Bits

Binary Form of -18 can be written as 1's complement to 18 + 1

2's complement of 18 gives us binary form of -18

Binary Form of 18 — 0 0 0 1 0 0 1 0

1's complement of 18 — 1 1 1 0 1 1 0 1 ⇒ for 2's complement

+ 1

1 1 1 0 1 1 1 0

128	64	32	16	8	4	2	0
0	0	0	1	0	0	1	0
-2^7	2^6	2^5	2^4	2^3	2^2	2^1	0
1	1	1	0	1	1	1	0 ⇒ -18

-18 can be obtained from its binary form as

$-2^7 + 2^6 + 2^5 + 2^3 + 2^2 + 0\ 2^2$

$= -128 + 64 + 32 + 8 + 4 + 0\ 2$

$= -64 + 32 + 8 + 4 = -32 + 12 + 2 =$

MSB notation is nothing but simply changing most significant bit to 1 for signed number

0 0 0 0 0 1 0 1 (+5)

1 0 0 0 0 1 0 1 (-5) ⇒ MSB notation

* MSB notation doesn't hold true because the value after the MSB notation is completely diff from original binary value & addition / subtraction gives wrong result.

Primitive Data Types

int 4B

float 4B

double 8B

char 1B & (2B in Java)

bool 1B

Byte (Java)

C++/C (4B/8B)

Short 2B

long → used in C++

long long (8B) (compiler dependent)

Java waala (8B)

2^3 2^2 2^1 2^0 (unsigned)

8 4 2 1 (max, min) 15, 0

(signed) -8 4 2 1 (7, -8)

# bits	signed	Unsigned
1	-1, 0	1, 0
2	-2, 1	3, 0
4	-8, 7	0, 15
8	-128, 127	255, 0
n	-2^{n-1}, $2^{n+1}-1$	$0, 2^n-1$

$2^{10} = 1024 \simeq 10^3$

	(bits)	(signed)	(unsigned)
integer (int)	32 bits	-2^{31}, $2^{31}-1$	$0, 2^{32}-1$
		$-2 \cdot 10^9$, $2 \cdot 10^9 - 1$	$0, 4 \cdot 10^9 - 1$
long long	64 bits	-2^{63}, $2^{63}-1$	$0, 2^{64}-1$
		$-8 \cdot 10^{18}$, $8 \cdot 10^{18}-1$	$0, 16 \cdot 10^{18}-1$

For summation of array elements

$1 \le N \le 10^5$

$-10^6 \le ar[i] \le 10^6$

range is $-10^{11} \le ans \le 10^{11}$ (Long should be used)

Operators

Arithmetic ⇒ % + - / *

Logical ⇒ &&, ||, !

Relational ⇒ <, <=, >, >=, ==, !=

Bitwise ⇒ &, |, ^, ~, <<, >>

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

Bitwise operators or both commutative & Associative

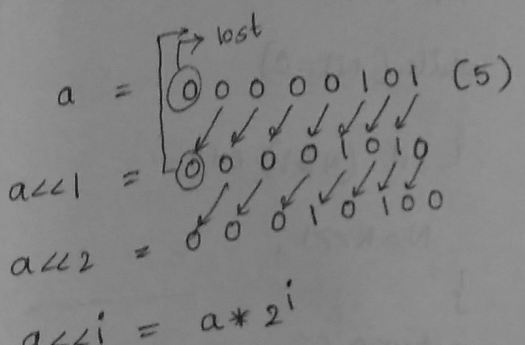
$a|b = b|a$ $\quad a \wedge b = b \wedge a$ $\quad a\&b = b\&a$

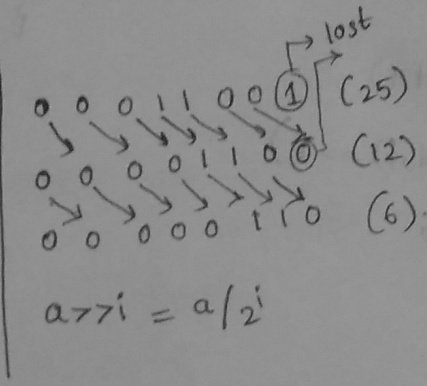
$a|b|c = c|a|b$ $\quad b|c|a = c|b|a$

a b
38, 21

	-128	64	32	16	8	4	2	1
a =	0	0	1	0	0	1	1	0
b =	0	0	0	1	0	1	0	1
a|b =	0	0	1	1	0	1	1	1
a&b =	0	0	0	0	0	1	0	0
a^b =	0	0	1	1	0	0	1	1
~a =	1	1	0	1	1	0	0	1

