Bit Magic

Bitwise operators in C++ (Part 1)

AND & (When both are 1 then give 1) else 0.
OR |
XOR ^ (When different then 1 if same then 0

Bitwise AND

eg $x = 3$
$y = 6$

Binary Representation

x: 00...0011
y: 00...0110
(x&y): 00...0010

cout << (x&y) // 2

Bitwise OR

x: 3
y: 6

Binary Representation

x: 00...0011
y: 00...0110
(x|y): 00...0111

Page No.:
Date:

Bitwise XOR

x : 3
y : 6
Binary Representation
x : 00...0011
y : 00...0110
(x^y) : 00...0101

cout << (x^y) // 5

Bitwise operation in C++ (Part 2)

→ **Left Shift Operator**

x : 3

x : 000...011

(x<<1) : 000...110 ← Shift to left by 1 add zero at end

(x<<2) : 00...1100 ← Shifted to left by 2 added zeroes

(x<<y) : 00....110000

```
int x = 3
x<<1   // 6
x<<2   // 12
x<<4   // 48
```

* If we assume that the leading y bits are 0, then result of (x<<y) is equivalent to $x * 2^y$

OR,

Assuming that last y bits in the binary representation of x are ze

```
int x = 3
x<<1   // 6  → 3 X 2^1
x<<2   // 12 → 3 X 2^2
x<<4   // 48 → 3 X 2^4
```

→ Right Shift operator >>

x : 000 ... 0100001 ← last bit are ignored. Shift one right
(x>>1): 000 ... 0010000
(x>>2): 000 ... 0001000
(x>>y): 000 ... 0000010

int x = 33
x>>1 // 16
x>>2 // 8
x>>4 // 2

* x>>y is equivalent to floor of $\left\lfloor \frac{x}{2^y} \right\rfloor$

x = 33 CL16
x>>1 // 16 $\left[33/2^1\right] = 16$
x>>2 // 8 $\left[33/2^2\right] = 8$
x>>4 // 2 $\left[33/2^4\right] = 2$

Bitwise Not ~

$x = 1$

x : 00 01

$\sim x$: 11 10

$x = 5$

x : 00 0101

$\sim x$: 11 1010

eg:

```
unsigned int x = 1;
cout << (~x)
  // 4294967294
x = 5
cout << (~x)
  // 4294967290
```

unsigned

$2^{32} - 1$

4294967296

4294967295

Signed Input $(-2^{31}$ to $2^{31}-1)$

last bit → Number type

0 → positive

1 → negative

$x = 1$

x : 00 01

$\sim x$: 11 ..... 10 $(2^{32}-1-1)$

$2^{32}-2$

$x = 5$

x : 00 0101

$\sim x$: 11 ..... 1010 $(2^{32}-1-5)$

2^s complement of x in n bit representation

eg int x = 1

cout << (~x) // -2

x = 5

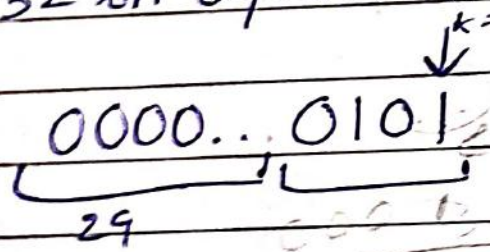
cout << (~x) // -6

Check if k-th bit is set?

1. I/P : n = 5, K = 1

Sol

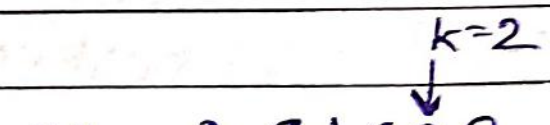
32 bit representation of 5



O/P = Yes

2. I/P : n = 8, K = 2

Sol O/P : NO



3. I/P : n = 0, K = 3

O/P : NO

K ≤ No. of bits in binary representation

Method 1 (left shift)

```
Void Kth Bit (int n, int K) {
    if (n & (1 << (K-1)) != 0)
        print ("Yes");
    else
        print ("No");
}
```

eg. I/P : n = 5, K = 3
O/P = Yes

AND operation

n = 000....0101
1<<(K-1) = 000....0100
000....0100

Method 2 (Righ shift)

```
Void Kth Bit (int n; int K) {
    if ((n >> (K-1)) & 1 == 1)
        print ("Yes");
    else
        print ("No");
}
```

