

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 00010010

1's complement of 18 $\begin{array}{r} 11101101 \\ +1 \\ \hline 11101110 \end{array}$ \Rightarrow for 2's complement

128 64 32 16 8 4 2 0
0 0 0 1 0 0 1 0

$\begin{array}{r} -2^7 2^6 2^5 2^4 2^3 2^2 2^1 0 \\ 1 1 1 0 1 1 1 0 \end{array} \Rightarrow -18$

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

$00000101 (+5)$

$10000101 (-5) \Rightarrow$ MSB notation

-18 can be obtained from its binary form as

$$\begin{aligned} & -2^7 + 2^6 + 2^5 + 2^3 + 2^2 + 0 \\ & = -128 + 64 + 32 + 8 + 4 + 0 \\ & = -64 + 32 + 8 + 4 = -32 + 12 + 2 = \end{aligned}$$

* 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 long long long (8B) \Rightarrow used in C++

↓
2B 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 ($7, -8$)

<u># bits</u>	<u>signed</u>	<u>Unsigned</u>	
1	-1,0	1,0	$2^{10} = 1024 \approx 10^3$
2	-2,1	3,0	
4	-8,7	0,15	
8	-128,127	255,0	
n	$-2^{n-1}, 2^{n-1}$	$0, 2^n - 1$	
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

$$1 \leq N \leq 10^5$$

$$-10^6 \leq ax[i] \leq 10^6$$

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

Operators

Arithmetic $\Rightarrow \% + - / *$

a	b	a/b	a&b	a^b	no
0	0	0	0	0	1

Logical $\Rightarrow \&&, ||, !$

0	1	1	0	1	0
---	---	---	---	---	---

Relational $\Rightarrow <, \leq, >, \geq, ==, !=$

1	0	1	0	1	0
---	---	---	---	---	---

Bitwise $\Rightarrow \&, |, \wedge, \sim, \ll, \gg$

1	1	1	1	1	0
---	---	---	---	---	---

Bitwise operators are both commutative & associative

$$a/b = b/a \quad a^b = b^a \quad a \& b = b \& a$$

$$a/b/c = c/a/b$$

$$b/c/a = c/b/a$$

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

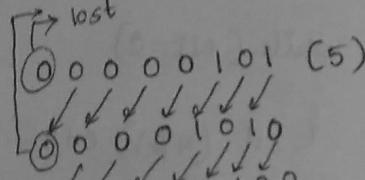
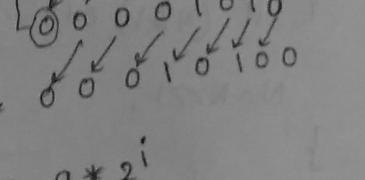
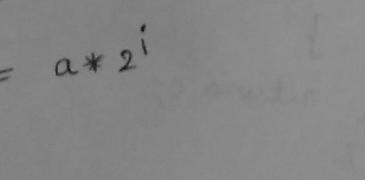
alb = 0 0 1 1 0 1 1 1

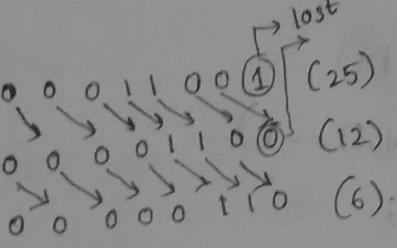
a & b = 0 0 0 0 0 1 0 0

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

~a = 1 1 0 1 1 0 0 1

Right shift & Leftshift

a =  (5)
a<<1 =  (10)
a<<2 =  (20)
a<<i = a * 2^i

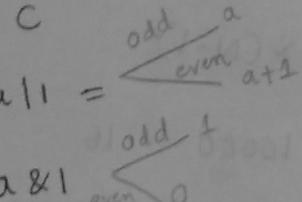
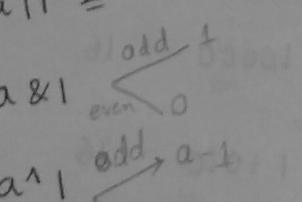
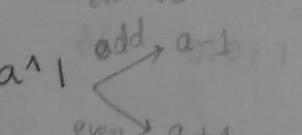