Right shift & Leftshift






```
pow2(int N)
{
    return 1<<N;
}
```

N is long long
then return 1ll<<N;
1L<<N;
1LL<<N;

A	B	C
a|0 = a	a|a = a	a|1 = odd → a, even → a+1
a&0 = 0	a&a = a	a&1 = odd → 1, even → 0
a^0 = a	a^a = 0	a^1 = odd → a-1, even → a+1

int a = MSB - - - - - - - - LSB

	1/0	LSB
&	0 0 0 0 0 0 0 0	1
	0 0 0 0 0 0 0 0	1/0

bool checkBit (int N, int i) { $0 \le N \le 10^9$, $0 \le i \le 30$

I/P		O/P
5,1	$(101)_{2}$ (2nd, 1st, 0th)	F
5,2		T
9,3	$(1001)_{2}$ (3, 2, 1, 0)	T

```
bool checkBit (int N, int i)
{
    return ((N>>i)&1)==1;
    return ((N>>i)%2)==1;
    return ((N&(1<<i))!=0);
}
```

→ prefer this

% costly int

↳ +ve only

```
int countbits (int N) { 0 ≤ N ≤ 10^9 }
{
    int c=0;
    for (int i=0; i<32; i++)
        if (checkBit (N,i)==T)
            c++;
    return c;
}
```

```
int countBits (int N)
{
    int c=0;
    while (N!=0)
    { if (N&1) c++;
      N=N>>1;
    }
    return c;
}
```

N		N-1		N&(N-1)	
10010	18	10001	17	10000	16
11011	27	11010	26	11010	26
100100	36	10011	35	100000	32
10001	17	10000	16	10000	16
10000	16	01111	15	00000	0

Least significant set Bit becomes zero

Whenever we consider & operation between N & (N-1) then Least significant set bit of N becomes '0'

Complexity Analysis of Algorithm

```
int divisors(int N)
{
    int c=0;                    // I   ✓ only once
    for (int i=1; i<=N; i++)    // II  1 ↓   2 ↓
        if (N%i==0)             //     ↓9
            c++;                // III 3 ↑   5 ↑
    return c;
}
```

Does this program work for 10^9 ~~iterations~~

$N = 10^9$

1 GHz = 10^9 instructions / sec

$N = 10^{18}$

iterations = 10^9

instructions / iteration = ~~5~~ 5

instructions = 5×10^9

Time (1 GHz) = 5 sec

10^{18} approx

10^9 sec (31.1 years)

10^9 inst → 1 sec

10^{18} inst → 10^9 sec

N = 20

1	20
2	50
4	5
5	4
10	2
20	1

N = 100

1	100
2	50
4	25
10	10

i ←

Observe here

$i^2 \le N \Rightarrow i \le \sqrt{N}$

we can find all factors

```
for (int i=0; i<=N; i=i*2)
```

$\equiv O(1)$

i	total
1	1
2	1
$2^2=4$	1
$2^3=8$	1
$2^4=16$	1
2^K	K+1

$O(\log_2 N)$

$\log_2 N + 1$

i goes upto N

$2^K = N$

$K = \log_2 N$

```
int a=0;
for(int i=0; i<N; i++)
{
  for(int j=N; j>i; j--)
  {
    a=a+i+j;
  }
}
```

i	j	total
0	[N,0)	N
1	[N,1)	+ N-1
2	[N,2)	+ N-2
⋮		+ ⋮
N-1	[N,N-1)	+ 1

$\frac{N(N+1)}{2}$

$= \frac{N^2}{2} + N/2$

$O(N^2)$

```
int a=0, i=N;
while(i>0)
{
  a+=i;
  i/=2;
}
```

	i	total
	N	1
$N/2^1$	N/2	+ 1
$N/2^2$	N/4	+ 1
$N/2^3$	N/8	+ 1
	⋮	
$N/2^k$	k+1	
	⋮	
	1	(k+1)

4

$\frac{N}{2^k} = 1$

$\Rightarrow 2^k = N$

$k = \log_2 N$

$\Rightarrow \log_2 N + 1 = O(\log_2 N)$

```
int count=0;
for(int i=N; i>0; i/=2)
{
  for(int j=0; j<i; j++)
  {
    count+=1;
  }
}
```