I/P : n = 13, K = 3
O/P : Yes

AND

n = 1101
(n >> (K-1)) = 0011
1 = 0001
0001

Count Set Bits

1. n = 5

Binary representation : 101

o/p = 2

2. I/p = n = 7

Binary representation : 0111

o/p = 3

3 I/p : n = 13

Binary representation : 1101

o/p = 3

Method 1

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

OR

```
if((n&1) == 1)
   cs++;
n = n >> 1;
```

OR

```
cs = cs + (n&1);
n = n>>1
```

Method 2

Brian Kernigam's algorithm (Set Bit Count)

$n = 40$

Binary Representation:

initial : 00....0101000

After Ist Iteration: 00...0100000

After IInd Iteration: 00...0000000

```
int CountBits(int n)
{
    int ans = 0;
    While (n > 0)
    {
        n = (n & (n-1));
        ans++;
    }
    return ans;
}
```

101000	n = 40
100111	n-1 = 39
100000	n = 32

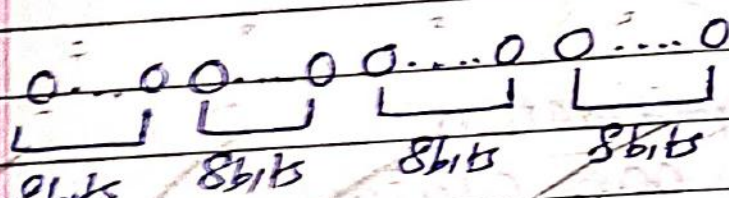
100000	n = 32
011111	n-1 = 31
000000	

Method 3

Lookup Table Method for 32 bits No.

No. represent in 8 bit = 0 .... 2^8-1

n = 13



i = count of set bit

```
int table[256];

void initialize()
{
    table[0] = 0;
    for (int i=1; i<256; i++)
        table[i] = (i&1) + table[i/2];
}
int count(int n)
{
    int res = table[n & 0x ff];
    n = n >> 8;
    res = res + table[n & 0xff];
    n = n >> 8
    res = res + table[n & 0xff];
    n = n >> 8
    res = res + table[n & 0xff];
    return res;
}
```

Complexity

→ O(1).

Power of two

1. I/p : n = 4
   o/p : Yes

2. I/p : n = 6
   o/p : No

Method 1 (Naive)

```
bool isPow2(int n)
{ if (n==0)
      return false
  while (n!=1)
    { if (n%2!=0)
        return false
      n = n/2;
    }
  return true;
}
```

n = 32 bit number

0Xff = 00...0 11111111 (24 bit | 8 bit)

n & 0Xff

↳ last 8 bit (getting right most part)

Method 2

Brian Kernigam's algorithm

the power of 2 element have set count alway equal to 1.

eg: 0000...0100 → 4
0000...1000 → 8

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

if (countBit(n) == 1)
    Print true;
else
    Print false;
```

Method 3

```
bool IsPow2(int n)
{
    if (n==0)
        return false;
    return ((n & (n-1)) == 0);
}
```

example 1

$n = 4$: 00...100

$(n-1) = 3$: 00...011

00...000

example 2

$n = 6$: 00...110

$(n-1) = 5$: 00...101

00...100

Find the only odd occurring number

1. I/p : arr[] = [4, 3, 4, 4, 4, 5, 5]
   o/p : 3

   I/p : arr[] = [8, 7, 7, 8, 8]
   o/p : 8

Method 1

Navive Solution

```
for (int i = 0; i < n; i++)
    {
      int count = 0;
    for (int j = 0; j < n; j++)
            if (arr[j] == arr[i])
                count++;
            if (count % 2 != 0)
                print (arr[i]);

    }
```

number

Method 2

Bitwise XOR ^

$x \wedge 0 = x$

$x \wedge y = y \wedge x$

$x \wedge (y \wedge z) = (x \wedge y) \wedge z$

$x \wedge x = 0$

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

Q Variation Question : given an array of n number that has value in range [1.... n+1]. Every no. appear exactly once. Hence one no. is missing. find the missing no.

I/P : arr[] = {1, 4, 3}

O/P : 2

(1 ^ 2 ^ 3 ... ^ (n+1)) ^ arr[0] ^ arr[1] ... arr[n-1]

I/P : arr[] = [1, 5, 3, 2]

O/P : 4.

Sol: Bitwise OR of given no [array] & bitwise of range created.

Find two odd appearing number.

I/P: arr[] = [3, 4, 3, 4, 5, 4, 4, 6, 7, 7]
O/P: 5, 6

I/P: arr[] = [20, 15, 20, 16]
O/P: 15

Method 1 Naive solution O(n²)

