storage ☆	char	short	int	long	. .
range / size	16 bit	16 bit	4 byte	8 byte	
operator's			32 bit	64 bit	

bit

1 byte → 8 bit

∞ 1 nibble → 4 bit

sign bit

+V

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

-V

1000	8	-0 -8
1001	9	-1 -7
1010	10	-2 -6
1011	11	-3 -5
1100	12	-4 -4
1101	13	-5 -3
1110	14	-6 -2
1111	15	-7 -1

0010

+1

0011

0 2 -0 -2 11

2's compl

n → (1's complement) +1

n = $(0110)_2$

1

1's comp = 1001

+1

1010

f
-

1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

+1

-2^3

-2^{n-1}

1 1 1 1
1 1 1 1
+1
0 0 0 0

n = 4

7
+1
8

1 1
0 1 1 1
+1
1 0 0 0

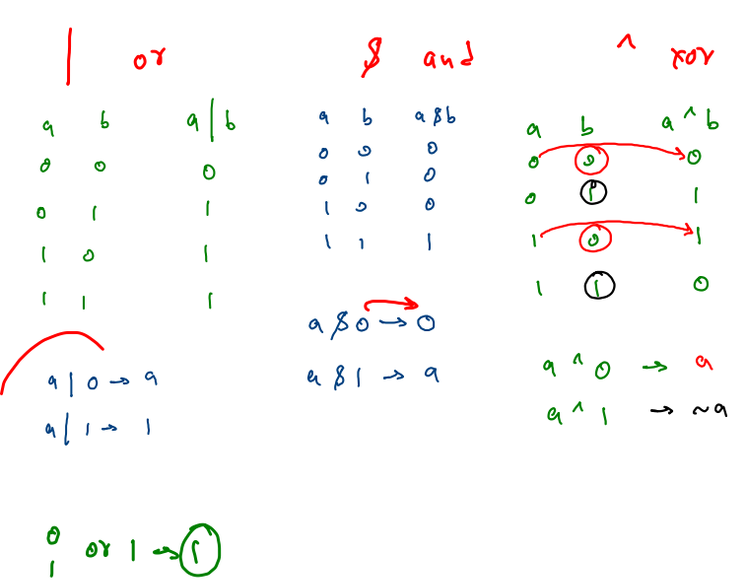
$2^3 - 1$

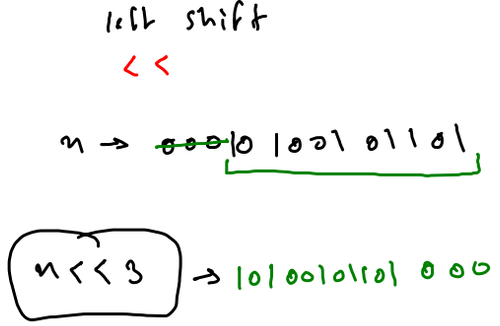
$2^{(n-1)} - 1$

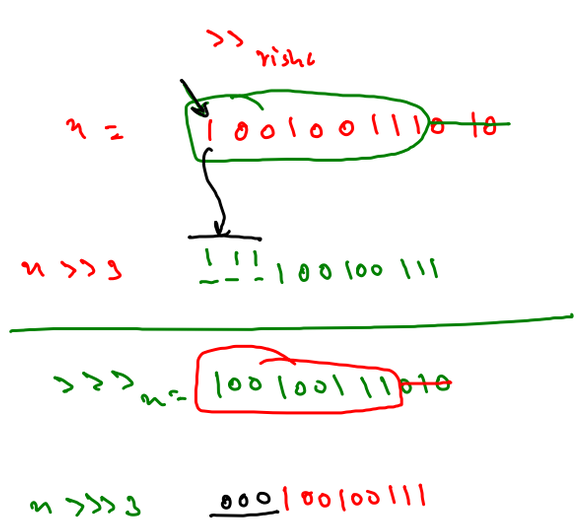
char	short	int	long
1 byte	2 byte	4 byte	8 byte
8 bit	16 bit	32 bit	64 bit