a>>i = a / 2^i

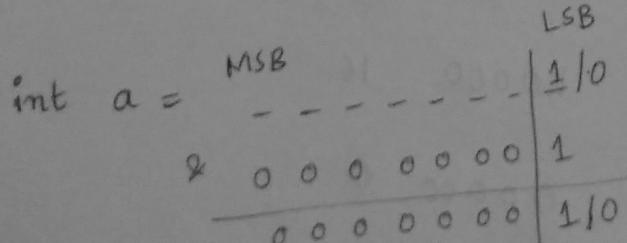
pow2(int N)
{ return 1 << N;

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

A
a|0 = a
a&0 = 0
a^0 = a

B
a|a = a
a&a = a
a^a = 0

C
a|1 = 
a&1 = 
a^1 = 

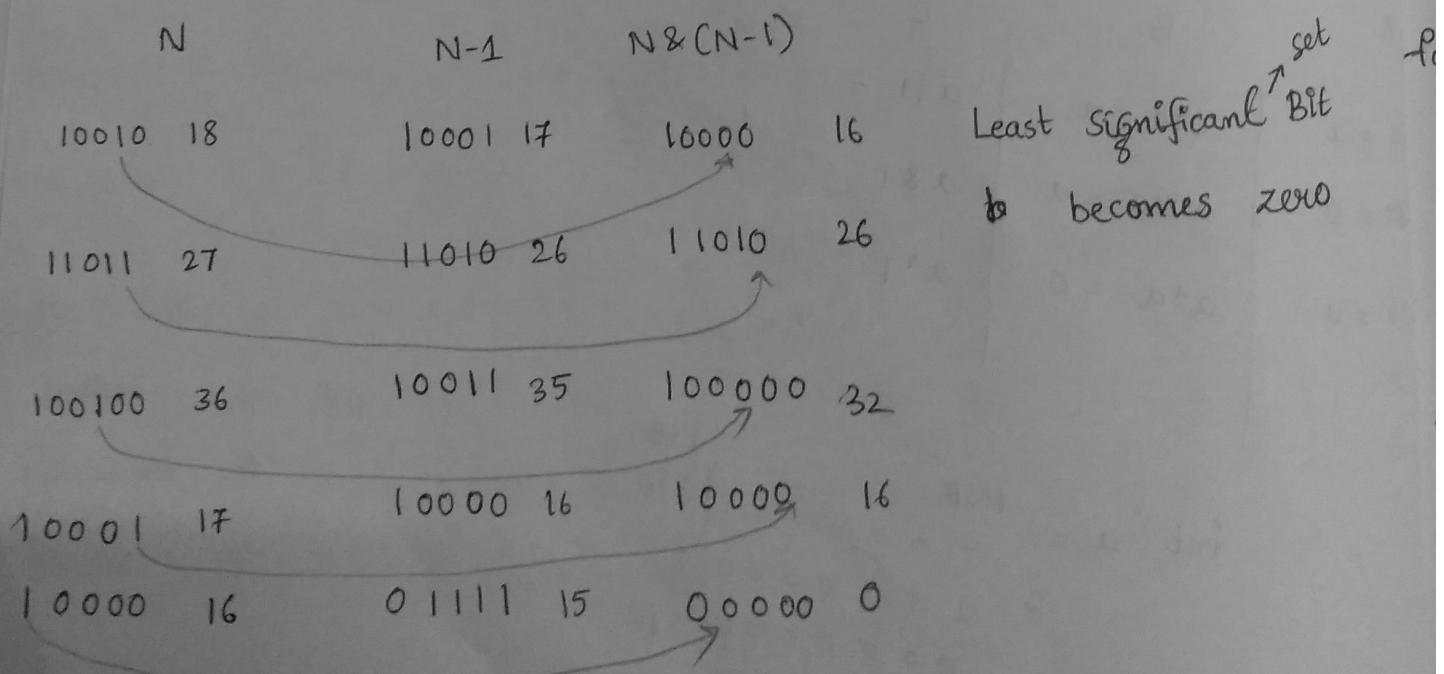
int a = 
----- | 110
* 0 0 0 0 0 0 |
----- | 110