```
void oddAppearing(int arr[], int n)
{
    for(int i=0; i<n; i++)
    {
        int cnt=0;
        for(int j=0; j<n; j++)
            if(arr[i] == arr[j])
                cnt++;
        if(cnt%2 != 0)
            print(arr[i]);
    }
```

Method 2 (O(n))

Discussion: XOR of all no.

XOR = 3 ^ 4 ^ 3 ..... ^ 7

= 5 ^ 6

= 3

5 : 1 0 1

6 : 1 1 0

0 1 1

← Take this set bit and divide the array into two group such that group1 have ~~set bit and group2~~ didn't have.

group 1 = [3, 3, 5, 7, 7] = 5

group 2 = [4, 4, 4, 4, 6] = 6

Code ON NEXT PAGE

```
void addAppearing (int arr[], int n) {
    int XOR = 0, res1 = 0, res2 = 0;
    for (int i = 0; i < n; i++)
    {
        XOR = XOR ^ arr[i];
    }

    int Sn = XOR & ~(XOR - 1);   // Right most

    for (int i = 0; i < n; i++)
    {
        if ((arr[i] & Sn) != 0)
            res1 = res1 ^ arr[i];
        else
            res2 = res2 ^ arr[i];
    }
    printf (res1, res2);
}
```

Explanation

I/P: arr[] = [3,4,3,4,5,4,4,6,7,7]

O/P: 5 6

XOR = 3 ^ 4 ^ ..... ^ 7
XOR = 5 ^ 6
= 3

XOR = 3 : 0 0 ... 0 1 1
(XOR-1) (3-1) : 0 0 ... 0 1 0
~(XOR-1) : 1 1 ... 1 0 1
3 & ~(3-1) = (00...011) & (11...101)
= (00...001)

xcs1

group 1: [3, 3, (5), 7, 7]
group 2: [4, 4, 4, 4, (6)]

xcs2

I/P: arr[] = [3,4,3,4,8,4,4,32,7,7]

O/P: 8, 32

XOR = 3 ^ 4 ^ 3 ..... ^ 7
XOR = 8 ^ 32
= 40

~~XOR~~ ~~8~~ : 0 ... 0001000
~~XOR~~ 32 : 0 ... 0100000
XOR : 40 0 ... 0101000
(XOR-1) 0 ... 0100111
~(XOR-1) 1 ... 1011000
XOR & ~(XOR-1),
00 ... 01000

Power Set using Bitwise Operator

I/P : s = "abc"
O/P : "", "a", "b", "c", "ab", "ac", "bc", "abc"

I/P : s = "ab"
O/P : "", "a", "b", "ab"

Counter (Decimal)	Counter (Binary)	Subset
0	000	""
1	001	"a"
2	010	"b"
3	011	"ab"
4	100	"c"
5	101	"ac"
6	110	"bc"
7	111	"abc"

value varies from 0 to $2^n - 1$

I/P : s = "ab"
O/P : "", "a", "b", "ab"

Counter (Decimal)	Counter (Binary)	Subset
0	00	""
1	01	"a"
2	10	"b"
3	11	"ab"

value varies from 0 to $2^n - 1$

```
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[j]);
    }
    print("\n");
}
}
```

counter = 3

j = 0 to j = 2

j = 0

011 & (001<<0)

011 & 001

001

print → a

j = 1 011 & (010)

010

Print - b

j = 2

011 & (100)

000

ch: 4 Recursion

Write a recursive function to check if a string palindrome.

I/P: str = "aabaa"
O/P: Yes
I/P: str = "geeks"
O/P: No

// Initially s=0, e=n-1

```
bool isPal(String str, int s, int e)
{
    if (s==e) return true;
    if (s>e) return true;
    if (str[s] != str[e])
        return false;
    return isPal(str, s+1, e-1);
}
```

Writing Base Cases in Recursion.