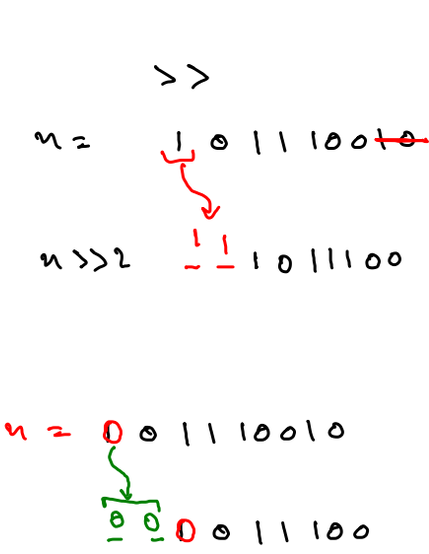
min -2^{31} -2^{63}

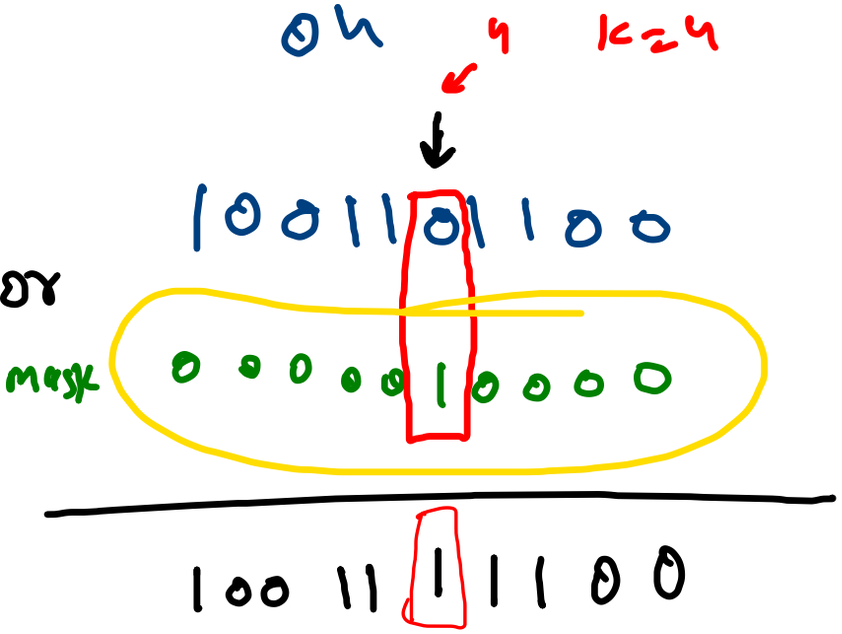
max $2^{31}-1$ $+2^{63}-1$

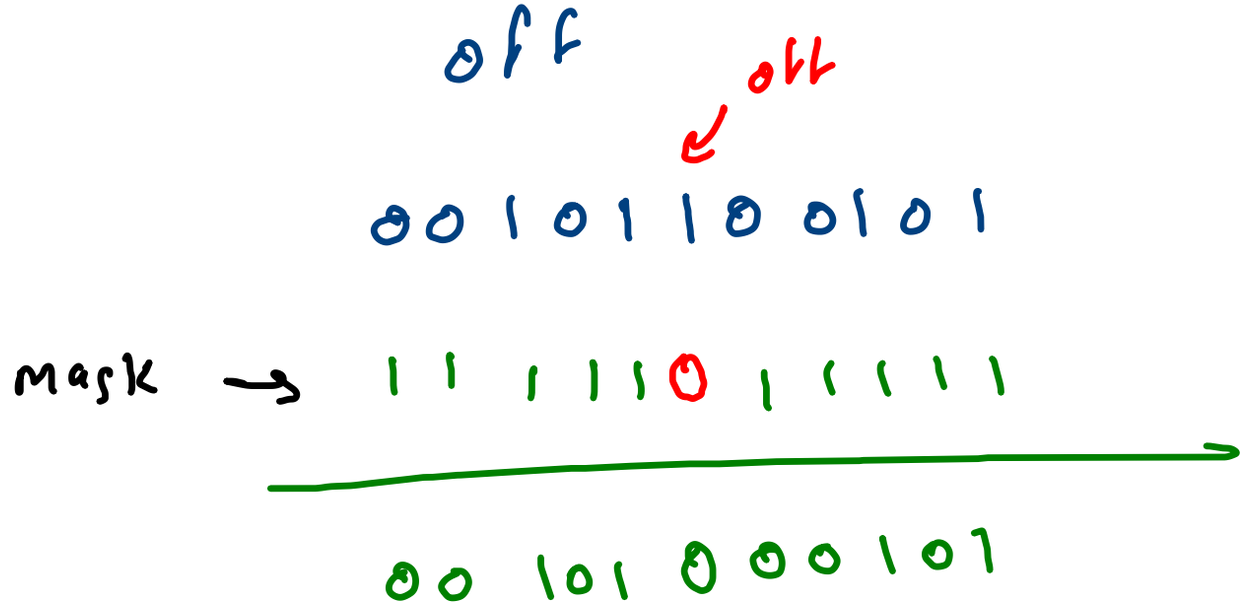

| or
& and
^ xor
a b a|b
0 0 0
0 1 1
1 0 1
1 1 1
a|0 → a
a|1 → 1
a b a&b
0 0 0
0 1 0
1 0 0
1 1 1
a&0 → 0
a&1 → a
a b a^b
a ^ 0 → a
a ^ 1 → ~a
0 or 1 → 1


left shift
<<
n → 10110100101000
n<<3 → 10100101000 000


>> right
n = 10010011110 10
n >> 3 111 10010011
>>> n = 10010011110 10
n >>> 3 000 10010011


