Bitwise Operators

a	b	a & b	a | b	a ^ b	~ a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Even # 1's : 0
odd # 1's : 1

} XOR }

Basic Properties

1) Even / odd

- odd:

1: 0 0 0 1
3: 0 0 1 1
5: 0 1 0 1
7: 0 1 1 1
9: 1 0 0 1

Even

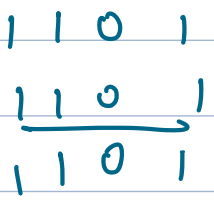
2: 0 0 1 0
4: 0 1 0 0
6: 0 1 1 0
8: 1 0 0 0
10: 1 0 1 0

```
  1 0 1 1 0 1
& 0 0 0 0 0 1
-------------
  0 0 0 0 0 1
```

```
if( A & 1 == 1 ) {
    "Odd"
} else {
    "Even"
}
```

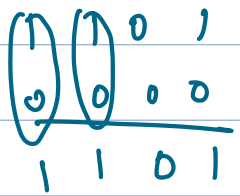
2) $A \& 0 = 0$

3) $A \& A = A$



4) $A | 0 = A$

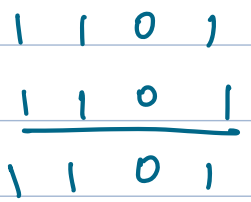
5) $A | A = A$



6) $A \hat{} 0 = A$

7) $A \hat{} A = 0$

$(A \hat{} A \hat{} A \hat{} A = 0)$

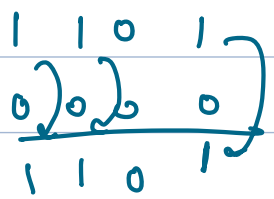


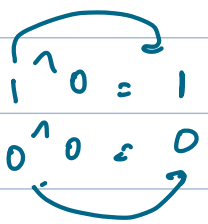
Commutative Property

$A \& B = B \& A$

$A | B = B | A$

$A \hat{} B = B \hat{} A$





Associative property

$$(A \,\&\, B) \,\&\, C = A \,\&\, (B \,\&\, C)$$

$$(A \mid B) \mid C = A \mid (B \mid C)$$

$$(A \wedge B) \wedge C = A \wedge (B \wedge C)$$

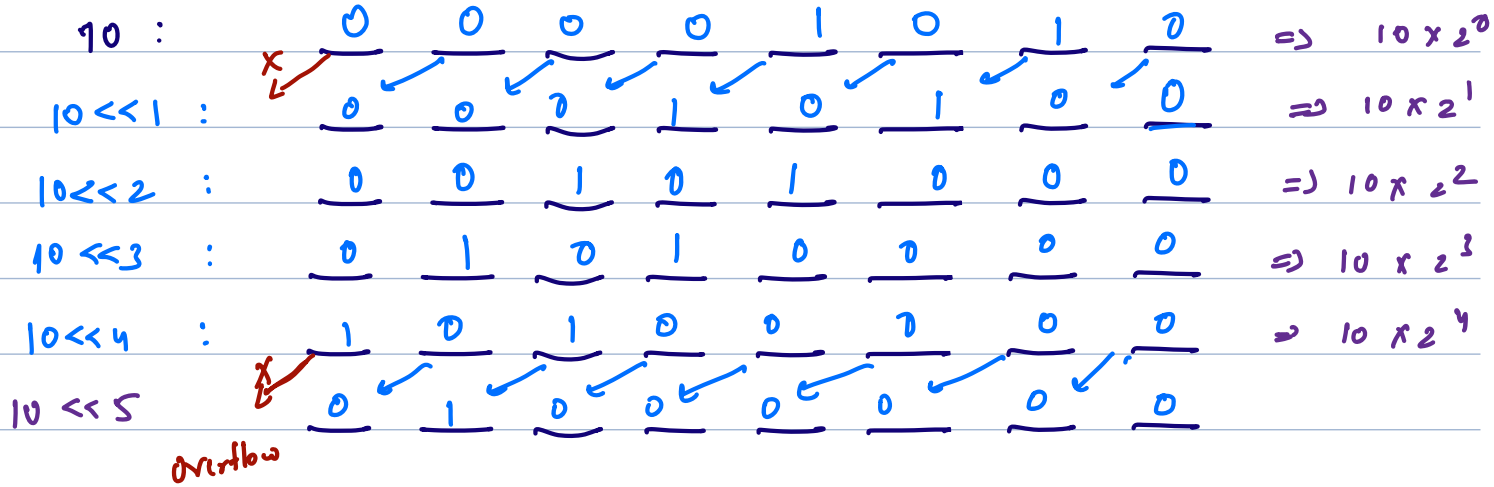
Quiz:

$$a \wedge b \wedge a \wedge d \wedge b = \underbrace{a \wedge a}_{0} \wedge \underbrace{b \wedge b}_{0} \wedge d = d$$