```
int fact(int n)
{
  if (n==0)
    return 1;
  return n * fact(n-1);
}
```

nth Fibonacci Number where n >= 0

0, 1, 1, 2, 3, 5, 8, 13.....

I/P : n = 3
O/P : 2

```
int fib(int n) {
    if (n==0) return 0
    if (n==1) return 1;
  return fib(n-1) + fib(n-2);
}
```

Write a recursive function to find sum of digits in a number.

1. I/P : n = 253
   O/P : 10
   2+5+3

2. I/P : n = 9987
   O/P : 33
   9+9+8+7

```
int fun (int n)
    if (n<10)
        return n;
    return fun (n/10) + n%10;
```

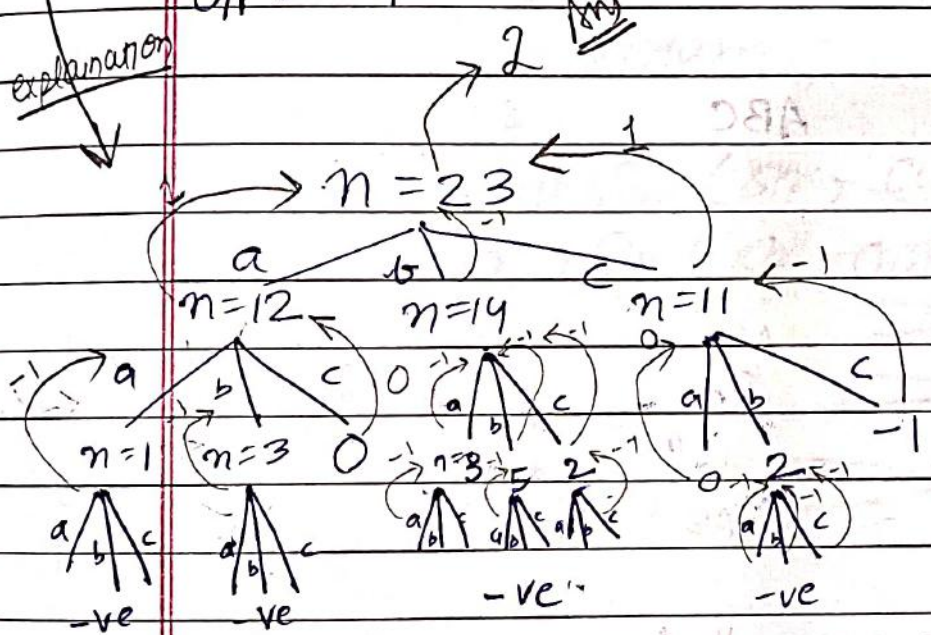
Given a rope of length n, you need to find maximum number of pieces you can make such that length of every piece is in set {a, b, c} for given three value a, b & c

1. I/P : n = 5
   a = 2 , b = 5 , c = 1
   O/P : 5

2. I/P : n = 23
a = 12, b = 9, c = 11
O/P : 2

3. I/P : n = 5
a = 4, b = 2, c = 6
O/P : -1

explanation




```
int maxCuts (int n, int a, int b, int c)
{
    if (n == 0) return 0;
    if (n < 0) return -1;

    int res = max ( maxCuts(n-a, a, b, c),
                    maxCuts(n-b, a, b, c),
                    maxCuts(n-c, a, b, c) );