>>
n = 10111001 0
n >> 2 10111001
n = 00111001 0
00 00111001

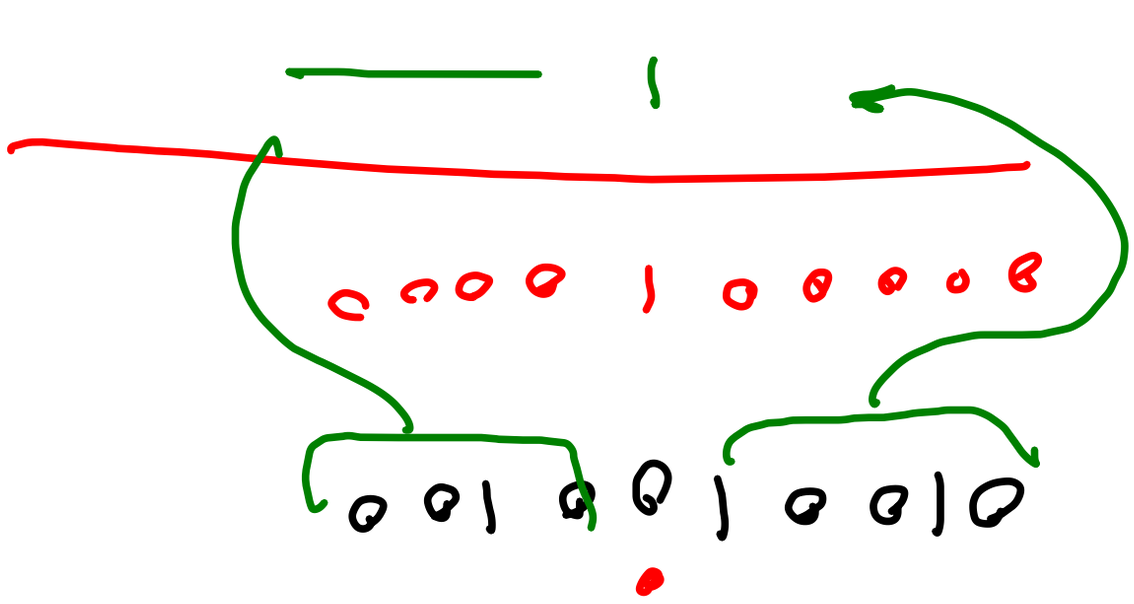


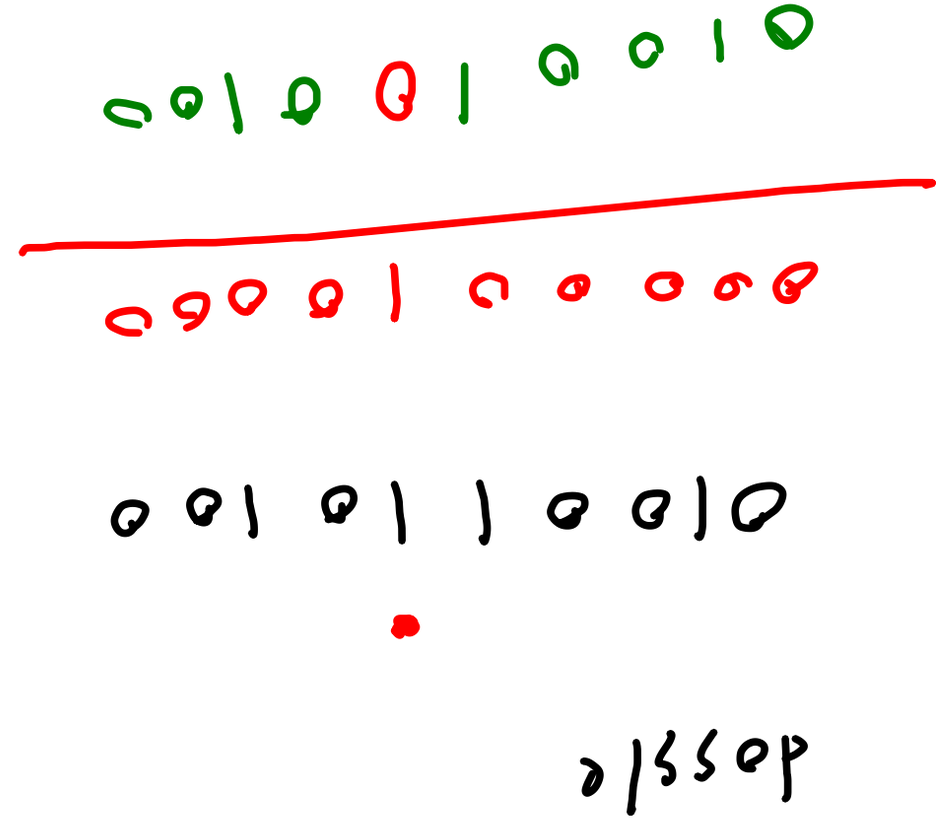


$n = (1)_{10} \rightarrow (00000001)$

