→ Modulo :-

#Math_algo_nu...

$$(a+b) \% m = ((a \% m) + (b \% m)) \% m$$
$$(a-b) \% m = ((a \% m) - (b \% m)) \% m$$
$$(a*b) \% m = ((a \% m) * (b \% m)) \% m$$
$$(a/b) \% m = ((a \% m) * (inv(b) \% m)) \% m$$

✓✓ C++ order of calculation →

Double
Float
long long Int
long Int
Int
char

int a = 3/2 a = 1

double a = 3/2.0 a = 1.5

double a = 3/2 a = 1 # calculation done on int then stored in double a.

int a = 3/2.0 a = 1

#⭐

✗ int → -10^9 to 10^9
long → 10^{12}
long long → 10^{18}

✓✓ int a = 10^5 b = 10^5
long long c = a * b ✗ error
long long c = a * b * (1 long long)

✓✓ print Binary (a) a=3
↳ 0000 0011

#C++STL

✓✓ function to count number of 1 bit

__builtin_popcount(n); → int ✓
__builtin_popcountll(n); → long long ✓

✓✓ check even & odd (n & 1) → odd
else even

↓
faster than n%2 == 0 ...

✓ $n >> 1$ $n/2$
$n << 1$ $n * 2$

→ **Uppercase & lowercase :—**

'A' = 000 .... 0 1 0 0 0 0 0 1 (6 5 4 3 2 1 0)

'a' = 000 .... 0 1 1 0 0 0 0 1

all uppercase & lowercase differ by 5th bit.

A | (1<<5) == a

a & (~(1<<5)) == A

ASCII of " " = 0 0 0 1 0 0 0 0 0

ASCII of "_" = 0 0 1 0 1 1 1 1 1

#⭐ #Trick

B | ' ' = b space convert lowercase

b & '_' = B underscore convert uppercase.

→ **Clear last n bit :-**

$a' = a \& (\sim((1 << n) - 1))$

e.g. 1 0 1 1 0 0 1 1 0 1 $n = 6$

(6 5 4 3 2 1 0)

(1<<6) 1 0 0 0 0 0 0

(1<<6)-1 0 1 1 1 1 1 1

~(1<<6-1) 1 1 1 0 0 0 0 0 0

#⭐ #Trick

if $(n \& (n-1))$ → n is power of 2

#algorithm/XOR

→ **XOR :-**

$a\bar{b} + b\bar{a}$

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

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

a = 2
b = 3

in order

a = a ^ b
b = a ^ b = a ^ b ^ b = a
a = a ^ b = a ^ b ^ a = b

$a \wedge a = 0$
$a \wedge 0 = a$

→ Bit Masking :-

person1 → ability { 2, 3, 5 }
person2 → ability { 1, 3 }
person3 → ability { 2, 3, 4 }
⋮

abilities

	6	5	4	3	2	1	0
	0	1	0	1	1	0	0
	0	0	0	1	0	1	0
	0	0	1	1	1	0	0

ability list = [5, 8, 7, 2, 3, 9]

```
for ( int i=0 ...
    if ( ab_person & (1<< ability_list[i]) == 1 )
            apility list[i] is present.
```

E.g. Subset generation ;

arr = [1 2 2]

0 →	0	0	0	[ ]
1 →	0	0	1	[2]
2 →	0	1	0	[2]
3 →	0	1	1	[2 , 2]

⋮

```
ans = [[ ]]
n =  size (arr).

for (mask =0; mask <(1<<n) ; mask ++)
{    for (i=0; i <n; i++)
      { if (mask & (1<< i))
                temp . push_back (arr[i]);
     ans . push_back (temp).
```

→ GCD & LCM :-

$$gcd(a, b) * Lcm(a, b) = a \times b$$

#⭐

```
def gcd(a , b):
    if(b ==0) return a ;
    return ( b, a % b) ;
```

} if b > a no problem

$$18 \overline{)12}\ 0$$

$$12 \overline{)18}\ 1$$

⋮

→ Binary Exponentiation :-

$exp(a, b) = a^b$

```
int exp(a, b){
    if (b == 0) return 1;
    long long temp = exp(a, b/2);
    if (b & 1) return a * temp * temp      // odd case
    else       return temp * temp          // even case
}
```

if (b<0) return $exp(\frac{1}{a}, -b)$

Iterative method :-

```
int ans = 1
while (b){
    if (b & 1) ans = ans * a * (1LL) %m
    a = a * a * (1LL) %m
    b = (b >> 1)
}
return ans;
```

Best with the mod operation otherwise power of a goes too high.