    if (res == -1) return -1;
    return (res + 1);
}
```

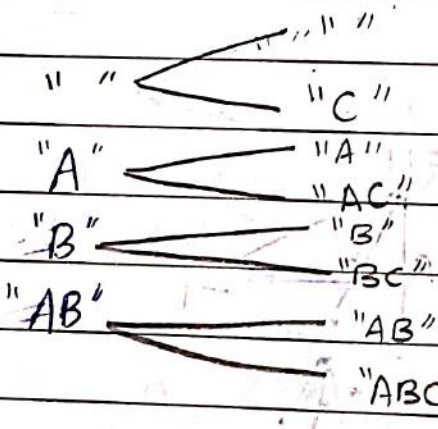
Complexity. → $O(3^n)$

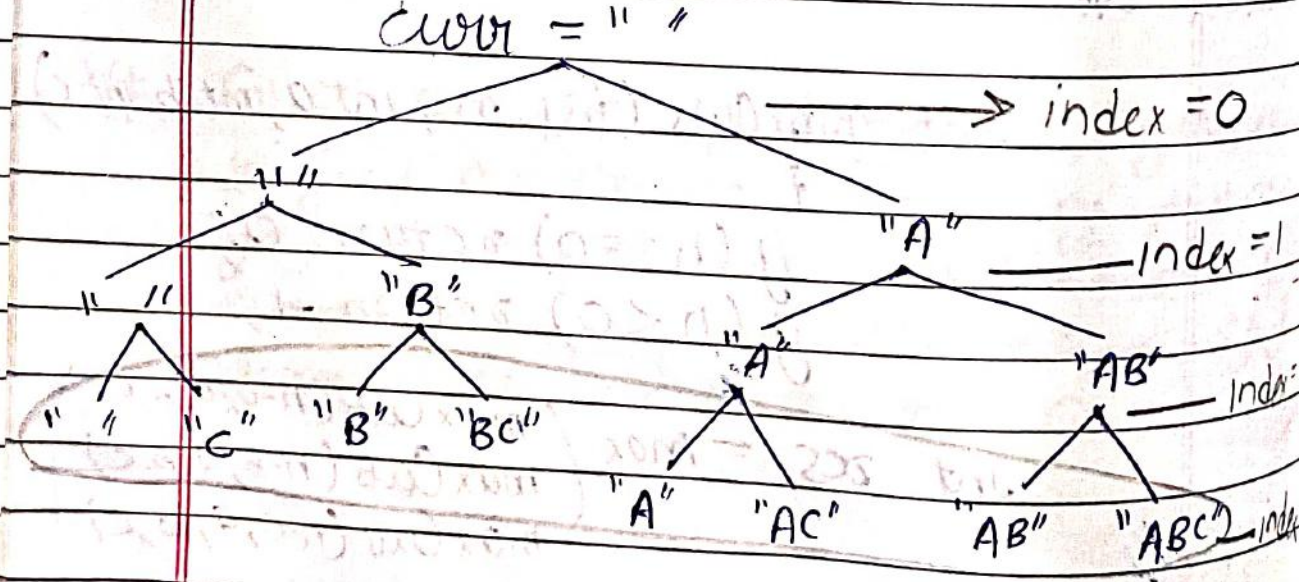
Given a string print all subsets of it (in any order)

I/P : str = "ABC"

O/P : "", "A", "B", "C", "AB", "AC", "BC", "ABC"

If I have solution for AB



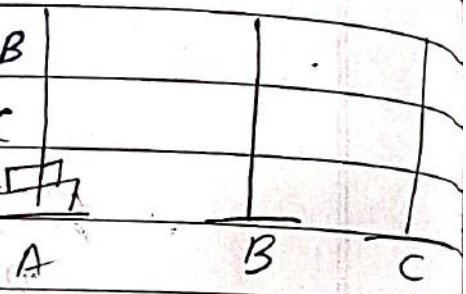


```
void printSub (String str, String curr="",
                int index=0)
{
    if (index == str.length())
    {
        count << curr << " "
        return;
    }
    printSub (str, curr, index+1)
    printSub (str, curr+str[index], index+1);
}
```

Tower of Hanoi

1. I/p : n = 1
   o/p : Move Disc 1 from A to C

2. I/p : n = 2
   o/p : move Disc 1 from A to B
   move Disc 2 from A to C
   move Disc 1 from B to C



Rules:

1. Only one Disc moves at a time
2. No larger Disc above smaller
3. Only the top Disc of a tower can be moved.

Problem 1

I/p n = 1

o/p move Disc from A to C

Problem 2

I/p : n = 2

o/p : move Disc 1 from A to B
move Disc 2 from A to C
move Disc 1 from B to C

Problem 3. I/P : n = 3

move disc 1 from A to C
move disc 2 from A to B
move disc 1 from C to B
move disc 3 from A to C
move disc 1 from B to A
move disc 2 from B to C
move disc 1 from A to C

source Auxillary destination