$1 << k$

n



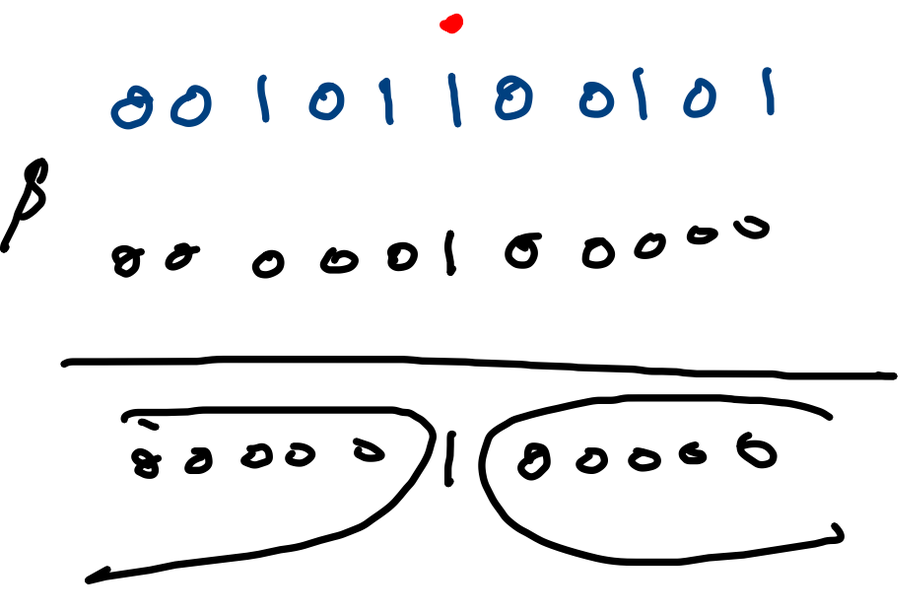


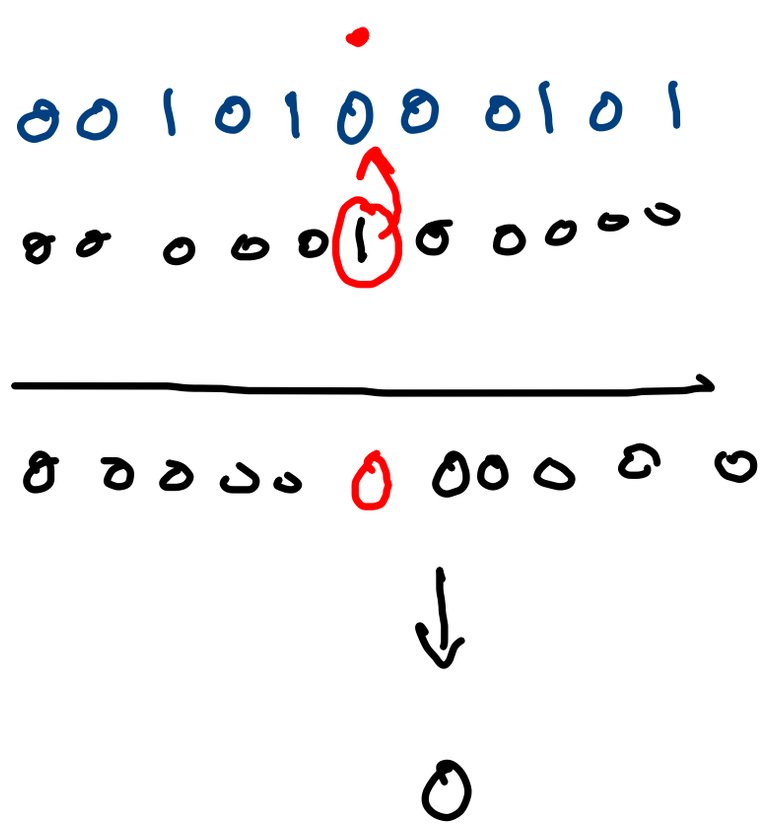
and

check

0 → false

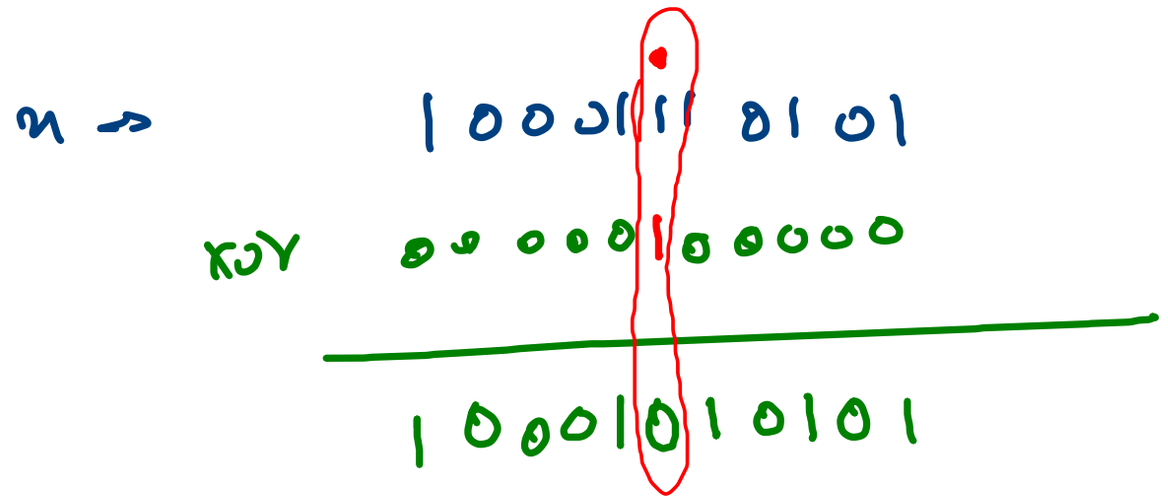
1 → True