$O(N)$

i	j	total
N	[0,N]	N
N/2	[0,N/2]	N/2
N/4	[0,N/4]	N/4
	⋮	

GP $\Rightarrow$ a=1, r=1/2

$S_n = \frac{a(1-r^n)}{1-r}$

$= N\left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots\right)$

$= \frac{N \cdot 1 \cdot \left(1 - \frac{1}{2}^k\right)}{1 - 1/2} = 2N\left(1 - \frac{1}{2^k}\right)$

k=1 → 0.5
k=2 → 0.25
k=3 → 0.1

$\frac{1}{2^k} \to 0$

$= 2N$

```
int k=0;
for (int i= N/2 ; i<= N ;++i)          → N/2
    for (int j= 2; j<=N ; j=j*2)        → log2 N
        k= k+ N/2;
```

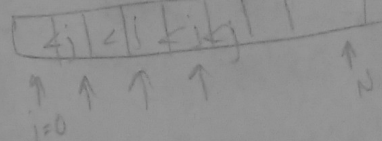
$\frac{N}{2} \log_2 N$

$O(N \log_2 N)$

```
int j=0;
for (int i=0; i<N ;++i) {
    while( j<N && arr[i] <= arr[j])
    {    j++;
    }
}
```

$N + N = 2N$ $O(N)$

→ This like initially j=0; it runs till j=N if the condition of arr[i] <= arr[j] for only once. In worst case overall it runs only once 'N' times. The second loop isn't dependent of i.



```
void func (int N, int K)
{   for (int i=1; i<=N ;i++) {
        /* Assume that pow() takes constant
                              amount time */

        int P= pow(i,K);

    for (int j=1 ; j<= P ;++j) {
            O(1)
    }
}
```

i	j	total
1	$[1, 1^k]$	1^k
2	$[1, 2^k]$	2^k
3	$[1, 3^k]$	3^k
⋮		
N	$[1, N^k]$	N^k

Total = $1^k + 2^k + 3^k + 4^k + \dots + N^k$

$k=1: 1+2+3+\dots+N = \frac{N(N+1)}{2} \rightarrow \frac{N^2}{2}$

$k=2: 1^2+2^2+3^2+\dots+N^2 = \frac{N(N+1)(2N+1)}{6} \rightarrow \frac{N^3}{3}$

$k=3: 1^3+2^3+3^3+\dots+N^3 = \frac{N^2(N+1)^2}{4} \rightarrow \frac{N^4}{4}$

$O\left(\frac{N^{k+1}}{k+1}\right)$

Suppose

sol A $10N^3 + 5N + 10 \rightarrow O(N^3)$

sol B $15N^2 + 10000 \rightarrow O(N^2)$

Sol A may is good for small inputs but for larger inputs sol B is preferred.

Big-O Notation

Puts an upper bound on the complexity of an algorithm based on the input size after certain thriesfold

Big-O:	N^3	N^2	$N\log_2 N$	N	$\sqrt{N}$	$\log_2 N$	1	2^N (N=30)	2^N (N=60)
#iterations:	10^{18}	10^{12}	$2 \cdot 10^7$	10^6	10^3	20	1	10^9	10^{18}
Time:	10^9 sec	10^3 sec	$2 \cdot 10^{-2}$ sec	.001s	10^{-6} sec	20ns	1ns	1 sec	10^{99} sec
	31.7 years	16.6 mins	0.02sec	1ms	1 μs	$20 \cdot 10^{-9}$	$1 \cdot 10^{-9}$		
				↓ millisec	microsec				

$N = 10^6$ 1 iteration = 1 instruction

1 GHz = 10^9 ins/sec $\log_2 10^3 = \log_2 2^{10} = 10$

$N^2 = (10^6)^2 = 10^{12}$

$10^9 - 1$

$10^{12} - x$

$\frac{10^{12}}{10^9} = x$

$N\log_2 N$

$10^6 \Big/ \log_2 10^6$

$\log_2 (10^3)^2$

$\log_2 (2^{10})^2$

$= 20$

$20 \cdot 10^6 - x$

$10^9 - 1$

$\frac{10^9}{20 \cdot 10^7} \; 10^{-2}$

2^{30}

$2^{10} \simeq 10^3$

$(10^3)^3$

10^9

For a program with Test cases

Read T ← Test case

loop (T)

