

# n=4[100]=>1     n=5[101]=>2   n=6[110]=>2  n=7[111]=>3

# n=8[1000]=>1

#n=3 = 1 + (n-2) = 2

#n=7 = 1 + n[7-4] bits =3bits

To check if number is power of 2, do (n)&(n-1)==0

1. Number of Bit changes to Make Two integers Equal

class Solution:

    def minChanges(self, n: int, k: int) -> int:

        # Approach

        # 1. Iterate through bits in n and k:

        # n[i]: i-th bit of n

        # k[i]: i-th bit of k

        # if n[i]==1 and k[i]==0, count+=1

        # if n[i]==0 and k[i]==1, return -1

        # A. Bit manipulation

        # Time: O(log(min(n,k)))

        # Space: O(1)

        count = 0

        while n>0 or k>0:

            if n&1==1 and k&1==0:

                count+=1

            elif n&1==0 and k&1==1:

                return -1

            n>>=1

            k>>=1

        if n:

            return -1

        return count

1. Find XOR of numbers from L to R

Let check the pattern, p(n) = xor of nums from 1 to n

* p(1) = 1 p(5) = 1^2^3^4^5 = 1
* p(2)= 1^2 = 3 p(6) = 1^2^3^4^5^6 = 7
* p(3)=1^2^3 = 0 p(7) = 1^2^3^4^5^6^7 = 0
* p(4) = 1^2^3^4=4 p(8) = 1^2^3^4^5^6^7^8=8

class Solution:

    def findXOR(self, l, r):

        # Code here

        def xorInRange(N):

            if N%4==0: return N

            if N%4==1: return 1

            if N%4==2: return N+1

            if N%4==3: return 0

        return xorInRange(r)^xorInRange(l-1)

1. Sort Integers by The Number of 1 Bits Let check

We will use **Brian Kernighan’s algorithm:** Brian Kernighan's algorithm, also known as the "flip rightmost set bit" algorithm, is an efficient method for counting the number of set bits (1s) in a binary representation of an integer.

It says, n&(n-1) flips right most set bit

* **Eg**. n=6(110) , n-1 = 5(101) doing and of both = (100) unset right most set bit
* N=4(100), n-1=3(011) , n&n-1 = (000) unset right most set bit

class Solution:

    def countBits(num):

        count = 0

        #. Brian Kerninghan's algorithm

        while num > 0:

            count += 1

            num &= (num - 1)  # Clear the least significant set bit.

        return count

    def sortByBits(self, arr):

        arr.sort(key = lambda num: (Solution.countBits(num), num))

        return arr

1. Single Number

We know a^a =0

So if all numbers appearing twice, they will cancel each other. Leaving the number which appeared only once.

class Solution:

    def singleNumber(self, nums: List[int]) -> int:

        x=0

        for num in nums:

            x = x^num

        return x

1. Missing Number

class Solution:

    def missingNumber(self, nums: List[int]) -> int:

        n = len(nums)         #expected length

        x1, x2=0,0

        #expected xor

        for i in range(n+1):

            x1 ^= i

        #original xor

        for num in nums:

            x2 ^= num

        return x1^x2

1. Find the XOR of Numbers Which Appear Twice

class Solution:

    def duplicateNumbersXOR(self, nums: List[int]) -> int:

        s = set()

        result = 0

        for i in nums:

            if i in s:   #means appearing twice

                result ^= i

            s.add(i)

        return result

# class Solution:

#     def duplicateNumbersXOR(self, nums: List[int]) -> int:

#         # c = Counter(nums)

#         # xv=0

#         # for k,v in c.items():

#         #     if v==2:

#         #         xv^=k

#         # return xv

1. Convert a Number to Hexadecimal

class Solution:

    def toHex(self, num: int) -> str:

        if num==0:

            return "0"

        # If the input number is negative, convert it to its corresponding positive value using two's complement

        if num<0:

            num = (1<<32)+num

        hexs = "0123456789abcdef"  #hex codes

        hex\_num = ''

        while num>0:

            digit = num%16

            hex\_digit = hexs[digit]

            hex\_num = hex\_digit + hex\_num

            num //=16

        return hex\_num

## **LEVEL 2:**