```
TOH (n, A, B, C)
{
    TOH((n-1), A, C, B)
    move Disc n from A to C
    TOH((n-1), B, A, C)
}
```

** No. of movement for given n

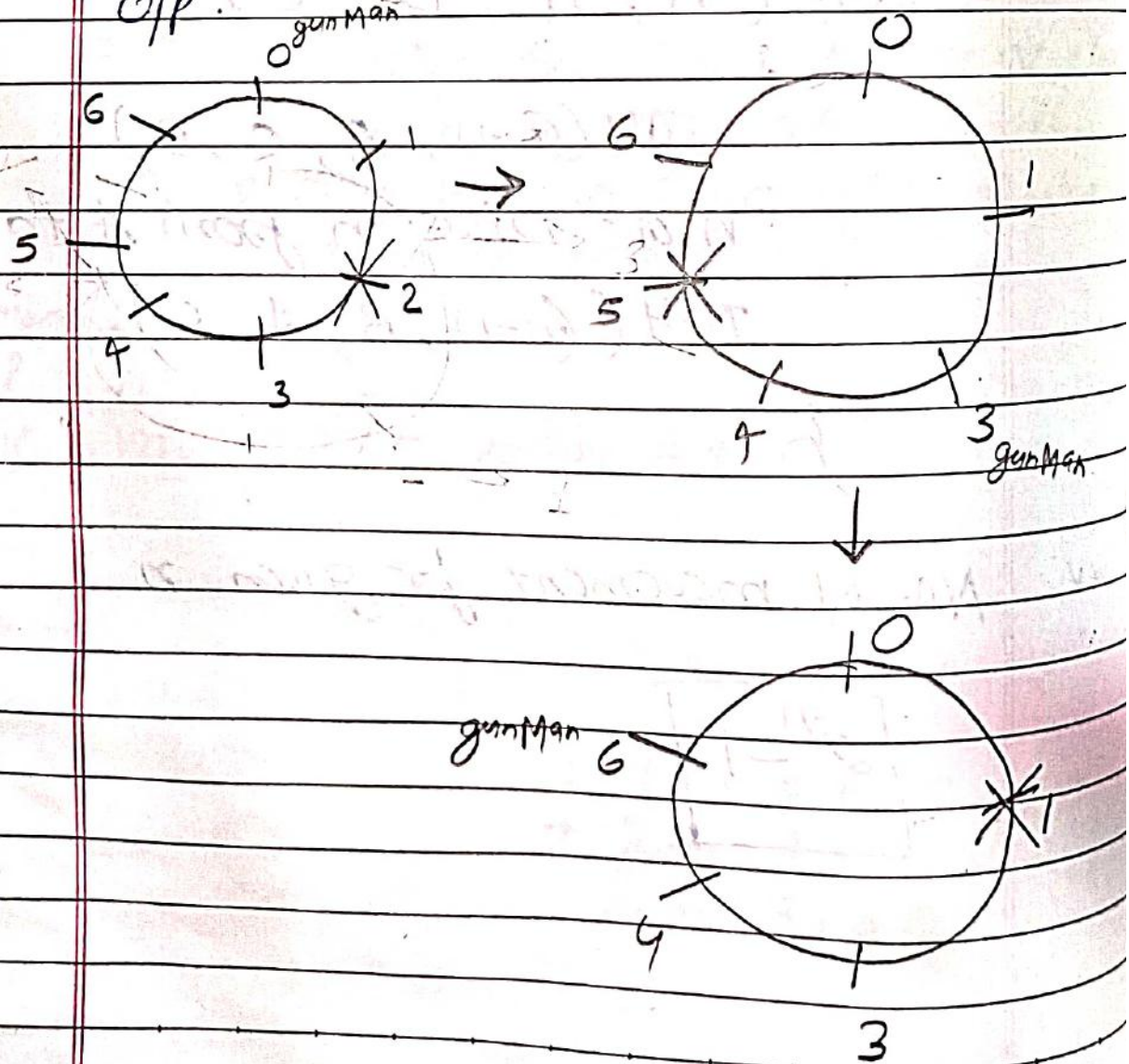
$$2^n - 1$$

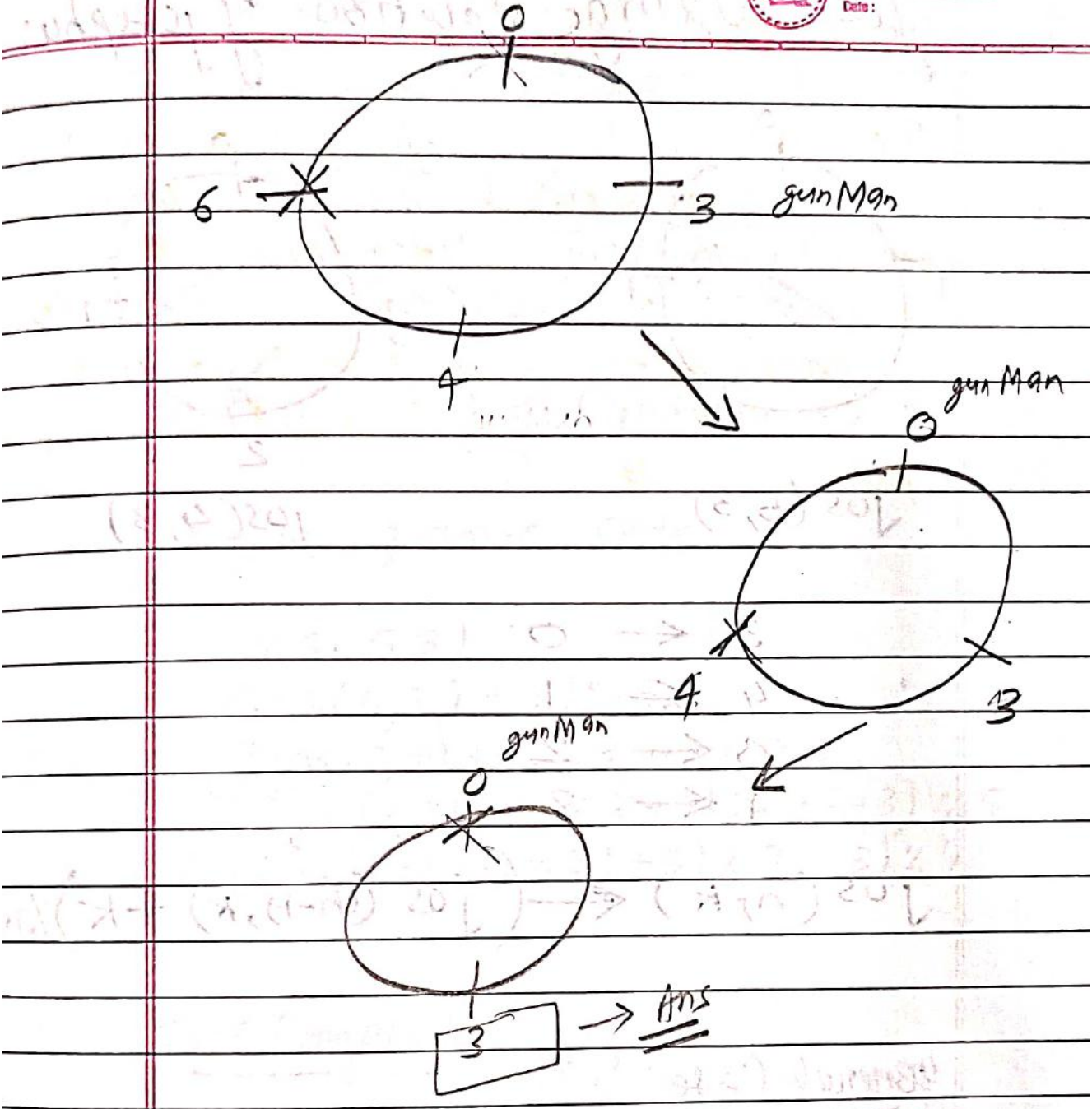
Josephus Problem

There are n people standing in a circle. we need to kill kth person in every iteration and this has to be done in circular manner. After repetately doing this we have to find the position of Survival.

1. I/P : n = 7 // 7 people in circle