$\text{bool checkBit}(\text{int } N, \text{int } i)$ $\left\{ \begin{array}{l} 0 \leq N \leq 10^9 \\ 0 \leq i \leq 30 \end{array} \right.$
 I/P O/P
 5,1 2nd & 0th F
 5,2 T
 9,3 3rd & 0th T

$\text{bool checkBit}(\text{int } N, \text{int } i)$ \rightarrow prefer this
 {
 return $((N \gg i) \& 1) == 1;$
 return $((N \gg i) \% 2) == 1;$ % costly int
 return $((N \& (1 \ll i)) != 0);$
 }
 ↳ +ve only

$\text{int countbits}(\text{int } N)$ $\left\{ 0 \leq N \leq 10^9 \right\}$
 {
 int c=0;
 for (int i=0; i<32; i++)
 if ($\text{checkBit}(N, i) == T$)
 c++;
 return c;

$\text{int countBits}(\text{int } N)$
 {
 int c=0;
 while ($N \neq 0$)
 { if ($(N \& 1) == 1$) c++;
 N = $N \gg 1$;
 }
 return c;



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;
    for (int i=1; i<=N; i++)
        if (N % i == 0)
            c++;
    return c;
}
    
```

Does this program work for 10^9 iterations?

$$N = 10^9$$

$$1 \text{ GHz} = 10^9 \text{ instructions/sec}$$

$$N = 10^{18}$$

$$\# \text{ iterations} = 10^9$$

$$\# \text{ instructions/iteration} = 5 \times 10^9$$

$$\# \text{ instructions} = 5 \times 10^9 \times 10^9$$

$$\text{Time (1 GHz)} = 5 \text{ sec}$$

$$10^{18} \text{ sec (approx 31.1 years)}$$

$$10^9 \text{ inst} \rightarrow 1 \text{ sec}$$

$$10^{18} \text{ inst} \rightarrow 10^9 \text{ sec}$$

$$N = 100$$

	$1 \cdot 100$
1	$2 \cdot 50$
2	$4 \cdot 25$
4	$10 \cdot 10$
5	$5 \cdot 4$
10	$10 \cdot 2$
20	$20 \cdot 1$

observe here

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

we can find
all factors

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

$$\equiv O(1)$$

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

$$O(\log_2 N)$$

$$2^K$$

$$K+1$$

$$\log_2 N + 1$$

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;
}

```

	<u>i</u>	<u>j</u>	<u>total</u>
	0	[N, 0)	$\frac{N(N+1)}{2}$
	1	[N, 1)	$= \frac{N^2 + N}{2}$
	2	[N, 2)	
	:	:	$O(N^2)$
	N-1	[N, N-1)	

```
int a=0, i=N;
```

```
while (i>0)
```

```
{ a+=i;
```

```
    i/=2;
```

```
}
```