Read I/P

Process

Print O/P

N / N^2 / $N\log_2 N$

Total Time complexity is $N \cdot T$, $N^2 \cdot T$, $N\log_2 N \cdot T$ $\quad M = (N, N^2, N\log_2 N)$

Here $\boxed{T \cdot M \leq 10^8}$

for TLE not exceeding

for suppose

Constraints
$1 \le T \le 100$
$1 \le N \le 10^6$
$O(N^2) - 10^2 \times 100 = 10^{14}$ iterations $= 10^5$ sec (TLE)

Constraints

⇒ Helps in determining data type

⇒ Helps us towards an approach to solution

$1 \le T \le 10^3$
$1 \le N \le 10^6$
$O(N)$ $N \cdot T = 10^9$ (TLE)
$O(\sqrt{N})$ $10^3 \cdot 10^3 = 10^6 \le 10^8$ ✓

$1 \le T \le 10$
$1 \le N \le 10^{18}$
~~N~~ ~~√N~~ fails $O(\log_2 N)$ or $O(1)$

SPACE COMPLEXITY

```
void f(int ar[], int N)
{
    int x,y,z;          12B
    double d1,d2;       16B
    int br[10];         40B
}
```
68B → This is O(1) Because it is a constant & doesn't depend on N

for
```
int cr[N]          O(N)
double d[N]        O(N)
int arr[N][N]      O(N^2)
```

$1 \le T \le 100$
$1 \le N \le 10^5$

10^9 B ≈ 1GB

```
int mat[N][N];   ⇒ 40GB   ← RTE
4B               10^5 10^5
4B · 10^6
int ar[10^6] = 4MB
```
360MB

```
bool ar[10^8] = 100 MB
1B · 10^8
```

bool → 6–7.8

$O(S) \le 10$

$O(T \times M) \le 10^8$

Find the duplicate

1Q:

	0	1	2	3	4	5	6	7	8
arr[N] :	3	5	8	2	8	5	(10)	2	3

O/P : 10

$a \wedge a = 0$

$O(N^2)$ ① int f(int ar[], int N)

```
{
  for(int i=0; i<N; i++)
      bool dup = false;
  {
    for(int j=0; j<N; j++)
    {
      if (ar[i]==ar[j] && i!=j)
          dup = true;
      break;
    }
    if (!dup) return ar[i];
  }
}
```

② int f(int ar[], int N)

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

① $O(N^2), O(1) \rightarrow N^2, 1$

② $O(N), O(1) \rightarrow N, 1$

2Q: Subset

arr_N : 3 10 5 -6 18 -2

For a given array N, is subset sum value of K is possible or not

K = 26, 25, 0, 50, 1

T T T F T

26 - [10, 18, -2] 10+18-2=26

0 - [3, 5, -6, -2] 3+5-6-2=0

1) Consider a simple array of 2 element [a, b]

No. of subsets a, b, ab, ba, ϕ

Simple array of 3 elements [a, b, c]

No. of subsets a, b, c, ab, bc, ca, abc, ϕ

So for array of N elements there will be 2^N subsets.

ar[0]=3 ar[1]=10 ar[2]=5

	2nd bit	1st bit	0th bit	
	4	2	1	ϕ
0	0	0	0	ϕ
1	0	0	1	{3}
2	0	1	0	{10}
3	0	1	1	{3,10}
4	1	0	0	{5}
5	1	0	1	{3,5}
6	1	1	0	{10,5}
7	1	1	1	{3,10,5}

$O(N \cdot 2^N), O(1)$

```
bool f (int ar[], int N, int K)
{
  for (int i=0; i< (1<<N); i++)
  {
    int s=0, j=0;
    while (i>0)
    {
      if (i&1 ==1)
        s+ = ar[j];
      j++;
      i= i>>1;
    }
    if (s== K) return T;
  }
  return F;
}
```

```
for (int j=0; j<N; j++)
  if (checkBit (i,j) ==T)
    s+ = ar[j];
```

Set has n elements, We will use the bits of numbers to show the presence of elements in set. The key is
- If ith bit in the number is SET i.e. 1, then ith element in given set is present in current subset. We will loop from 0 to $(2^n)-1$, and for each number, we will check the SET Bits in it & take corresponding elements

Modulo Arithmetic

1) $(a+b)\%M = (a\%M + b\%M)\%M$

$(a+b+c)\%M = ((a+b)+c)\%M = ((a\%M + b\%M)\%M + c\%M)\%M$

2) $(a-b)\%M = (a\%M - b\%M + M)\%M$

3) $(a/b)\%M = (a\%M * b^{-1}\%M)\%M$ Inverse Modulo

↳ Found by Extended Euclid's Algorithm

4) $(a*b)\%M = (a\%M * b\%M)\%M$.

```
int pow(int a, int N)
{
    int ans = 1;
    while(N>0)
    {
        ans = ans * a;
        N--;
    }
    return ans;
}
```

For greater values of M. this program fails.