Quiz: $1 \wedge 3 \wedge 5 \wedge 3 \wedge 2 \wedge 1 \wedge 5 = \underbrace{1 \wedge 1}_{0} \wedge \underbrace{3 \wedge 3}_{0} \wedge \underbrace{5 \wedge 5}_{0} \wedge 2 = 2$

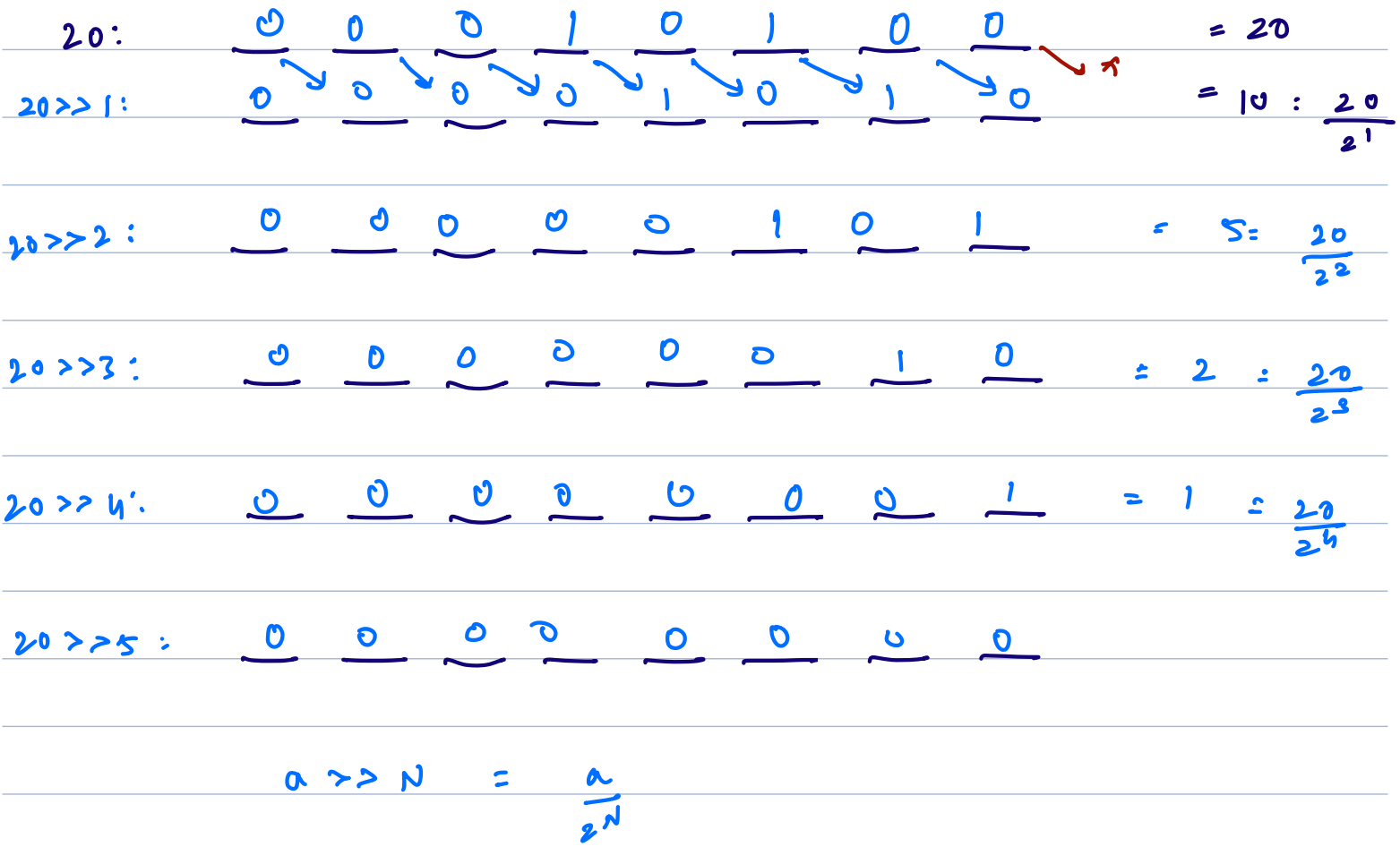
Quiz: $120 \wedge 5 \wedge 6 \wedge 6 \wedge 120 \wedge 5 = \underbrace{120 \wedge 120}_{0} \wedge \underbrace{5 \wedge 5}_{0} \wedge \underbrace{6 \wedge 6}_{0} = 0$

