A

Bitwise Operators

Check kth bit is set

Count set bits

Power of two

Find only odd occuring numbers

Find two odd apearing numbers

Power set using bitwise Operators

Find the two non-repeating elements in an array of repeating elements

Count number of bits to be flipped to convert A to B

Count total set bits in all numbers from 1 to n

Calculate square of a number without using *, / and pow()

Divide two integers without using multiplication, division and mod operator

Find position of the only set bit

set the ith bit

Clear the ith bit

Toggle the ith bit

A	B
Bitwise Operators	
Check kth bit is set	
Count set bits	
Power of two	
Find only odd occuring numbers	
Find two odd apearing numbers	
Power set using bitwise Operators	
Find the two non-repeating elements in an array of repeating elements	
Count number of bits to be flipped to convert A to B	
Count total set bits in all numbers from 1 to n	
Calculate square of a number without using *, / and pow()	
Divide two integers without using multiplication, division and mod operator	
Find position of the only set bit	
set the ith bit	
Clear the ith bit	
Toggle the ith bit	

Naive Solution

```
void isSet(int n, int k)
{
    int x = 1;
    for(int i=0; i<(k-1); i++)
        x = x*2;
    if((n & x) != 0)
        print("yes")
    else
        print("No")
}
```

n = 5 (0...0101)

k = 3

After loop:

x = 4 (0...0100)

```
  0...0101
& 0...0100
-----------
  0...0100
```

↓

Representation

Naive Solution

```
void isSet(int n, int k)
{
    int x=1;
    for(int i=0; i<(k-1); i++)
        x=x*2;
    if((n&x)!=0)
        print("yes")
    else
        print("No")
}
```

n=5 (0...0101)

k=3

After loop:

x=4 (0...0100)

```
  0...0101
& 0...0100
-----------
  0...0100
```

↓

Representation of 4..

Alternate

Naive Soln

We Reduce

n to $\lfloor n/(2^{k-1}) \rfloor$

Time : $\theta(k)$

```
void isSet(int n, int k)
{
  for(int i=0; i<(k-1); i++)
      n=n/2;
  if((n&1)!=0)
      print("yes")
  else
      print("No")
}
```

n=5 (0...0101)

k=3

After loop

n=1 (0...001)

Efficient Method I

```
void isKthSet(int n, int k)
{
    int x = (1<<(k-1)); // 2^(k-1)
    if ((n&x)!=0)
        printf("yes");
    else
        printf("No");
}
```

n=5 (00...0101)
k=3
x=4 (00...0100)

00...0101
& 00...0100
00...0100

x=1

Efficient Method II

(int n, int K)

Efficient method II

```
void isKthSet(int n, int k)
{
  int x = (n >> (k-1)); // ⌊n/(2^(k-1))⌋
  if ((x & 1) != 0)
      printf("yes")
  else
      printf("No")
}
```

00 .. 0100

n = 5 (00 .. 0101)

k = 3

x = 1 (00 .. 0001)

```
   00.... 0001   x
&  00... 0001    1
-----------------
   00 .... 0001
-----------------
```

Count Set Bits

I/P: n=5

O/P: 2

n=5

101

I/P: n=7

O/P: 3

n=7

111

I/P: n=13

O/P: 3

n=13

1101

Binary Representation.

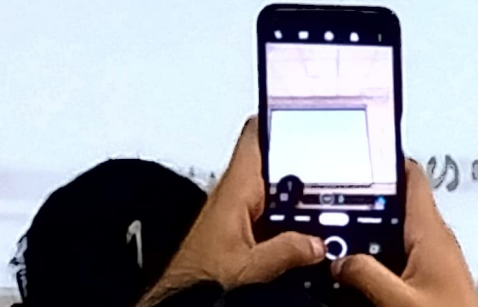
5/2 ⇒ 1

2/2 ⇒ 0

1/2 ⇒ 1

Count ⇒ 0

Naive Solution



```
int countSetBits(int n)
{
    int res = 0;
    while(n>0)
    {
        If(n%2==1)
            res++;
        n = n/2;
    }
    return res;
}
```

n = 5 (00...0101)

res = 0

1st Iteration -

res = 1, n = 2 (00...010)

IInd Iteration -

res = 1, n = 1 (00...001)

IIIrd Iteration -

res = 2, n = 0 (00...000)

Time = $\theta(d)$

```
int countSetBits(int n)
{
    int res=0;
    while(n>0)
    {
        if(n%2==1)          → res = res+(n&1);
            res++;
        n=n/2;   ⇒ n = n>>1
    }
    return res;
}
```

n = 101

Brian Kernigam's Algorithm

Idea:

Traverse through only the set bits.