Sometimes in Question it is mentioned to return the value after doing modulo operation with some value $M = 10^9+7$.

Modulo operation is costly. So we must minimize the use of it. When we calculate power for larger values long long datatype also fails to hold the value so simply returning ans%M can cause overflow and gives wrong answer.

To overcome this we could perform modulo inside while loop

ans = ans * a using Modulo Arithmetic

1) `ans = (ans%M * a%M)%M;`

↳ Instead of performing a%M everytime we could simply do it once before the while loop

2) `ans = (ans%M * a)%M`

↓ (2)

↳ Here after doing the modulo we are storing the value insided ans. So (2) can is unnecessary. We can further simplify it

3) `ans = (ans * a)%M;`

The final program is

```
long
int pow(int a, int N)
{   long ans = 1;
    a = a%M;
    while (N>0)
    {
        ans = (ans*a)%M;
        N--;
    }
    return ans;
}
```

$O(\log_2 N), O(1)$ — solution

```
def pow (a, N)
{   ans = 1;
    x = a;
    while (N!=0) :
        if (N%2 == 1):
            ans = ans * x
        x = x*x
        N = N>>1
    return ans;
}
```

N

11 → 1011 (4)
26 → 11010 (5)
31 → 11111 (5)
32 → 100000 (6)

$\log_2 N$

Compute a^N

1) O(N), 1

2) $a^{11} = a^{x_1} \cdot a^{x_2} \cdot a^{x_3} \cdots a^{x_m}$

$11 = x_1 + x_2 + x_3 + \cdots + x_m$

$x_i \in [1, 2, 4, 8, 16, 32, \ldots]$

$a^{11} = a^1 \cdot a^2 \cdot a^8$ (8 4 2 1) (1011)

$a^{26} = a^2 \cdot a^8 \cdot a^{16}$ (16 8 4 2 1) (11010)

$a^N = \{a^1, a^2, a^4, a^8, a^{16}, \ldots\}$

a^{11} N=11 → a a a a
1 0 1 1
3rd 2nd 1st 0th

ans = 1 , x = a

#bit	x	ans
0	a	ans*=a
1	a^2	a^3
2	a^4	a^3 (bit is not set)
3	a^8	a^{11}

```
for(int i=0 ; i<32; i++)
{
    if (checkBit (N, i) == T)
        ans = ans * x;
    x = x*x;
}
return ans;
```

TRIPLE TROUBLE

Q:

	0	1	2	3	4	5	6	7	8	9	10	11	12
arr:	7	10	18	18	10	12	5	7	12	12	7	10	18

o/p: 5

Find the unique element when every other element is repeated thrice

N=13

7 → 0 0 1 1 1

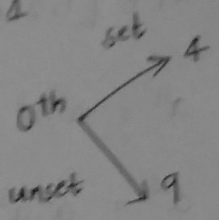
10 → 0 1 0 1 0

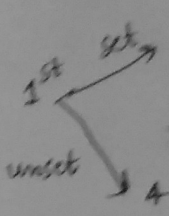
18 → 1 0 0 1 0

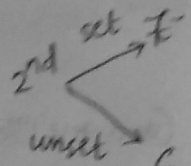
12 → 0 1 1 0 0

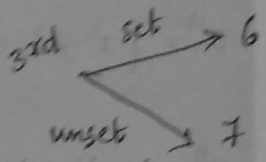
5 → 0 0 1 0 1

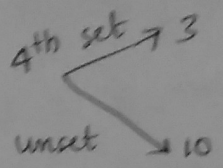
3x+1












```
int f (int ar[], int N)
 {
    int ans=0;
   for (int i=0 ; i<31; i++)
     {
       int c=0;
     for (int j=0 ; j<N ; j++)
      {
        if (checkBit (ar[j],i)==T)
            c++;
      }
    if (c%3 != 0)
      ans = ans + (1<<i);
 }
return ans;
}
```

Here in this code we are checking for the count of set bits if the count is divisible by 3 then there unique bit isn't present. So when the set bit count is not divisible by 3 then we add it to our answer. We traverse the array for 31 times for counting the set bit value.

