Bit Manipulation.

(A)

1) AND (&) 4) << (left shift)
2) OR (|) 5) >> (right shift)
3) XOR 6) ~ (bitwise NOT)

Java → 32 bit representation

(B) If you do x = 3 and y = 6 ⇒ x & y

Step 1 → then first they are converted to binary code (i.e 0 & 1)
Step 2 → then AND operation takes place
Step 3 → and then result binary converted to number again.

(C)

AND

0 0 → 0
0 1 → 0
1 0 → 0
1 1 → 1

OR

1 1 → 1
0 1 → 1
1 0 → 1
0 0 → 0

XOR

1 1 → 0
0 0 → 0
1 0 → 1
0 1 → 1

Same inputs → 0
Diff inputs → 1.

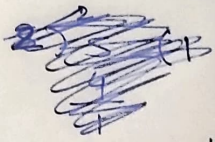
(D)

Binary to Number

1 0 1
2^2 2^1 2^0

↳ 4 + 0 + 1 ⇒ 5

Number to Binary



2 | 5 → 1
2 | 2 → 0
2 | 1 → 1
0

(E) Other operators

1) BITWISE NOT → Number to Binary → inversion (0 → 1, 1 → 0) → Binary to Number.

1s compliment.

⇒ In java neg num represented by $[-x = 2^{32} - x]$

↳ This is 2s compliment

$(2^{32} - 1 - y)$ ↳ +ve number.

Eg:- x = 5 & ~x ⇒ -6

1) Left Shift (<<) → Removes left most ignored, zero added at last.

(If x << y y > 30 < 32 number becomes -ve.)

x = 3;
→ x << 1

then 000 ··· 0011 → 3
000 ··· 0110 → Shift 1 → Now 6.

1) Similarly Right Shift (>>)

→ x = 3;
x >> 1

For +ve

00 ···· 0011 → 3
00 ····· 001 → 1

For signed R shift

For neg 1 added

111 ···· 10
1111 ···· 11

for unsigned → 0 fixed.

* → Note 1 is not shifted from 1 to last in any case, 1 is removed, 0 added in left & right both shifts.

→ $2^{32} - 1$ → All 32 ones.

→ Unsigned & Signed same for +ve.

Problems in Bit Manipulation.

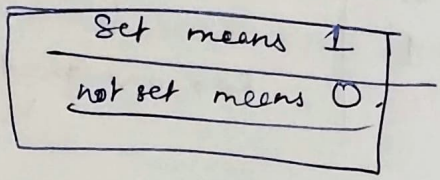
1) → Check if Kth bit is set or not

* I/p → n = 5, k = 1

  O/p → Yes

Ans 5 → 000 ... 0101 → 5

↳ k = 1



Program.

* In Java, K < 32. bcz 32 bits representation.

* Left Shift.

```
void KthBit(int n, int k)
{
  if(n & (1<<k-1) != 0)
    { print(Yes) }
  else { print(no) }
}
```

Right Shift

```
Void KthBit(int n, int k)
{
  if((n>>(k-1) & 1) == 1)
    { print(yes) }
  else { print(no) }
}
```

(n & 1) == 1

↳ Always put bracket, as == has higher precedence than &.

2) Count set bits (3 ways)

i) Naive.

→ Go through all 32 bits. O(n)

```
int countSet (int n)
{
    int res = 0;
    while (n > 0) {
    if (n % 2 == 0)
      { count ++; }
    n = n/2;
    }
}
```

Also be written as → `if((1 & n) == 1)`

→ `n >>= 1`

ii) Brian Kernighem's Algo

→ only O(m) m = no: of set bits.

Only while loop diff.

```
while (n > 0)
{ n = (n & (n-1))*;
  res ++;
}
```

n = 40
101000
1st iter → 100000
2nd iter → 000..0

→ drop exp:

* If you do n & (n-1) → you will remove the rightmost 1 in the n.

iii) Constant time

Lookup table method.

Divide into chunckes and count.

3) Power of two (or) not

① ⇒ { n%2 != 0 ⟶ false return
n = n/2 } till n==2 then return true. } Naive.

② Optimisation

→ Powers of 2 have only set bit = 1.
→ Then can use Brain Kerninghem method.

O(1) time.

---

4) Odd appearing number

① ⇒ Naive sol ⟶ 2 loops, O(n2)

② ⇒ Efficent ⟶ O(n) time, O(1) space using XOR operator. } ⟶ Props of XOR.

X ^ 0 ⟶ X
X ^ Y = Y ^ X
X ^ X = 0
X ^ (Y ^ Z) = (X ^ Y) ^ Z

```
int findOdd (int arr[], int n)
{
    int res = 0;
    for (int i=0 ; i<n ; i++){
        res = res ^ arr[i];
    }
    return res;
}
```

5 5 2 3 3
5 ^ 5 ^ 2 ^ 3 ^ 3
left.

Variation.

⇒ Can also be used to find a missing number.

⟶ { 1, 2, 4, 5 } Ans 3.

Ans take XOR of 1 to 5 then do and with given array.

6) Powerset using Bitwise

→ I/p → "abc"

O/p → "" "a" "b" "c" "ab" "bc" "ca" "abc"

```
① Void printPowerset(String str)
{   int n = str.length();
    int powsize = pow(2,n);

for(int counter=0; counter<powsize;
                        counter++)
{   for(int j=0; j<n; j++){
      if(counter & (1<<j) != 0)
        { print(str[i]);
        }
    }
    print("\n");
}
}
```

$\Theta(2^n * n)$