	<u>i</u>	<u>total</u>
	N	1
		+
	N/2	1
		+
	N/4	1
		+
	N/8	1
		+
	N/16	1
		+
	N/32	1
		+
	N/64	1
		+
	N/128	1
		+
	N/256	1
		+
	N/512	1
		+
	N/1024	1
		+
	N/2048	1
		+
	N/4096	1
		+
	N/8192	1
		+
	N/16384	1
		+
	N/32768	1
		+
	N/65536	1
		+
	N/131072	1
		+
	N/262144	1
		+
	N/524288	1
		+
	N/1048576	1
		+
	N/2097152	1
		+
	N/4194304	1
		+
	N/8388608	1
		+
	N/16777216	1
		+
	N/33554432	1
		+
	N/67108864	1
		+
	N/134217728	1
		+
	N/268435456	1
		+
	N/536870912	1
		+
	N/1073741824	1
		+
	N/2147483648	1
		+
	N/4294967296	1
		+
	N/8589934592	1
		+
	N/17179869184	1
		+
	N/34359738368	1
		+
	N/68719476736	1
		+
	N/137438953472	1
		+
	N/274877906944	1
		+
	N/549755813888	1
		+
	N/1099511627776	1
		+
	N/2199023255552	1
		+
	N/4398046511104	1
		+
	N/8796093022208	1
		+
	N/17592186044416	1
		+
	N/35184372088832	1
		+
	N/70368744177664	1
		+
	N/140737488355328	1
		+
	N/281474976710656	1
		+
	N/562949953421312	1
		+
	N/112589990684264	1
		+
	N/225179981368528	1
		+
	N/450359962737056	1
		+
	N/900719925474112	1
		+
	N/1801439850948224	1
		+
	N/3602879701896448	1
		+
	N/7205759403792896	1
		+
	N/14411518807585792	1
		+
	N/28823037615171584	1
		+
	N/57646075230343168	1
		+
	N/11529215046068632	1
		+
	N/23058430092137264	1
		+
	N/46116860184274528	1
		+
	N/92233720368549056	1
		+
	N/184467440737098112	1
		+
	N/368934881474196224	1
		+
	N/737869762948392448	1
		+
	N/1475739525896784896	1
		+
	N/2951479051793569792	1
		+
	N/5902958103587139584	1
		+
	N/11805916207174279168	1
		+
	N/23611832414348558336	1
		+
	N/47223664828697116672	1
		+
	N/94447329657394233344	1
		+
	N/18889465931478846688	1
		+
	N/37778931862957693376	1
		+
	N/75557863725915386752	1
		+
	N/15111572745823077504	1
		+
	N/30223145491646155008	1
		+
	N/60446290983292310016	1
		+
	N/120892581966584620032	1
		+
	N/241785163933169240064	1
		+
	N/483570327866338480128	1
		+
	N/967140655732676960256	1
		+
	N/1934281311465353920512	1
		+
	N/3868562622930707841024	1
		+
	N/7737125245861415682048	1
		+
	N/15474250491722831364096	1
		+
	N/30948500983445662728192	1
		+
	N/61897001966891325456384	1
		+
	N/123794003933782650912768	1
		+
	N/247588007867565301825536	1
		+
	N/495176015735130603651072	1
		+
	N/990352031470261207302144	1
		+
	N/1980704062940522414604288	1
		+
	N/3961408125881044829208576	1
		+
	N/7922816251762089658417152	1
		+
	N/15845632533524179316834304	1
		+
	N/3169126506704835863366864	1
		+
	N/6338253013409671726733728	1
		+
	N/1267650602681934345346456	1
		+
	N/2535301205363868690692912	1
		+
	N/5070602410727737381385824	1
		+
	N/1014120482145547476277168	1
		+
	N/2028240964291094952554336	1
		+
	N/4056481928582189905108672	1
		+
	N/8112963857164379810217344	1
		+
	N/1622592771432879820443488	1
		+
	N/3245185542865759640886976	1
		+
	N/6490371085731519281773952	1
		+
	N/1298074217146303856354784	1
		+
