Bit manipulation

(num>>(i-1))&1 returns number ← returny bit

1. Get i^{th} bit ⇒ num & (1<<(i-1))
   set i^{th} bit ⇒ num | (1<<(i-1))
   clear i^{th} bit ⇒ num & ~(1<<(i-1))
   Toggle i^{th} bit ⇒ num ^ (1<<(i-1))

0 ⇒ 8 ⇒ 1000
9 ⇒ 1001

```
4 3 2 1
3 2 1 0
1 0 0 0
0 0 0 1
1 0 0 1
```

70 ⇒ 1 0 0 0 1 1 0

7 6 5 4 3 2 1

Get i
Set
clear

2. Count no. of 1 Bits

```
while (n!=0){
    cnt += n&1;
    n = n>>1;
}
```

3. k-th Bit is set or not

n=4 k=0 ⇒ (n>>k)&1
n=4 k=2
n=500 k=3

4. Non Repeating Numbers

I⇒ 1,2,3,2,4 O⇒ 3,4 occur only once in increasing order

2 1 3 2 ⇒ 1 3

2 1 3 3 ⇒ 1 2

2n+2 +ve Nums

5. Odd or Even

	Odd	even
15 ⇒	1 ⇒ 0001	10 ⇒ 1100
44 ⇒	7 ⇒ 0111	2 ⇒ 0010

return (n&1)==0 is even

6. Bit Difference

16 8 4 2 1
10 ⇒ 0 1 0 1 0
20 ⇒ 1 0 1 0 0
× × × × ✓

compare in while loop

out ⇒ 4 bits

7. Check if power of two

1 ⇒ 2^0=1 true 00001
16 ⇒ 2^4=16 true 10000
3 ⇒ 10 ~~false~~ 00011

Count no. of 1's ==1

8. count total set bits

4 ⇒ 1 → 0000 → 0
0001 → 1
2 → 0010 → 1
3 → 0011 → 2
4 → 0100 → 1
} 5 output

expected TC ⇒ O(log n) SC O(1)

16 8 4 2 1
0 1 0 1 → 2
0 1 1 0 → 2
0 1 1 1 → 3
1 0 0 0 → 1
1 0 0 1 → 2
1 0 1 0 → 2
1 0 1 1 → 3
1 1 0 0 → 2
1 1 0 1 → 3
1 1 1 0 → 3
1 1 1 1 → 4
1 0 0 0 0 → 1

1 1 2
1 2 2 3
1 2 2 3 2 3 3 4

Power of 2 ⇒ 1

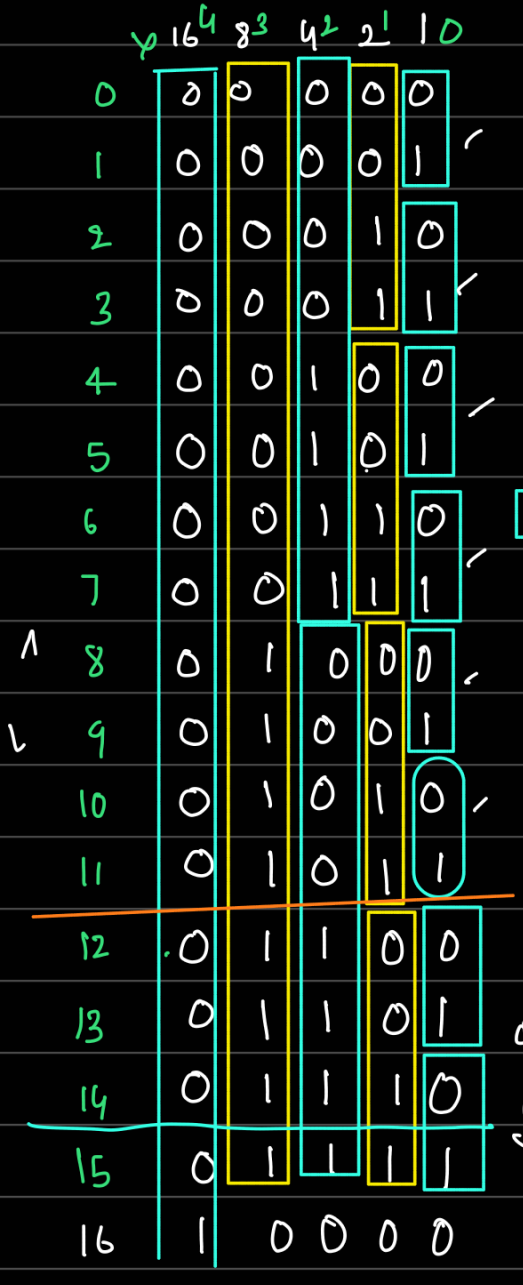
4 3 2 1 0
8 4 2 1
2^3 2^2 2^1 2^0
ones

```
class Solution {
  public:
    vector<int> singleNumber(vector<int>& arr) {
        // Code here.
        int unique_xor = 0;

        for(int i=0; i<arr.size(); i++){
            unique_xor ^= arr[i];
        }

        int rightmost_diff_bit = unique_xor & -unique_xor;

        int x_ele = 0, y_ele = 0;
        for(int i=0; i<arr.size(); i++){
            if( arr[i] & rightmost_diff_bit ) x_ele ^= arr[i];
            else y_ele ^= arr[i];
        }

        if(x_ele < y_ele) return {x_ele,y_ele};
        else return {y_ele,x_ele};
    }
};
```