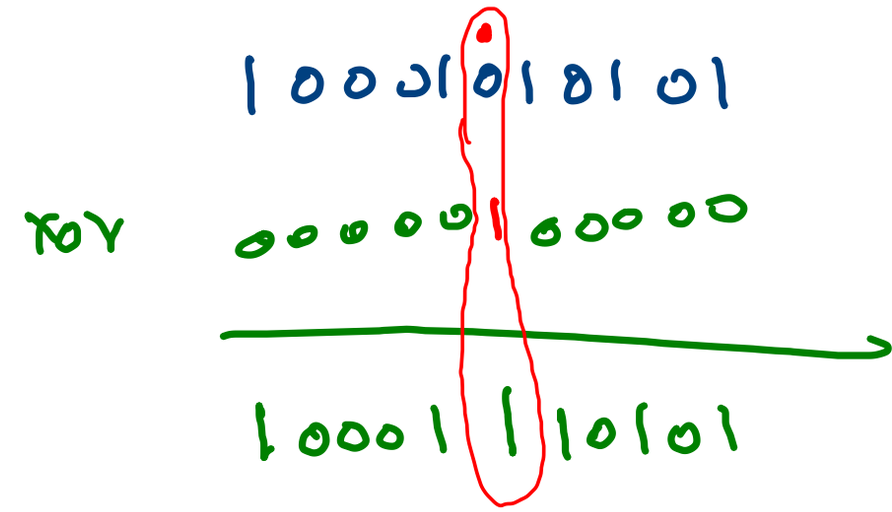




2^k


n = 2n





Fenwick Tree

Bit manipulation

O(1)

Sparse table →

8

4

int

32

00100110

01

10

01100000

100000

TC. → O(1)