	N/2596148434292607712709568	1
		+
	N/5192296868585215425419136	1
		+
	N/10384593737170430850838272	1
		+
	N/20769187474340861701676544	1
		+
	N/41538374948681723403353088	1
		+
	N/83076749897363446806706176	1
		+
	N/16615349779472689361341232	1
		+
	N/33230699558945378722682464	1
		+
	N/66461399117890757445364928	1
		+
	N/132922798235781514890729856	1
		+
	N/265845596471563029781459712	1
		+
	N/531691192943126059562919424	1
		+
	N/1063382385886252119125838848	1
		+
	N/2126764771772504238251677696	1
		+
	N/4253529543545008476503355392	1
		+
	N/8507059087090016953006710784	1
		+
	N/17014118174180339066013421568	1
		+
	N/34028236348360678132026843136	1
		+
	N/68056472696721356264053686272	1
		+
	N/13611294539344271252810737244	1
		+
	N/27222589078688542505621474488	1
		+
	N/54445178157377085011242948976	1
		+
	N/10889035631475417002248589752	1
		+
	N/21778071262950834004497179504	1
		+
	N/43556142525801668008994359008	1
		+
	N/87112285051603336017988718016	1
		+
	N/174224570103206672035977436032	1
		+
	N/348449140206413344071954872064	1
		+
	N/696898280412826688143909744128	1
		+
	N/139379656082565337628781548824	1
		+
	N/278759312165130675257563097648	1
		+
	N/557518624330261350515126195296	1
		+
	N/111503724866530270103025239056	1
		+
	N/223007449733060540206050478112	1
		+
	N/446014899466121080412100956224	1
		+
	N/892029798932242160824201912448	1
		+
	N/1784059597864484321648403824896	1
		+
	N/3568119195728968643296807649792	1
		+
	N/7136238391457937286593615299584	1
		+
	N/14272476782915874573187230599168	1
		+
	N/28544953565831749146374460198336	1
		+
	N/57089907131663498292748920396672	1
		+
	N/114179814263326985855497840793344	1
		+
	N/228359628526653971710995681586688	1
		+
	N/456719257053307943421991363173376	1
		+
	N/913438514106615886843982726346752	1
		+
	N/182687702821323577368796545269304	1
		+
	N/36537540564264715473759309053808	1
		+
	N/73075081128529430947518618107616	1
		+
	N/14615016225705886189503723621532	1
		+
	N/29230032451411772379007447243064	1
		+
	N/58460064902823544758014894486128	1
		+
	N/11692012905646788556029788972256	1
		+
	N/23384025811293577112059577944512	1
		+
	N/46768051622587154224119155889024	1
		+
	N/93536103245174308448238311778048	1
		+
	N/187072206490348616896476623556096	1
		+
	N/374144412980697233792953247112192	1
		+
	N/748288825961394467585906494224384	1
		+
	N/149657765192278893517181298844864	1
		+
	N/299315530384557787034362597689728	1
		+
	N/598631060769115574068725195379456	1
		+
	N/119726212153823114813550390675888	1
		+
	N/239452424307646229627100781351776	1
		+
	N/478904848615292459254201562703552	1
		+
	N/957809697230584918508403125407104	

```

int k=0;
for (int i=N/2; i<=N; i++)  $\nearrow N/2$ 
    for (int j=2; j<=N; j=j*2)  $\nearrow \log_2 N$ 
        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++;