Left Shift (<<)



$a << N = a \times 2^N$ ✓ (when no overflow)

Right:



$a >> N = \frac{a}{2^N}$

Quiz: $1 << 3$

$a << N : \quad a \cdot 2^N$

$a = 1 \quad N = 3 \quad : \quad 1 \cdot 2^3 = 8$

$1 << 1 : \quad 2^1$

$1 << 2 : \quad 2^2$

$1 << 3 : \quad 2^3$

$\vdots$

$1 << N = 2^N$

T.C of all bitwise operations = $O(1)$

Question: Given an integer N and i, set the i^{th} bit

N = 22

i = 3

1 0 1 1 0

4 3 2 1 0

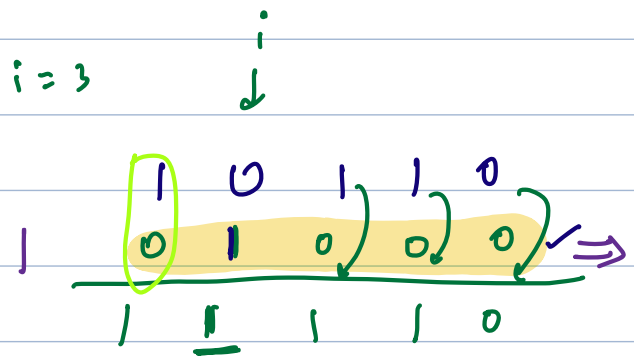
1 1 1 1 0 ⇒ 30

i = 4

4 3 2 1 0

1 0 1 1 0 ⇒ 22

i = 3



⇒ (1<<i)

x | 1 = 1

x | 0 = x

1<<1: 10

1<<2: 100

1<<3: 1000

```
int setBit( int N, int i ){
    return N | (1<<i);
}
```

Question: Toggle i^{th} bit

Ex:

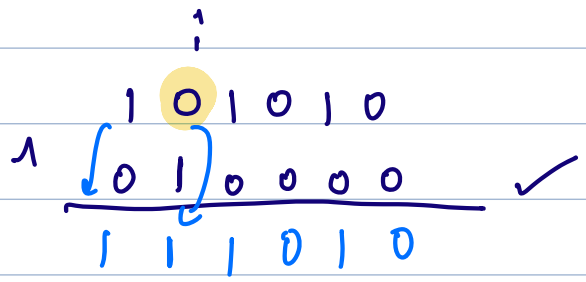
i = 2

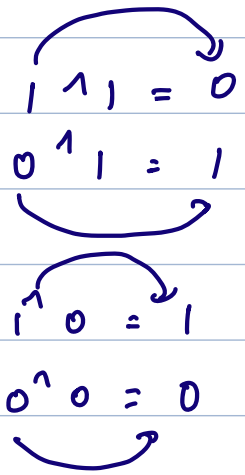
i = 1

1 0 1 => 0 0 1 = 1

1 0 1 => 1 1 1 => 7

2 1 0






```
int toggle(int N, int i){
    return N^(1<<i);
}
```

Question: Check if i^{th} bit is set or unset

N =

4 3 2 1 0

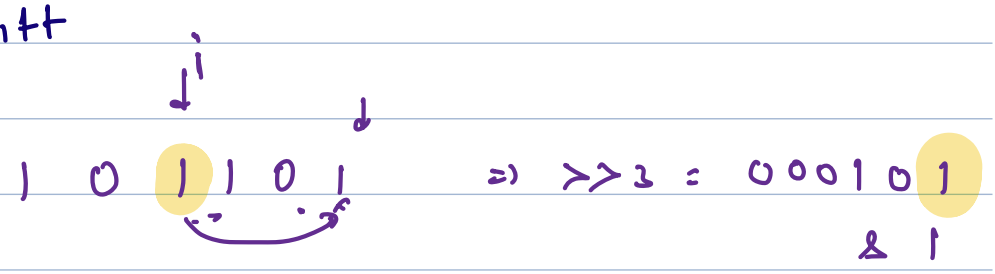
1 0 1 1 0

i = 1 => True

i = 3 => False

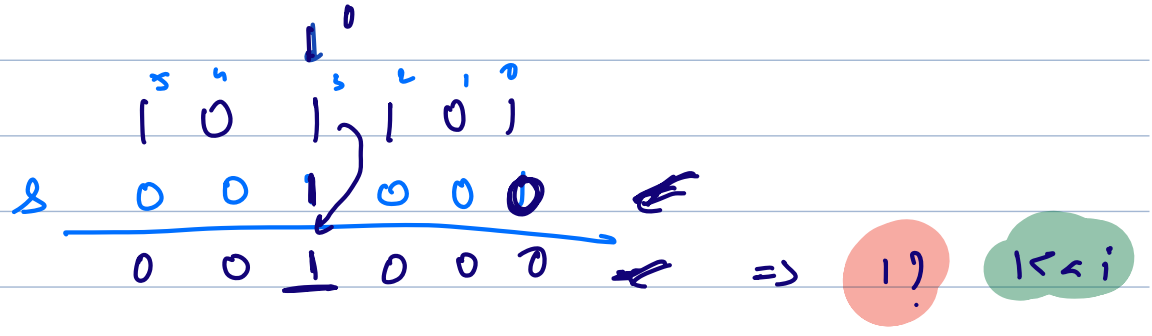
i = 0 => false

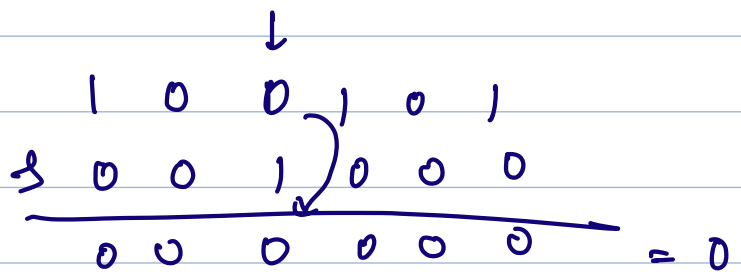
Way 1: Right Shift




```
if( (N >>3) & 1 == 1 ){
        " SET "
}
else {
        " UNSET "
}
```

Way2: Left Shift






```
if( N & (1<<i) == 0 ){
        UNSET
}else{
        SET
}
```

HW: Unset the i^{th} bit

Question:

We are given an integer array where every number occurs twice except for one number which occurs just once. Find that number.

A: 4 5 5 4 1 6 6 => 1

A: 7 5 5 1 7 6 1 6 4 => 4

Approach 1:

A: 7 5 5 1 7 6 1 6 4 :- 7 7 5 5 1 1 6 6 4

0 0 0 0 4

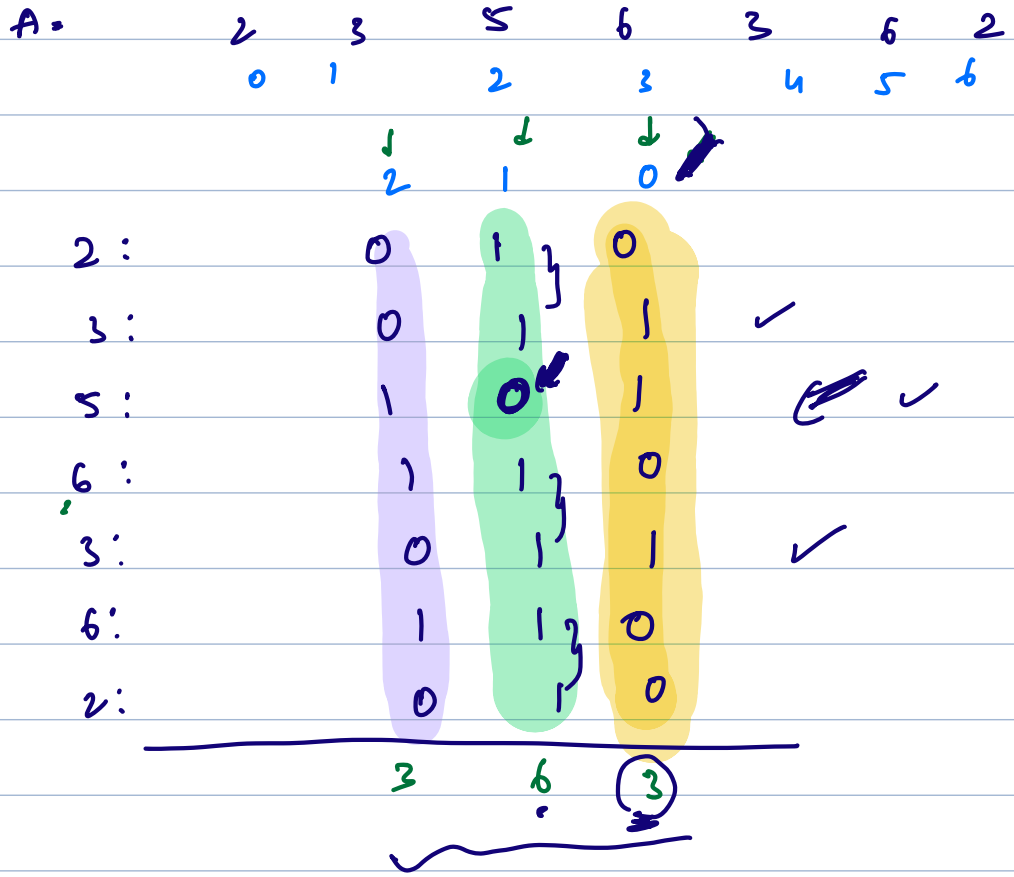
```
ans=0;
for(i=0; i<N; i++){
    ans = ans ^ A[i];
}
```

T.C: O(N)

S.C: O(1)

8:23

Approach:-



Ans: 1 0 1
2 1 0

```
if( #set bits is odd at ith position){
    ith bit is set in unique num ✓
} else {
    ith bit is unset in unique num
}
```

```
ans = 0;

for(i -> 0 to 31) { // go to every bit one by one
    cnt = 0;

    for(j -> 0 to arr.size - 1) { // iterate on array

        // check if ith bit is set
        if((arr[j]& (1<<i))cnt++;
    }

    if(cnt & 1)// If the count is odd
        ans = ans | 1 << i; // set ith bit in ans
}

print(ans);
```

T.C: $32 * N \Rightarrow O(N)$

S.C: $O(1)$

Question: All numbers occur twice except 2 numbers which occur only. Find these unique numbers

A: 4 5 4 1 6 6 5 2

Approach1: Sort and iterate

T.C: O(NlogN)

S.C: O(1)

Approach2: Hashmap / Hashset

T.C: O(N)

S.C: O(N)

Approach3:

A: 4 5 4 1 6 6 5 2

XOR(A) = 1 ^ 2 = 3

3 → 0, 2 → 1

0 0 1
0 1 0
0 1 1
2 1 0

$a \wedge b = 3$ — 0^{th} bit (OR) 1^{st} bit

At 0^{th} and 1^{st} bit, a and b have different bits

→ Let's divide array into 2 groups:

group1: 0^{th} bit is 0

group2: 0^{th} bit is 1

→ a & b will diff belong to different groups

XOR = $2 \wedge 4$ = 6

a 0 1 0

b 1 0 0

1 1 0

2 1 0

↑ i = 1

2 ↓ 0

1 1 0

Steps:

1) Find XOR of entire array
2) Find any i^{th} bit in XOR which is set
3) Divide into 2 groups and find XOR of these 2 groups

```
xorAll = 0;

// XOR of all numbers in the array
for (i -> 0 to N - 1) {
    xorAll ^= A[i];
}

// Find the rightmost set bit position
// Note: Any other bit can be used as well
declare pos

for(pos = 0; pos < 32; pos++)
{
    if(checkbit(xorAll,pos))
        break;
}

num1 = 0;
num2 = 0;

// Divide the array into two groups based on the rightmost set bit
for (i -> 0 to N - 1) {
    if (checkbit(A[i], pos)) {
        num1 ^= A[i];
    } else {
        num2 ^= A[i];
    }
}

print(num1);
print(num2);
```

Step1 O(N)

Step2 O(1)

Step3 O(N)

// xor of group1

// xor of group2

// Add to group

T.C: O(N)

S.C: O(1)