n → 0 1 0 0 1 1 0 0

~n → 1 0 1 1 0 0 1 1

~n + 1 → 0 0 0 0 0 0 0 1

1 0 1 1 0 1 0 0

n & -n → 0 0 0 0 0 1 0 0

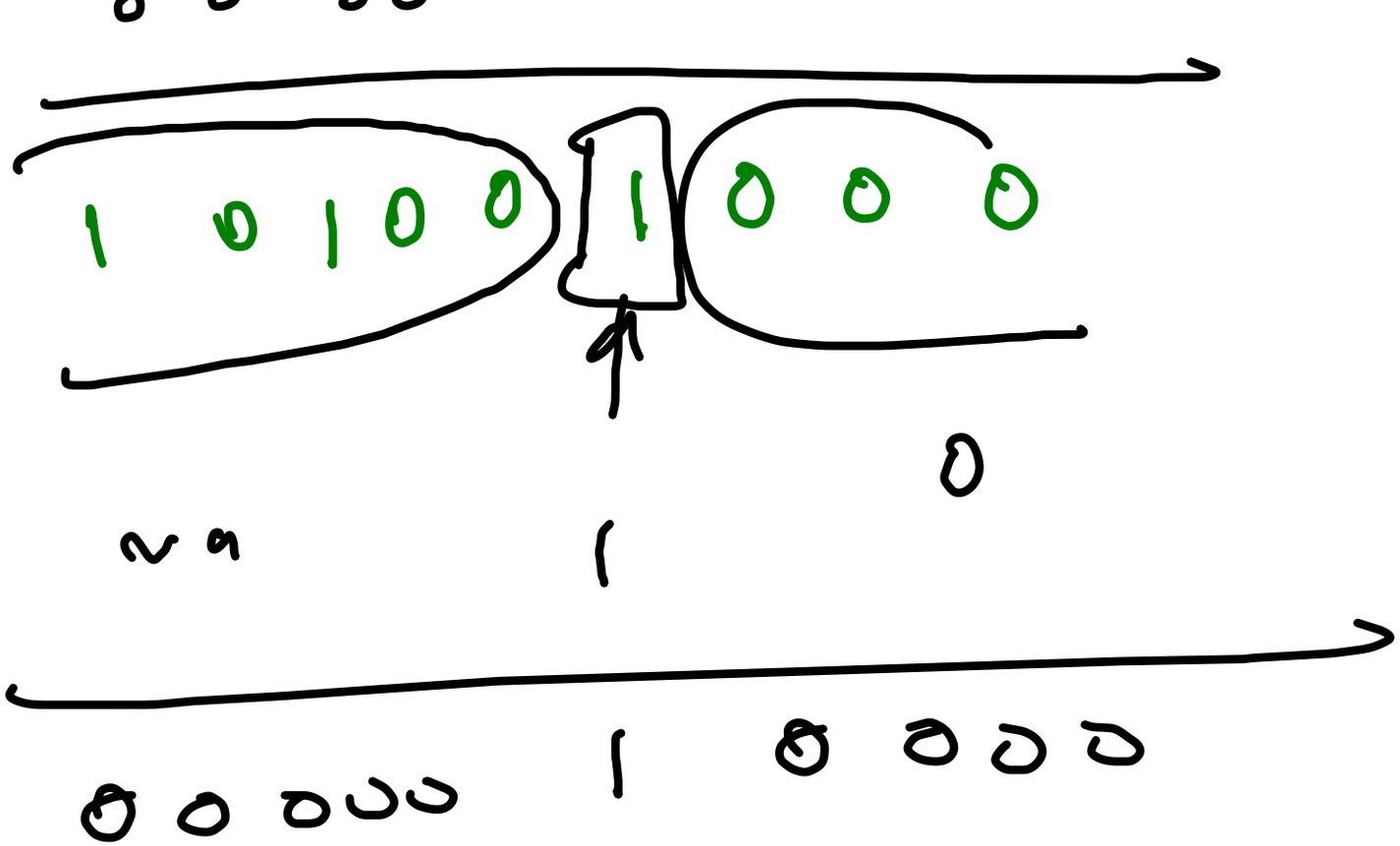
n & i'n

~n + 1

$n \rightarrow$ 0 1 0 1 1 1 0 0 0

$\sim n \rightarrow$ 1 0 1 0 0 0 1 1 1

$\sim n + 1$



$a n^2$

$(58)_{10}$

n → 0 10111000

(4) =

32

n → 0 1 0 1 1 1 0 0 0

count

1

mask → 0 0 0 0 0 1 0 0 0

0 1 0 1 1 0 0 0 0

n>0

0 0 0 0 0 1 0 0 0

0 1 0 1 0 0 0 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 → 0

[12 , 11 , 10 , 11 , 24 , 12 , 24]

↑

1

xcx

a	b	a ^ b
0	0	0
1	1	0
0	1	1
1	0	1

$a \wedge 0 \to a$

$a \wedge 1 \to \sim a$

$a \wedge 0 = a$

$a \wedge a = 0$

$\star\ (a \wedge b) \wedge c = a \wedge (b \wedge c)$

$a \wedge b \wedge b = c \wedge b$

$a \wedge b \wedge b = c \wedge b$

$a \wedge 0 = c \wedge b$

$a = c \wedge b$

$a = c \wedge b$

[13 , 11 , 10 , 11 , 27 , 13 , 27]

13 ^ 11 ^ 10 ^ 11 ^ 27 ^ 13 ^ 27

13 ^ 13 ^ 11 ^ 11 ^ 27 ^ 27 ^ 10

0 ^ 0 ^ 0 ^ 10

0 ^ 10 → 10

13, 11, 11, 13, 10

1101 ^ 1011 ^ 1011 ^ 1101 ^ 1010

0110 ^ 1011 ^ 1101 ^ 1010

1101 ^ 1101 ^ 1010

0 ^ 1010 = 10

$10 \rightarrow 1010$

$9 \rightarrow 1001$

[13, 11, 10, 11, 27, 13, 27, 9]

$(10 \wedge 9) \wedge 10 = 9$

$XOR = 10 \wedge 9$

$= 1010 \wedge 1001$

$= 0011$

10

9

13 → 01101

11 → 01011

10 → 01010

9 → 01001

27 → 11001

xor → 10 ^ 9

= 10 01010 ^

9 ^ 01001

= 00011

0 1

G1 G2

10

13 , 11 , 10 , 11 , 27 , 13 , 27 , 9

G1

G2 G2 G1 1 2 2 1

13
11
11
27
13
27
9

```
011011   a
110111   b
010101   c
010011   d
110011   e
```

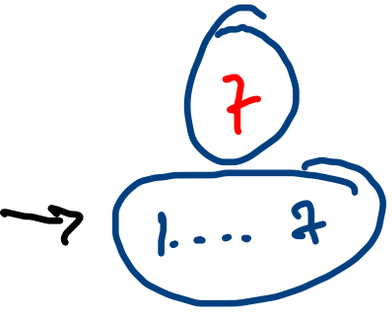
```
  110111   b
  010011   d
  ---1--
^ 100100

rsb -> 000100
```