```

```

}

```

```

}

```

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

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

$arr[0]$	$arr[1]$	$arr[2]$	$arr[3]$	$arr[4]$	$arr[5]$	$arr[6]$	$arr[7]$
\uparrow							
$j=0$							N

```

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++) {

```

$O(1)$

```

    }
}

```

<u>i</u>	<u>j</u>	<u>total</u>
1	$[1, 1^k]$	1^k
2	$[1, 2^k]$	2^k
3	$[1, 3^k]$	3^k
.	.	.
N	$[1, N^k]$	N^k

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

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

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

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

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

Suppose
sol A $10N^3 + 5N + 10$
sol B $15N^2 + 10000$

$\rightarrow O(N^3)$

$\rightarrow O(N^2)$

Sol-A may be 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 threshold

	Big-O: N^3	N^2	$N \log_2 N$	N	\sqrt{N}	$\log_2 N$	1	$\frac{2^N}{N=30}$	$N=60$
#iterations:	10^{18}	10^2	$2 \cdot 10^7$	10^6	10^3	20	1	10^9	10^{18}
Time:	1 sec	10 ³ sec	$2 \cdot 10^{-2}$ sec	0.001 sec	1 ms	10^{-6} sec	20 ns	1 ns	1 sec

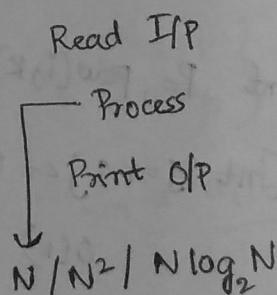
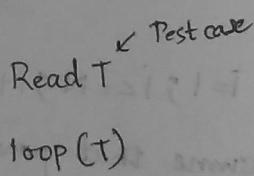
↓
millisecond

microsec

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

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

for a program with Test cases



Total Time complexity

is $N \cdot T$ $M = (N, N^2)$
 $N^2 \cdot T$ $N \log_2 N$
 $N \log_2 N \cdot T$

Here

$T \cdot M \leq 10^8$

for TLE not exceeding

for suppose

Constraints

$$1 \leq T \leq 100$$

$$1 \leq N \leq 10^6$$

$$O(N^2) - 10^2 \times 100 = 10^{14} \text{ iterations} = 10^5 \text{ sec (TLE)}$$

$$\left. \begin{array}{l} 1 \leq T \leq 10^3 \\ 1 \leq N \leq 10^6 \end{array} \right\} O(N) \quad N \cdot T = 10^9 \text{ (TLE)}$$

$$\left. \begin{array}{l} 1 \leq T \leq 10 \\ 1 \leq N \leq 10^{18} \end{array} \right\} O(N) \quad 10^3 \cdot 10^3 = 10^6 \leq 10^8 \checkmark$$

Constraint

⇒ Helps in determining data-type

⇒ Helps us towards an approach
to solution

SPACE COMPLEXITY

void f(int arr[], int N)

```
{  
    int x, y, z;      12B  
    double d1, d2;   16B  
    int b[10];        40B  
}
```

This is $O(1)$
Because it is
a constant &
doesn't depend
on N

For
 $\left. \begin{array}{l} \text{int arr}[N] \\ \text{double d}[N] \\ \text{int arr}[N][N] \end{array} \right\} O(N)$

$$\left. \begin{array}{l} 1 \leq T \leq 100 \\ 1 \leq N \leq 10^5 \end{array} \right\} 10^9 B \approx 1GB$$

int mat[N][N]; $\Rightarrow 40GB$ RTE

$$4B \quad 10^5 \quad 10^5$$

4B \cdot 10^6
int arr[10^6] = 4MB

$$360MB$$

$$O(s) \leq 10$$

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

$$6-7.8$$

bool

$$\text{bool arr}[10^8] = 100 MB$$

$$1B \cdot 10^8$$

Find the duplicate

1Q: $\text{arr}[N] : \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 3 & 5 & 8 & 2 & 8 & 5 & 10 & 2 & 3 \end{matrix}$

O/P: 10

$$a \wedge a = 0$$

$O(N^2)$ ① int f(int arr[], int N)

{
for (int i=0; i<N; i++)
 bool dup = false;

{
for (int j=0; j<N; j++)

{
if (arr[i] == arr[j] && i != j)

~~break~~ dup = true;

}

if (!dup) return arr[i];

}

② int f(int arr[], int N)

{
int ans = 0;
for (int i=0; i<N; i++)

ans \wedge arr[i];

return ans;

① $O(N^2), O(1) \rightarrow N^2 / 1$

② $O(N), O(1) \rightarrow N, 1$

2Q: Subset

$\text{arr}_N : \begin{matrix} 3 & 10 & 5 & -6 & 18 & -2 \end{matrix}$

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

$$\begin{array}{lll} K = 26, 25, 0, 50, 1 & 26 - [10, 18, -2] & 10 + 18 - 2 = 26 \\ \overline{T} \quad \overline{T} \quad \overline{T} \quad \overline{F} \quad \overline{T} & 0 - [3, 5, -6, -2] & 3 + 5 - 6 - 2 = 0 \end{array}$$

i) Consider a simple array of 2 elements $[a, b]$

No. of subsets a, b, ab, \emptyset

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

No. of subsets $a, b, c, ab, bc, ca, abc, \emptyset$

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

$\text{arr}[0]=3 \quad \text{arr}[1]=10 \quad \text{arr}[2]=5$
 2nd bit 1st bit 0th bit
 4 2 1 \emptyset
 0 0 0 \emptyset
 0 0 1 $\{3\}$
 0 1 0 $\{10\}$ $O(N \cdot 2^N), O(1)$
 0 1 1 $\{3, 10\}$
 1 0 0 $\{5\}$
 1 0 1 $\{3, 5\}$
 1 1 0 $\{10, 5\}$
 1 1 1 $\{3, 10, 5\}$

