Bit manipulation.

- $(7)_{10} = (111)_2$ and

2 | 7	1
2 | 3	1
1	

- $(13)_{10} = (1101)_2$

2 | 13	1
2 | 6	0
2 | 3	1
1	

and • $(1101)_2 = 1 + 0 + 4 + 8$

$= (13)$

$2^3 \; 2^2 \; 2^1 \; 2^0$

$8 + 4 + 0 + 1$

$\Rightarrow (13)$ Ans

```
string convert2binary (int n) {
    res = "";
    while (n != 1) {
        if (n % 2 == 1)
            res += '1'
        else
            res += '0'
        n = n/2;
    }
    reverse (res);
    return res;
}
```

TC = O(log n)
SC = O(log n).

```
int convert2decimal (string n) {
    int len = n.length();  p2 = 1,
    num = 0;
    For (i = len-1 → 0) {
        if (n[i] == '1')
            num = num + p2;
        p2 = p2 * 2;
    }
    return num;
}
```

TC = O(len)
SC = O(1).

- long long = 64 bits.

Now For 13 computer stores

→ 1's complement

(13) → (1101)$_2$

↓

(0010)$_2$

→ 2's complement

1) 1's complement
2) add 1 to it.

0010
+1
0011

0000 ... 000 | 1101

28 bits | 4 bits

- Computer All rest bits Zero.

→ AND, OR, XOR, LEFT, NOT.

$x = 13 \& 7 = (5)$

```
  1101
& 0111
------
  0101
```

all true = true
1 false = false

AND.

and $x = 13 \mid 8$
$= 13$

```
  1101
| 0111
------
 (1111)_2
```

all false = false
1 true = true

OR

and $x = 13 \wedge 8 = 10$

no. of 1's → odd = 1
no of 1's → even = 0

```
  1101
^ 0111
------
 (1010)
```

XOR.

Now >> (right) = $x / 2^k$ = $x >> k$

Eg: $x = 13 >> 1 = (6)$

```
  1101   = 13
(0 0 0 110)
     (6)
```

$13 / 2^1 = (6)$

Now For $x = -13$

So, 1101 → 13

1's complement = 0010

\+ 1

2's complement = 0011

Eg: $x = 13 >> 2 = (3)$

```
  1101
 (11)_2 = (3)
```

$13 / 2^2 = (3)$

Now << (left).

```
0 ... 1101
0 ... 11010
```

→ 26

Now largest no.

```
0 1 1 1 ... 1 1
```

So, $(2^{31} - 1)$ = INT_MAX.

Now smallest no.

-2^{31}

```
1 00 ... 0
0 1 1 1 1 1
```

```
011 .. 111
         +1
-----------
1 00 0 0 0
```

-2^{31} = INT_MIN.

Now num << k

$num \times 2^k$

13×2^1

(26)

→ Swap two numbers. → by using XOR operators.

```
temp = a;
a = b;
b = temp;
```

Eg. 5 ∧ 5 = 0

```
  101
∧ 101
-----
  000
```

Eg. a = a ∧ b

b = a ∧ b

= (a ∧ b) ∧ b

= a.

Now a = a ∧ b.

= (a ∧ b) ∧ ~~b~~ a

= b

→ Check if ith bit is set or not —

N = 13 i = 2

$(1101)_2$

yes

i = 1

(1101)

10

NO.

Now I.p can use left shift or right shift operator.

Eg. 1

N = 13 1101

i = 2

```
 1101
&0100
-----
 0100
```

1 << 2

1

10

100

Eg. N = 13

i = 1

```
 1101
 0010
-----
 0000
```

NO not set.

1 << 1

1

10

→ How to do this by right shift (>>)

Now N = 13 i = 2

3 2 1 0

1101

1101 >> 2

So 1101

& 0011

0001

Now if ((N & (1 << i)) != 0)

yes

else

NO.

```
| if ((N & (N >> i)) != 1)
* NO
| else
| yes.
```

→ Set the ith bit

N = 9 1001

i = 2 do or operation

```
 1001
 0100
-----
 1101 → 13.
```

so, take 1 << 2 i

10

100

→ clearing the ith bit

N = 13 (1101) → (1001)

i = 2

So, do 1 << 2

and do negation.

So, 000100

~ 111011

Now 000111

& .. 111011

000011

→ remove last set bit

N = 12

1100

& 1011 → 11

1000 done.

1000

→ toggle the ith bit

N = 13

i = 2

1101

^ 0100 do XOR

1001

Now

1 << 2

1

10

100

N ^ (1 << i).

OBSERVATION:

N = 16	N = 40	N = 84
10000	101000	1010100
N = 15	N = 39	N = 83
01111	100111	1010011

→ check if no. be power of two or not

N = 16 N = 13 N = 32

yes NO yes.

