**BIT MANIPULATION**

Table of Contents

[**Important Codes** 1](#_Toc171628510)

[Binary Search Tree Implementation: 1](#_Toc171628511)

[LEVEL 1: **EASY** 2](#_Toc171628512)

[1. Question 2](#_Toc171628513)

[2. Question 2](#_Toc171628514)

[LEVEL 2: **Medium** 3](#_Toc171628515)

[1. Array Removal 3](#_Toc171628516)

[LEVEL 3: **Difficult** 4](#_Toc171628517)

[**SOLUTIONS:** 5](#_Toc171628518)

[**LEVEL 1:** 5](#_Toc171628519)

[**LEVEL 2:** 6](#_Toc171628520)

# **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/>

3. <https://leetcode.com/problems/number-of-bit-changes-to-make-two-integers-equal/submissions/1696040022/?envType=problem-list-v2&envId=bit-manipulation>

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

3. <https://leetcode.com/problems/sort-integers-by-the-number-of-1-bits/?envType=problem-list-v2&envId=bit-manipulation>

4. <https://leetcode.com/problems/single-number/submissions/1695784950/?envType=problem-list-v2&envId=bit-manipulation>

5. <https://leetcode.com/problems/missing-number/submissions/1695980854/?envType=problem-list-v2&envId=bit-manipulation>

6. <https://leetcode.com/problems/find-the-xor-of-numbers-which-appear-twice/submissions/1695781188/?envType=problem-list-v2&envId=bit-manipulation>

7. <https://leetcode.com/problems/sum-of-all-subset-xor-totals/?envType=problem-list-v2&envId=bit-manipulation>

8. <https://leetcode.com/problems/convert-a-number-to-hexadecimal/submissions/1696277525/?envType=problem-list-v2&envId=bit-manipulation>

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

### Array Removal

Link: <https://www.codechef.com/problems/ARRREM>

# 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

## **LEVEL 2:**

1. Array removal

Problem 2, understand this and make your own explanation

[](https://www.youtube.com/watch?v=IqKZE6nwzko)

for \_ in range(int(input())):

    n = int(input())

    arr = list(map(int,input().split()))

    ans=float('inf')

    for i in range(32):

        val = (1<<i)-1

        num\_taken = 0

        bit\_or = 0

        for num in arr:

            if num>val:

                continue

            num\_taken+=1

            bit\_or = bit\_or | num

        if(bit\_or == val):

            num\_to\_remove = n - num\_taken

            ans = min(ans,num\_to\_remove)

    print(ans)