$\text{bool f(int arr[], int N, int K)}$
 $\{$
 $\text{for (int i=0; i < (1<<N); i++)}$
 $\{$
 int s=0, j=0;
 while (j > 0)
 $\{$
 if (i&1 == 1)
 st = arr[j];
 j++;
 $\}$
 i = i >> 1;
 $\}$
 $\text{if (s == K) return T;}$

$\text{for (int j=0; j < N; j++)}$

$\text{if (checkBit(i, j) == T)}$

$st = arr[j];$

$\} // \text{if } i = K;$

return F;

subsets

no, $i < 64$; $i++$

0 1 2 3 4 5
3 10 5 -6 18 -2

$i=1$

0001

$(3, 1) = 3 \quad st = 3 \quad s = 3$

$i=2$

0010

$0001 >> 1 \quad s = 3 \quad i = K$

$0010 \quad 0000$

$i = K$

do b a

Set has n elements. We will use the bits of numbers to show the presence of elements in set. The key is to use same prime with same i th bit in the number vs.

always always when SET $i=1$, then i th element in given set is present in current subset. We will loop

from 0 to $(2^k)-1$ and for

each number, we will check the

SET Bits in its in it & take

corresponding elements

cba

000

{a}

010

{b}

011

{a, b}

000

{c}

101

{a, c}

110

{a, b, c}

111

{a, b, c}

100

{ }

$\text{for (int i=0; i < } 2^k; i++)$

$\{$
 int sum=0;
 $\text{for (int j=0; j < N; j++)}$

$\{$
 $\text{if (checkBit(st, j) == 1)}$

$\{$
 $\text{sum = sum + arr[j];}$

$\}$
 $\}$
 $\}$

$\}$
 $\}$
 $\}$

$\}$
 $\}$
 $\}$

$\text{let } f(1, 2, 3, 4) = 2015$

Modulo Arithmetic

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

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

$$3) (a-b) \% M = (a \% M - b \% M + M) \% M$$

$$4) (a/b) \% M = (a \% M * b^{-1} \% M) \% M$$

Inverse Modulo
↳ Found by Extended Euclid's Algorithm

$$5) (a * b) \% M = ((a \% M * b \% M) \% M)$$

```
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 inside ans. So ② can
be 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;
```

```
}
```

$O(\log_2 N)$, $O(1)$

```
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;
```

```
}
```

$\frac{N}{2^i}$

11 → 1011 (4)

26 → 11010 (5)

31 → 11111 (5)

32 → 100000 (6)

Compute a^N

i) $a(N), 2$

ii) $a^n = a^{x_1} \cdot a^{x_2} \cdot a^{x_3} \cdots a^{x_m}$

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

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

$a^n = a^1 \cdot a^2 \cdot a^8 \quad (1011)$

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

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

$a^n \quad N=11 \rightarrow \begin{array}{cccccc} & a & a & a & a \\ & | & | & | & | \\ 3 & 2 & 1 & 0 & 0^{th} \end{array}$

$ans=1, x=a$

# bit	x	ans
0	a	ans 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:- arr: 0 1 2 3 4 5 6 7 8 9 10 11 12
 op: 5
 Find the unique element where every other element is repeated thrice

$N=13$

$\# \rightarrow 00111$	$3x+1$
$10 \rightarrow 01010$	0^{th} set $\rightarrow 4$
$18 \rightarrow 10010$	unset $\rightarrow 9$
$12 \rightarrow 01100$	1^{st} set $\rightarrow 9$
$5 \rightarrow 00101$	unset $\rightarrow 4$
	2^{nd} set $\rightarrow 7$
	unset $\rightarrow 6$
	3^{rd} set $\rightarrow 6$
	unset $\rightarrow 7$
	4^{th} set $\rightarrow 3$
	unset $\rightarrow 10$

int f (int arr[], int N)