Dry Run

28	14	7	3	1
$a = a^2$	a^4	a^8	a^{16}	a^{32}
ans = 1	1	a^9	a^{8+9}	a^{8+9+16}

7	3	1
$a = a^2$	a^4	a^8
ans = a	a^{1+2}	a^{1+2+8}

→ Multiply Very long Numbers :-

```
int ans = 0
while (b > 0){
    if (b & 1) ans = ans + a
    a = a + a
    b = b >> 1
}
```

a = 10 b = 7

ans	a	b
10	20	3
30	40	1
70	80	0
ans		

Number Theory

https://www.geeksforgeeks.org/eulers-totient-function/#:~:text=Euler's%20Totient%20function%20%CE%A6(n,Divisor)%20with%20n%20is%201.

Euler Totion Function (ETF)

ETF of n, $\phi(n)$ = count k such that k & n are co prime.

$\phi(n) = n \times \prod (1 - \frac{1}{p})$ p = prime factors of n.

→ Euler's theorem :-

$$a^b \equiv a^{b \bmod \phi(n)} \bmod n$$

$$\Rightarrow a^b \,\%\, n = a^{b \,\%\, \phi(n)} \,\%\, n$$

congruence modulo
$a \equiv b \mod(n)$
a & b are equivalent upon mod(n)
e.g $26 \equiv 11 \mod(5)$

if n is prime ; $\phi(n) = n \times 1(1 - \frac{1}{n})$

$= (n-1)$

$$\Rightarrow a^b \,\%\, n = a^{b \,\%\, n-1} \,\%\, n$$

this comes less than n-1

comes from here

#⭐, #Trick

→ Calculate Higher Power :-

$a^{b^c} = \exp(a, \exp(b, c, m-1), m)$

```
int exp(a, b, m){
    some old shit
};
```

→ Sum and Count of Devisors :-

$24 =$ 1×24, 2×12, 3×8, 4×6, 6×4, ⋮

$24 = 2^3 \times 3^1$

choices $= (3+1)(1+1) = 8$

Sum $= (1 + 2 + 2^2 + 2^3)(1 + 3)$

GP series ...

$n = p_1^{x_1} \; p_2^{x_2} \cdots$ p_1, p_2 are prime numbers.

count of devisors $= (x_1+1)(x_2+1)\cdots$

sum of devisors $= \frac{p_1^{x_1+1}-1}{p_1-1} * \frac{p_2^{x_2+1}-1}{p_2-1}$

→ Generate Prime Factors :-

```
unordered_map<int, int> mp;
for (i=0 ; i < (n) → or √n & append n afterwards. ; i++){
    while ( n % i == 0){
        mp[i]++;
        n = n/i ; } }
if (n>1) mp[n]=1;
```

→ Seive Algorithm :-

complexity of seive $O(n \log\log n)$

```
vector <bool>  isPrime ( n, true);
isPrime [0] = isPrime [1] = 0 ;
for ( i=2; i< n ; i++ ){
    if (isPrime[i]){
        for( j= 2*i ; j<n ; j+=i)
            isPrime[j] = 0 } }
```

highest & lowest prime :-

```
for ( i=2; i< n ; i++ ){
    if (isPrime[i]){
        for( j= 2*i ; j<n ; j+=i){
            isPrime[j] = 0
            highestPrime[j] = i
            if (lowestPrime[j]==0) lowestPrime[j] = i}
    }
}
```

✓ Prime Factors from highest Prime:

```
while (num > 1) {
    int prime_factor = highest_Prime[num];
    while (num % prime_factor == 0){
        ans.push_back(prime_factor)
        num = num / prime_factor;
```

→ Division Modulo :-

Fermat's Theorem :-

if m is prime and A is not multiple of m.

$$A^{m-1} \equiv 1 \mod(m)$$

$$A^{m-1} \times A^{-1} \equiv A^{-1} \mod(m)$$

$$\Rightarrow A^{m-2} \% m = A^{-1} \% m$$

binaryexponent (A, m-2, m)

Now; $(a/b)\% = [(a\%m) * (b^{-1}\%m)] \%m$

$= [(a\%m) * binexp(b, m-2, m)] \%m$

→ Square Root :-

```
double ee = 1e-6;
double i = 0, j = n, mid;
while ((j - i) > ee){
    mid = i+j/2
    if (mid * mid < n) i = mid
    else               j = mid
return mid;
```