Time: θ(set bits)

```
int countSetBits(int n)
{
  int res=0;
  while(n>0)
  {
    n= [ ? ]
    res=res+1;
  }
  return res;
}
```

→ This expression should make the last set bit as 0.

n = 10 - 1010 → A

(n-1) = 9 - &1001

1000 → B

A & B = 1000

0111

A & B & 0111 = 0000

res = 2

n = 40 (101000)

After I st Iteration

n = 32 (100000)

After II nd Iteration

0 (000000)

Time: O(set bits)

```
}
return ans;
```

After 1st Iteration

After 2nd Iteration

```
int countSetBits(int n)
{
    int ans = 0;
    while (n != 0)
    {
        n = n & (n-1);
        ans++;
    }
```

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

n = 40 : 101000

(n-1) = 39 : 100111

n & (n-1) = 32 : 100000

n = 32 : 100000

n-1 = 31 : 011111

n & (n-1) = 0 : 000000

Power of Two

Ip : n=1
op : True

Ip : n=6
op : False

Ip : n=16
op : True

n >= 0

9

12

16

12 → 6

odd

Naive Solution

1) Explicitly

a) Repeating ... 0

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

n=6

I^st Iteration : n=3

II^nd Iteration : return false

n=8

I^st Iteration : n=4

II^nd Iteration : n=2

III^rd Iteration : n=1

return true

16 9

8) 16/2 → 0 64 → 32

8/2 → 0 16 → 1

4/2 → 0

1) 1/2 → 0

4: 00...0100

n=4: 00..0100
&(n-1)=3: 00..0010

00..0000

n=6: 00..0110
&(n-1)=5: 00..0101

00..0100

```
bool isPow2(int n)
{
    if(n==0)
        return 0;
    return ((n&(n-1))==0);
}
```

return (n&((n&(n-1))==0))

Find the only Odd Occurring Number

I/p: arr[] = {4, 3, 4, 4, 4, 5, 5}

O/p: 3

I/p: arr[] = {8, 7, 7, 8, 8}

O/p: 8

Naive Solution

int findOdd(int arr[], int n)

1 0 0
0 1 1
---
1 1 1
1 0 0
---
0 1 1
1 0 0
---
1 1 1
1 0 0
---
0 1 1
1 0 1
---
1 1 0
1 0 1
---
0 1 1 → 3

Naive Solution

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

```
int findOdd(int arr[], int n)
{
    int res = arr[0];
    for(int i=1; i<n; i++)
        res = res ^ arr[i];
    return res;
}
```

arr[] = {4,4,7,4,8,7,7,8}

4^4^7^4^8^7^7^8

= (4^4^4)^(7^7^7^7)^(8^8)

= 4^0^0

= 4^0

= 4

Time: $\Theta(n)$

Aux space: $O(1)$

1) XOR of any num... n ^ n = 0

2) XOR of any nu... ^ 0 = n

Find Two Odd Appearing Numbers

I/p: arr[] = {3, 4, 3, 4, 5, 4, 4, 6, 7, 7}

O/p: 5 6

I/p: arr[] = {1, 3, 2, 3, 3, 1}

O/p: 2 3

Naive Solution

Traverse through the array, count Occurances of every number. If count is odd, print the

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

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

```
Void oddAppearing(int arr[], int n)
{
    int x = arr[0];
    for(int i=1; i<n; i++)
        x = x ^ arr[i];
    int k = (x & (~(x-1)));
    int res1=0, res2=0;
    for(int i=0; i<n; i++)
    {
        if((arr[i] & k)!=0)
            res1 = res1 ^ arr[i];
        else
            res2 = res2 ^ arr[i];
    }
    print(res1, res2);
}
```

//right set bit

x=3 : 00 0011

x-1=2 : 00 0010

~(x-1) : 11 1101

x & ~(x-1) : 00 0001

arr = {1, 6, 5, 6, 6, 1}

x = 1^6^5^6^6^1 = 3

x = 1

res1 = 1^5^1 = 5

res2 = 6^6^6 = 6

10 → 1010

9 → 1001

0110

5 → 0101

6 → 0110

10 → 1010

6 → 0110

10 → 1010

6 → 0110

3 → 0011

3 → 0011