{

 int ans = 0;

 for (int i=0; i<31; i++)

{

 int c=0;

 for (int j=0; j<N; j++)

{

 if (checkBit(arr[j], i) == 1)

}

 c++;

 flexe 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.

 if ($c \% 3 \neq 0$)

}

 ans = ans + (1 << i);

 return ans;

}

Optimizing

int f(int arr[], int N)

{
 int ans = 0;

 for (int i=0; i<N; i++)

 for (int j=0; j<N; j++)

 ans = ans ^ (arr[i] + arr[j]);

 return ans;

}

The program is optimized as

int f(int arr[], int N)

{
 int ans = 0;

 for (int i=0; i<N; i++)

 ans = ans ^ (arr[i] ^ arr[i]);

 return ans;

}

a b c

$$\begin{array}{c} \text{same:} \\ a^0 a + a^1 b + a^2 c \\ b^0 a + b^1 b + b^2 c \\ c^0 a + c^1 b + c^2 c \end{array}$$

1) NP1

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

⇒ array

0	1	2
a	b	c

$$O(N^2), O(1) = (a+a)^n (a+b)^n (a+c)^n (b+a)^n (b+b)^n \\ (b+c)^n (c+a)^n (c+b)^n (c+c)^n$$

$$= 2a^n 2b^n 2c^n$$

Q: int f (int arr[], int N)

{

 int ans = 0;

 for (int i=0; i<N; i++)

 for (int j=0; j<N; j++)

 ans = ans + (arr[i]^arr[j]);

 return ans;

}

a b c d

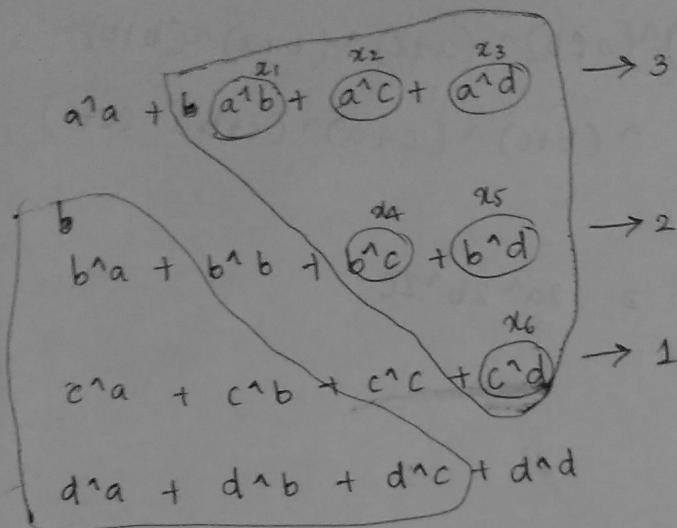
$$\begin{array}{c} a^0 a + a^1 b + a^2 c + a^3 d \\ b^0 a + b^1 b + b^2 c + b^3 d \\ c^0 a + c^1 b + c^2 c + c^3 d \\ d^0 a + d^1 b + d^2 c + d^3 d \end{array}$$

ans = ?

a b c d
5 8 12 9

$$\text{ans} = x_1 + x_2 + x_3 + x_4 + x_5$$

$$+ \dots + x_{\frac{N(N-1)}{2}} = 6$$



$$\text{ans} = 5^8 + 5^{12} + 5^9 + 8^{12} + 8^9$$

$$= 13 + 9 + 12 + 4 + 1 + 5 \\ = 8+4+1 = 8+1 = 8+4 = 4+1 = 1+1 = 4+2$$

$$= 1*4 + 2*0 + 4*4 + 8*3 + 16*0 \\ = 32.0 + \dots$$

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

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

int f (int arr[], int N)

{
 ans = 0

for i in range (0, 31):

 st = 0

 for j in range (N):

 if (checkBit (arr[j], i) == 1)

 st += 1

 ans = ans + (1 << i) * st * (N - st)

return ans;

}

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

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);
```