So, if (N & N-1 == 0)

yes

else

NO.

• If there is 1 set bit

so answer is yes

else no.

So, Eg (16)

10000

01111

00000 yes done.

→ count no. of set bits

↳ By using STL

__builtin_popcount(n)

03 Minimum bit flips to convert number

Eg. start = 10 goal = 7

1010

0111

total we have to change 3 bits.

1010 start
^ 0111 goal
1101 → ans

- no. of set bits in ans will be your ans we.

Now start = 3 goal = 4

Eg. 011 100

total we have to change 3 bits.

011
^ 100
111 → ans

3 set bits so ans.

04 Power set

Eg. nums = [1 2 3]

[] [1], [2], [3]

[1,2] [1,3] [2,3]

[1,2,3]

total = 8.

no. of subsets = 2^n

1 << n

	2	1	0
[] ←	0	0	0
[3] ←	0	0	1
[2] ←	0	1	0
[2,3] ←	0	1	1
[1] ←	1	0	0
[1,3] ←	1	0	1
[1,2] ←	1	1	0
[1,2,3] ←	1	1	1

done.

0 → don't take
1 → take

indexing in array.

TC = N × 2^n

SC = 2^n × N

05 Single number I

nums = [4, 1, 2, 1, 2]

1 ^ 1 = 0

2 ^ 2 = 0

4 ^ 0 = 4

* start XOR ke do. *

06 single number II

Eg. nums = [2,2,3,2]

all nos are appears thrice except one.
return that one.

Eg. 5 5 5 2 4 4 4

	2	1	0
5 →	1	0	1
5 →	1	0	1
5 →	1	0	1
2 →	0	1	0
4 →	1	0	0
4 →	1	0	0
4 →	1	0	0
	6	1	3

multiple of 3.

So, no index pe set hoga jo 3 ka divisible nhi hoga. agar 3 ka divisible nhi hua toh waha set bit dalo.

We concept of buckets.

nums = [2,2,3,2]

ones = 0
twos = 0
threes = 0

- nums[i] will go to ones, if not in twos
- nums[i] will go to twos, if it is in ones.
- nums[i] will go to threes, if it is in twos.

17 single number III

nums = [2,4,2,14,3,7,7,8]

XOR = (2^2) ^ (14^4) ^ (3^3) ^ (7^7)

0 ↓ 0 0

14^4 → 10

1110
0100
1010

Now 10 = 1010
9 = 1001
1000

Now 1000 ^ 1010 = 0010 → 2

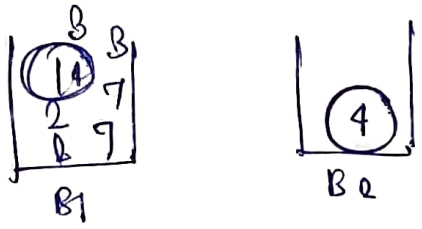
(num & num-1) ^ num.

14 = 1110
4 = 0100

- separate by buckets.

2

Now [2 4 2 14 3 7 7 8] • 14 & 4 always be in diff. bucket



turned 1 turned 0.

08 XOR of numbers in given range. | Now solve 1 to 2

N=4 1^2^3^4 = 4

```
Func (N){
    if (N%4 == 1) return 1
    else if (N%4 == 2) return N+1
    else if (N%4 == 3) return 0;
    else return N;
}
```

09 Divide two integers.

Eg 3 × 7

7 × (2^2 + 2^1 + 2^0)

(3 × 2^2) + 3(2^1) + 3(2^0)

12 6 3

So, 21 = 3 × 4 + 3 × 2 + 3 × 1

(3 × 4) + 6 + 3

12 + 6 + 3

= 21

So, 22 − 12 → 10 − 6 → 4 − 3 → 1

3 × 2^0 = 3

3 × 2^1 = 6

3 × 2^2 = 12

3 × 2^3 = 24

BIT MANIPULATION

01.) MINIMUM BIT FLIPS TO CONVERT NUMBER

```cpp
class Solution {
public:
    int minBitFlips(int start, int goal) {
        return __builtin_popcount(start^goal);
    }
};
```

02.) SUBSETS

```cpp
class Solution {
public:
    vector<vector<int>> subsets(vector<int>& nums) {
        int n = nums.size();
        int tot_sub = 1 << n;  // Total number of subsets is 2^n
        vector<vector<int>> ans;

        for (int i = 0; i < tot_sub; i++) {
            vector<int> temp;
            for (int j = 0; j < n; j++) {
                if (i & (1 << j)) {
                    temp.push_back(nums[j]);
                }
            }
            ans.push_back(temp);
        }
        return ans;
    }
};
```

03.) SINGLE NUMBER I

```cpp
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int ans=0;
        for(int i=0;i<nums.size();i++){
            ans=ans^nums[i];
        }
        return ans;
    }
};
```