--0-- : 0, G1: [a, a], [e, e]; xor -> d

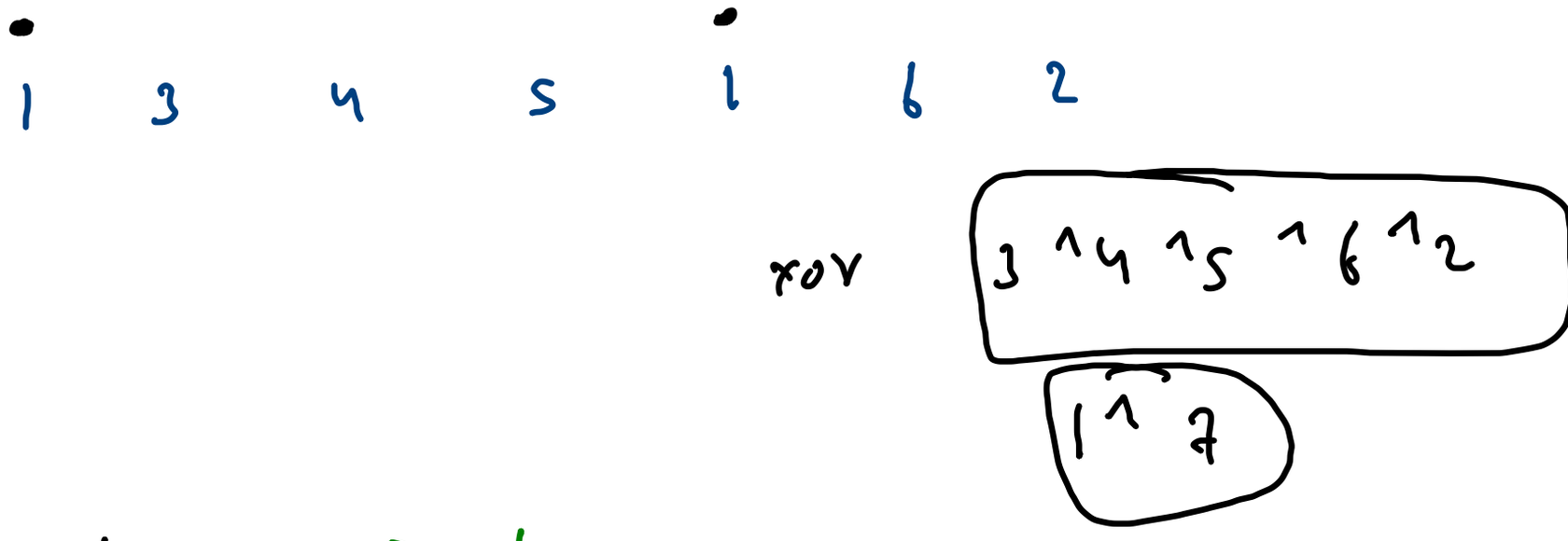
--1-- : 1, G2: [c, c]; xor -> b

```
G1  G1  G2  G2  G2  G1  G1  G1
a,  a,  b,  c,  c,  d,  e,  e
```