   k = 3 // kill the 3 person.

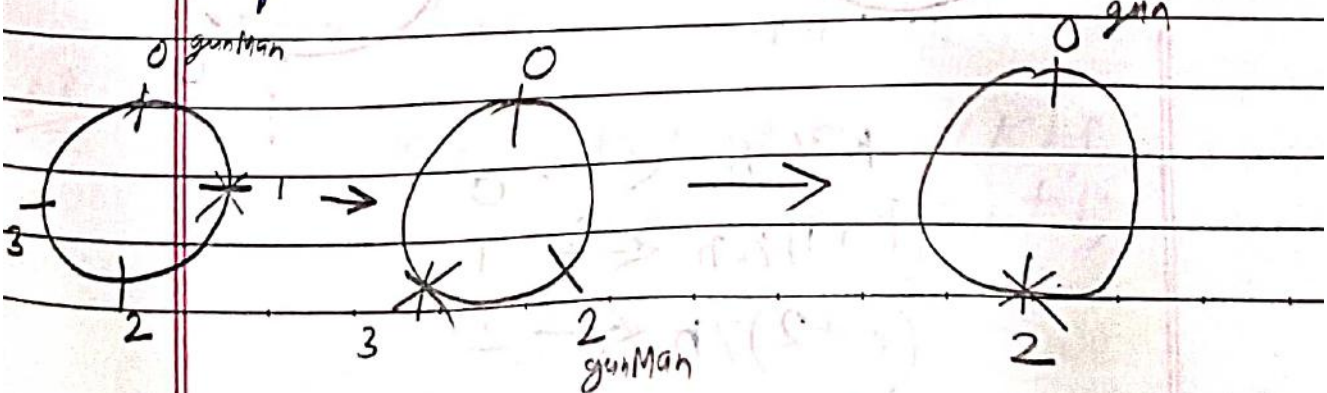
O/P :




0
6
3
gunMan
4
gunMan
0
4
3
gunMan
0
3
Ans

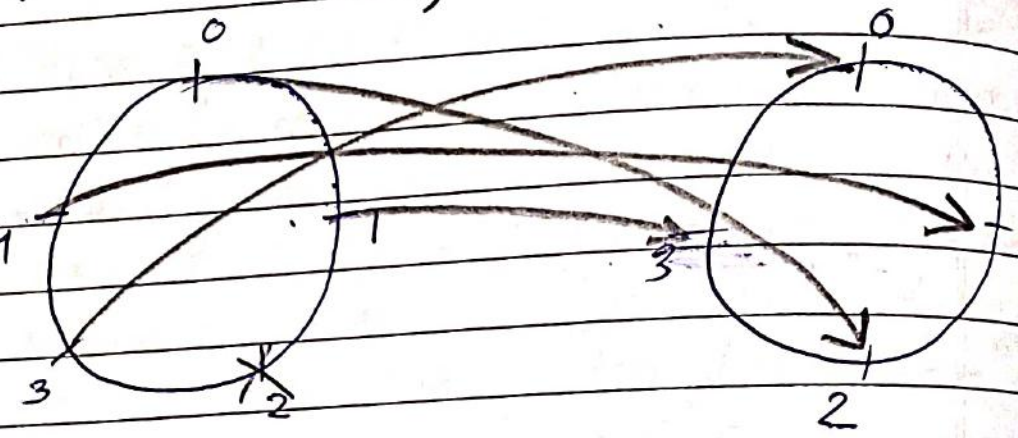
2. I/P : n=4, k=2

o/p : 0


0
gunMan
3
2
0
3
2
gunMan
0
gun
2

Generalizing Solution of Josephus

$n = 5, K = 3$



jos(5,3) jos(4,3)

$3 \leftarrow 0$

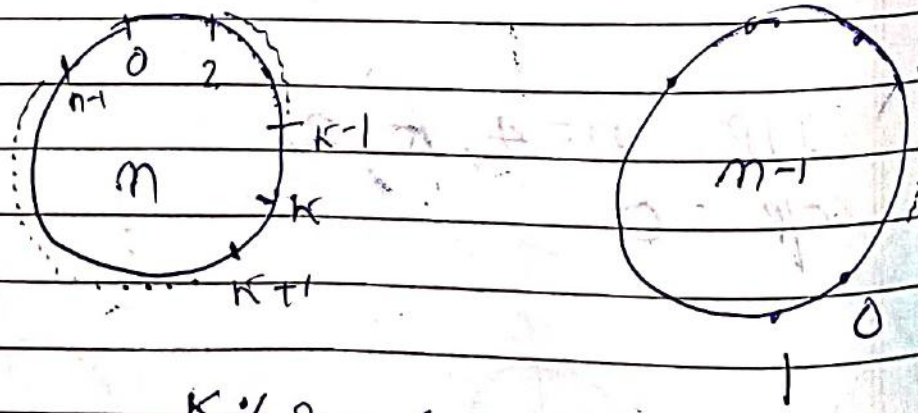
$4 \leftarrow 1$

$0 \leftarrow 2$

$1 \leftarrow 3$

$jos(n,K) \leftarrow (jos((n-1),K) + K)\%n$

general Case



$K\%n \leftarrow 0$

$(K+1)\%n \leftarrow 1$

$(K+2)\%n \leftarrow 2$

$(K+i)\%n \leftarrow i$

```
int jos (int n, int K)
{
    if (n==1)
        return 0
    else
        return (jos(n-1, K)+K) %n
```

For run of above code

jos(5,3)
(jos(4,3)+3)%5
((jos(3,3)+3)%4+3)%5
(((jos(2,3)+3)%3+3)%4+3)%5
((((jos(1,3)+3)%2)+3)%3+3)%4

Time Complexity $T(n) = T(n-1) + C$
$= O(n)$

* Extension to problem: If it is not start with index = 0. It will start with index 1.

Sol: Code

```
int myJobs (int n, int K){
    return jos(n,K)+1;
}
```