BIT MANIPULATION

4.) SINGLE NUMBER II

CODE 01

```cpp
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int bitCount[32] = {0};

        // Count the number of 1s in each bit position
        for (int num : nums) {
            for (int i = 0; i < 32; ++i) {
                if (num & (1 << i)) {
                    bitCount[i]++;
                }
            }
        }

        // Reconstruct the single number from the bit counts
        int result = 0;
        for (int i = 0; i < 32; ++i) {
            if (bitCount[i] % 3 != 0) {
                result |= (1 << i);
            }
        }

        return result;
    }
};
```

CODE 02

```cpp
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        int n = nums.size();
        for (int i = 0; i < n; i += 3) {
            if (i == n - 1 || nums[i] != nums[i + 1]) {
                return nums[i];
            }
        }
        return -1;
    }
};
```

BIT MANIPULATION

CODE 03

```
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int ones = 0, twos = 0, threes = 0;

        for (int num : nums) {
            int ones_and_num = ones & num;
            twos = twos | ones_and_num;
            ones = ones ^ num;

            int ones_and_twos = ones & twos;
            threes = ones_and_twos;
            ones = ones & ~threes;
            twos = twos & ~threes;
        }

        return ones;
    }
};
```

BIT MANIPULATION

5.) SINGLE NUMBER III

CODE 01

```cpp
class Solution {
public:
    vector<int> singleNumber(vector<int>& nums) {
        long long int xorr = 0;
        for(int i = 0; i < nums.size(); i++){
            xorr = xorr ^ nums[i];
        }
        int rightmost = (xorr & (xorr - 1)) ^ xorr;
        int b1 = 0;
        int b2 = 0;
        for(int i = 0; i < nums.size(); i++){
            if(nums[i] & rightmost){
                b1 = b1 ^ nums[i];
            }
            else {
                b2 = b2 ^ nums[i];
            }
        }
        return {b1, b2};
    }
};
```

CODE 02

```cpp
class Solution {
public:
    vector<int> singleNumber(vector<int>& nums) {
        long long xorr = 0; // Use long long to avoid overflow
        for(int i = 0; i < nums.size(); i++){
            xorr = xorr ^ nums[i];
        }
        // Find the rightmost set bit using bit manipulation
        long long rightmost = xorr & -xorr;
        int b1 = 0;
        int b2 = 0;
        for(int i = 0; i < nums.size(); i++){
            if(nums[i] & rightmost)
                b1 = b1 ^ nums[i];
            else
                b2 = b2 ^ nums[i];
        }
        return {b1, b2};
    }
};
```

BIT MANIPULATION

6.) FIND XOR FROM 1 TO N

```
int computeXOR(int n) {
    // Initialize result variable
    int res = 0;

    // If n is multiple of 4
    if (n % 4 == 0)
        res = n;

    // If n % 4 gives remainder 1
    else if (n % 4 == 1)
        res = 1;

    // If n % 4 gives remainder 2
    else if (n % 4 == 2)
        res = n + 1;

    // If n % 4 gives remainder 3
    else if (n % 4 == 3)
        res = 0;

    return res;
}
```

BIT MANIPULATION

7.) FIND XOR IN GIVEN RANGE FROM L TO R

```cpp
#include <iostream>
using namespace std;

// Function to calculate XOR of numbers from 1 to n
int computeXOR(int n) {
    // Initialize result variable
    int res = 0;

    // If n is multiple of 4
    if (n % 4 == 0)
        res = n;

    // If n % 4 gives remainder 1
    else if (n % 4 == 1)
        res = 1;

    // If n % 4 gives remainder 2
    else if (n % 4 == 2)
        res = n + 1;

    // If n % 4 gives remainder 3
    else if (n % 4 == 3)
        res = 0;

    return res;
}

// Function to compute XOR from L to R
int rangeXOR(int L, int R) {
    return computeXOR(R) ^ computeXOR(L - 1);
}
```

BIT MANIPULATION

7.) DIVIDE TWO INTEGERS

```cpp
class Solution {
public:
    int divide(int dividend, int divisor) {
        if (dividend == INT_MIN && divisor == -1)
            return INT_MAX;

        unsigned long long dvd = abs((long long)dividend);
        unsigned long long dvs = abs((long long)divisor);

        long long sign = (dividend > 0) ^ (divisor > 0) ? -1 : 1;
        int quotient = 0;
        long long multiple = 1;

        while ((dvs << 1) <= dvd) {
            dvs <<= 1;
            multiple <<= 1;
        }

        while (dvd >= abs((long long)divisor)) {
            while (dvd >= dvs) {
                dvd -= dvs;
                quotient += multiple;
            }
            dvs >>= 1;
            multiple >>= 1;
        }

        return sign * quotient;
    }
};
```

THANK YOU !