xor => b ^ d

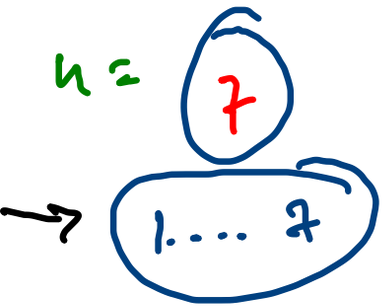


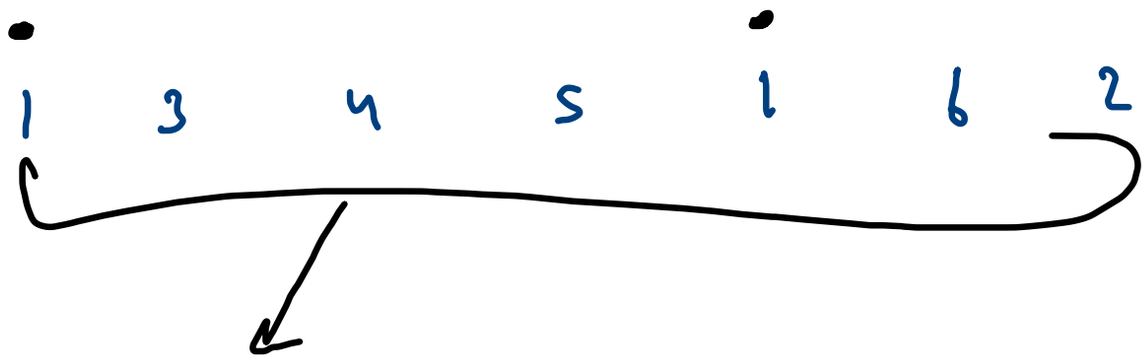
1 3 4 5 1 6 2

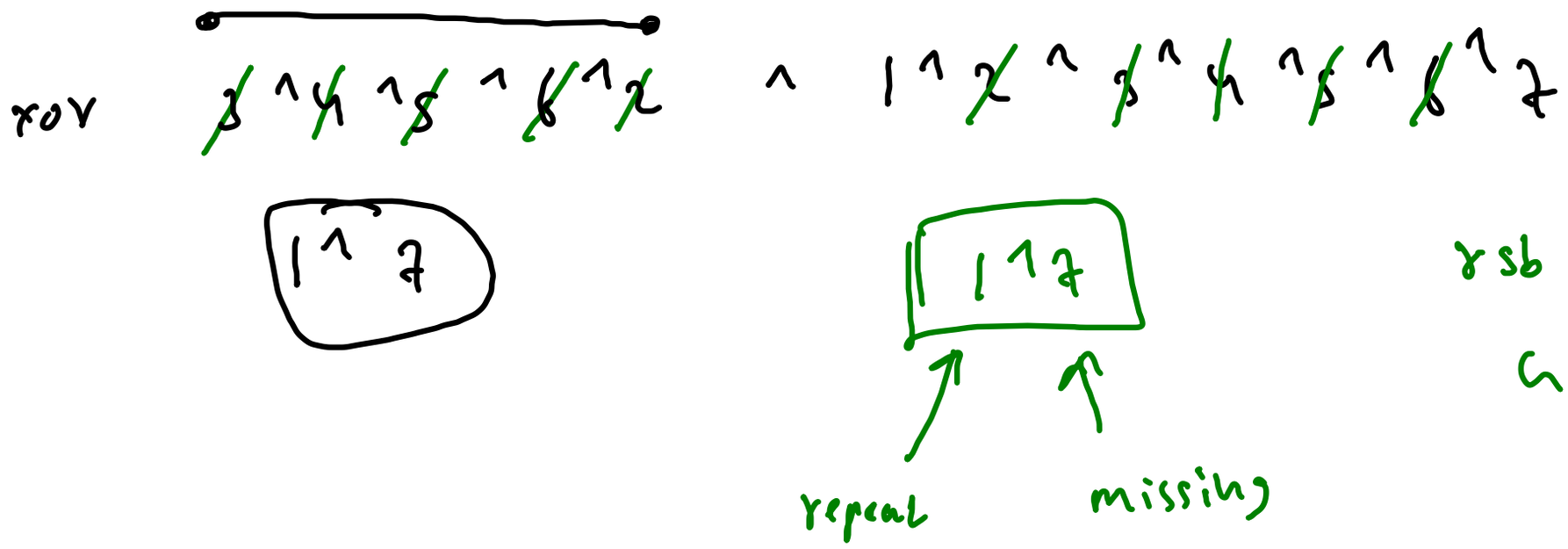


1 repeat → 1

1 missing → 7


n =
7
1 7


1 3 4 5 1 6 2


xor
1 ^ 7
1 ^ 7
repeat
missing
xsb
n1
n2

1	001
2	010
3	011
4	100
5	101
6	110
7	111

G1 G2 G1 G1 G1 G2 G2

1 3 4 5 1 6 2

XOR → 1 ^ 7

→ 001 ^ 111

→ 110

0 G1 → 1 → 4 → 5 → 1

1 G2 → 3 → 6 → 2 → (3 ^ 6 ^ 2) ^ 1 ^ 5 → ^ 6 ^ 7

[4 7 5] ^ 1 ^ 4 7 5

```java
int xor = 0;

for(int v: arr){
    xor ^= v;
}

for(int i=1;i<=arr.length;i++){
    xor ^= i;
}

// xor = repeating ^ missing
int rsb = xor&-xor;

int a = 0;
int b = 0;

for(int v: arr){
    if((v&rsb)==0){
        a ^= v;
    }else{
        b ^= v;
    }
}

for(int v=1;v<=arr.length;v++){
    if((v&rsb)==0){
        a ^= v;
    }else{
        b ^= v;
    }
}

for(int v: arr){
    if(v == a){
        System.out.println("Missing Number -> "+b);
        System.out.println("Repeating Number -> "+a);
        return;
```

You are sc

[1 3 4 5 1 6 2]

xor = 3 ^ 4 ^ 5 ^ 6 ^ 2

xor = (3 ^ 4 ^ 5 ^ 6 ^ 2) ^ 1 ^ 2 ^ 3 ^ 4 ^ 5 ^ 6 ^ 7

xor = 1 ^ 2 = (110)$_2$

1 → repeating, 2 → missing

rsb → 010

1	001
2	010
3	011
4	100
5	101
6	110
7	111

010

a = 1 ^ 4 ^ 5 ^ 1 → 4 ^ 5 ^ 4 → 0

b = 3 ^ 6 ^ 2 → 3 ^ 6 ^ 2 ^ 5 → 1

a = ~~4 ^ 5~~ ^ 1 ^ ~~4 ^ 5~~ → 1

b = ~~3 ^ 6~~ ^ ~~2~~ ^ 3 ^ ~~6~~ ^ 7 → 7