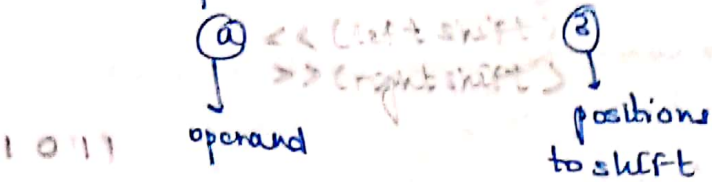
Day 9 . Bit Manipulation

(bit operations in c++)
(bitset)

*) main operations : OR, AND, XOR. (bitwise operations)




```
1011
0001
----
0001

1010
0001
----
0000
```

A ^ A = 0

A & 1 = 0 or 1 (depending on A being odd or even)

odd → 1

even → 0

```
1011
0000
----
1011
```

A ^ B ^ B = A

A ^ 0 = A (since B ^ B = 0)

Q) A : [1, 5, 8 . . .] → every element in A occurs twice except one. Find that element.

approach 1 : hash for frequency
O(n) space
2 passes of the array

↓ approach 2 :

XOR of all numbers, since the numbers occurring twice will result to zero and the XOR of the single freq. number with 0, will result in the number itself.

↓

O(1) space
1 pass of the array

variation : every element except one occurs thrice. Find the unique element.

```
bitset<32> foo;
foo.set();  → sets all bits
bitset<32> alt(20);
cout << foo << " " << alt << endl;
```

alt. to - ulong()
(converting bit pattern to number)

*) numbers can be assigned to bitset → special case.

*) indexing for bitset begins for LSB.

*) s1.flip()

*) s1.reset()

*) s1.to-string()

*) all numbers contribute 3 bits to any position. Now at any bit where the no. of occurences are of the form of 3X+1, we can say there we have the unique number's set bits.

↓

array unmodifiable

↓

$32 * N$ ($N \log N$) → range of any element

↓ no. of elements

↓ 32 → whereas sorting of computing would be $O(N \log N)$

Q. In an array containing numbers from 1 to N.

1 number in the array has been replaced by another number in the range 1 to N, such that all numbers in the array occur once except two i.e. A : missing number

B : exists twice

↓

Find A

↓

approach 1: iterate on the array and mark the A[i] as its -ve counterpart. Then when we pass again, the index in the array which has a +ve number is the missing element between 1 and N.

approach 2: calculate the sum of the normal array (S_1) and then the sum of mutated array (S_2).

we also calculate the sum of squares of normal array (S_1^2) and sum of squares of mutated array (S_2^2)

Now, $B - A = S_2 - S_1$

$B^2 - A^2 = S_2^2 - S_1^2$

$(B-A)(B+A) = S_2^2 - S_1^2$

$B + A = \frac{S_2^2 - S_1^2}{(B-A)}$

*) be careful for overflows.
↓
take difference of element first and then add

approach 2: XOR all numbers in the mutated array along with the ones in the normal array.

↓

for the value obtained $\rightarrow A \wedge B$

↓

select for any set bit in $A \wedge B$ and segregate the 2 arrays into 2 sets based on whether this bit is set or not. and take XORs of these two sets.

This operation gives 2 numbers one of which is the missing number, that can be checked by performing another pass on the array.

O(N)
↓
(no overflow issued)

Q. Given an array A and K, and we need to report whether there exist two elements such that

$A[i] \wedge A[j] = K$.

↓

$A \wedge B = K$, now $A = B \wedge K$

$B = A \wedge K$

If we hash the array and look for $A \wedge K$, we should be good for each element of the array.

Q. Given an array, find two elements of the array which have the min XOR value.

~~to minimize XOR between 2 values, their~~
~~set & unset bits should be same at as many~~
~~positions as possible~~

↓

so if we sort the array, then two consecutive elements should have the least XOR value candidate value.

↓

we have in a way reduced the search space from O(n²) to O(n)

↓

and according return the minimum value among these.

~~A < B < C~~
~~A^B < A^C~~

Q. Given an array, find two elements of the array which have the max XOR value

mask = 0, max = 0

i : 32 → 0

mask |= (1 << i) (mask gets appended a bit)

// another array where
ele[j] = A[i] & mask

candidate = max | (1 << i)

// solving as done in A^B = candidate in element array

if Yes, then max = candidate