Optimizing

⇒ array [0:a | 1:b | 2:c]

```
int f(int ar[], int N)
{
    int ans = 0;
    for(int i=0; i<N; i++)
        for(int j=0; j<N; j++)
            ans = ans ^ (ar[i] + ar[j]);
    return ans;
}
```

$O(N^2), O(1)$

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

$= 2a \wedge 2b \wedge 2c$

The program is optimized as

```
int f(int ar[], int N)
{
    int ans = 0;
    for(int i=0; i<N; i++)
        ans = ans ^ (ar[i] + ar[i]);
    return ans;
}
```

a b c

same

$a^a + a^b + a^c$

$b^a + b^b + b^c$

$c^a + c^b + c^c$

1) $N^2, 1$

2) $\frac{N(N-1)}{2} = \frac{N^2}{2} - \frac{N}{2} \Rightarrow O(N^2), O(1)$.

2Q:-

```
int f(int ar[], int N)
{
    int ans = 0;
    for(int i=0; i<N; i++)
        for(int j=0; j<N; j++)
            ans = ans + (ar[i] ^ ar[j]);
    return ans;
}
```

a b c d

$a^a + a^b + a^c + a^d$

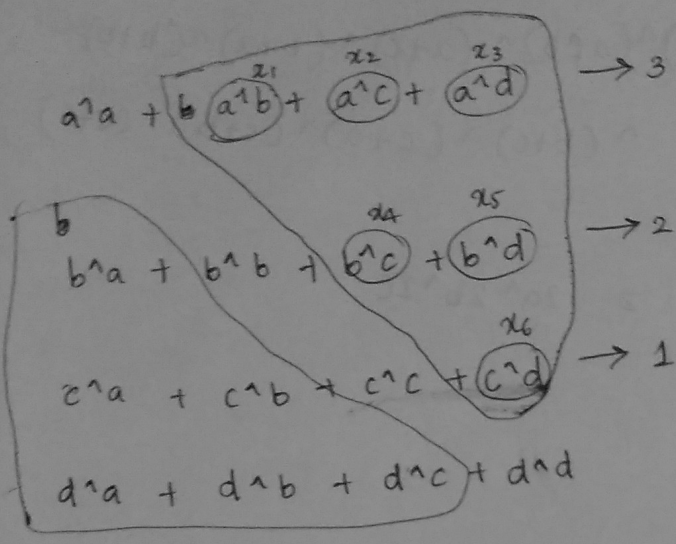
$b^a + b^b + b^c + b^d$

$c^a + c^b + c^c + c^d$

$d^a + d^b + d^c + d^d$

pg- 21

a	b	c	d
5	8	12	9



$ans = x_1 + x_2 + x_3 + x_4 + x_5 + \cdots + x_{\frac{N(N-1)}{2}} = 6$

$ans = 5\hat{}8 + 5\hat{}12 + 5\hat{}9 + 8\hat{}12 + 8\hat{}9$

$= 13 + 9 + 12 + 4 + 1 + 5$

$8+4+1 \quad 8+1 \quad 8+4 \quad 4 \quad 1 \quad 4+1$

$= 1*4 + 2*0 + 4*4 + 8*3 + 16*0 + 32*0 + \cdots$

$= \sum_{i=0}^{30} 2^i *$ # pair of elements which when XORed gives a set bit in i^{th} position

bit	set	unset	#pairs	ans
3rd	3	1	3*1	8*3
0th	2	2	2*2	1*4
1st	0	4	0*4	2*0

	8	4	2	1
5	0	1	0	1
8	1	0	0	0
12	1	1	0	0
9	1	0	0	1

```
int f (int ar[], int N)
{
    ans=0
    for i in range (0,31):
        st=0
        for j in range (N):
            if (checkBit (a[j], i) == T)
                st+=1
        ans = ans + (1<<i) * str (N-st
    return 2*ans;
}
```

Recursion

a) Assumption

b) Main Logic

c) Base Condition

```
int sum(int N)
{
    if (N == 0)
        return 0;
    return N + sum(N-1);
}
```

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

```
int fib(int N)
{
    if (N==1 || N==2)
        return 1;
    return fib(N-1) + fib(N-2);
}
```

```
int APSum(int a, int d, int N)
{   if (N==1) return a;
    return a + (N-1)d +
        AP(a, d, N-1);
}
```