num & 2 ((num&2)>>2) (or) num&2

all numbers give any num or zero

XOR each group to get unique element from that group



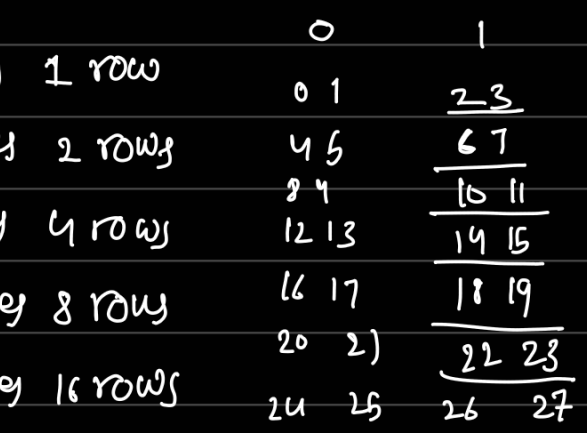
1 ⇒ changes 1 row

2 ⇒ changes 2 rows

4 ⇒ changes 4 rows

8 ⇒ changes 8 rows

16 ⇒ changes 16 rows



n=6 | 0 1 2 3 4 5 6

1 | 0 1 0 1 0 1 0 ⇒ No. of odds = (n+1)/2

2 | 0 0 1 1 0 0 1 ⇒

4 | 0 0 0 0 1 1 1 ⇒

8 | 0 0 0 0 0 0 0 ⇒

n=17 k=4

13 numbers

n=17

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 0

① ② ③ ④

17−4 = 13 | 13−4 = 9 | 9−4 = 5 | 5−4 = 1 | 1−4

0

4 0 4+4 = 8 0

Odd set ⇒ 0's

even set ⇒ 1's

$2^i \le 16 \le 2^{i+1}$

n/2i

17/4 = 4/2 = 2 *4 = 8

17%4 = 1

13/4 = 3/2 = 1 *4 = 4 → = 5

13%4 = 1

odd next even

9/4 = 2/2 = 1 *4 = 4

even next odd.

9%4 = 1 = 0

odd

n 15 i=1 fill=2

15/2 = 7/2 = 3*2 = 6

$2^0 = 1$

$2^1 = 2$

$2^2 = 4$

$2^3 = 8$

$2^4 = 16$

12 n=12 i=0

i=0 2^0 = 1 = fill

12/1 = 12/2 = 6 = ones

i=1 2^1 = 2 = fill

12/2 = 6/2 = 3

```
while (2^i > n){
    fill = 2^i;
    grps = n/(fill*2);
    ones = grps*(2^i);
    value = n%2^i;
    if ((n/fill) is odd)
        ones += value;
```

```
12  while (2^i <= n)
        i++
```

TC ⇒ log n

0 1 2 4 8 16 (log n)

count_set_bit_to_n.cpp

Striver_A2Z_DSA > bit_manipulation > count_set_bit_to_n.cpp > main()

```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution {
public:
    int countsetbits(int n) {
        int i=0, total_ones = 0;
        n=n+1;
        while(pow(2,i) <=n){
            cout << "i : " << i << endl;
            int fill = pow(2,i);
            cout << "fill : " << fill << endl;
            int odd_or_even = n/fill;
            cout << " odd or even : " << odd_or_even << endl;
            int grps = odd_or_even/2;
            cout << " grps : " << grps << endl;
            int ones = grps*fill;
            cout << " ones : " << ones << endl;
            int value = n%(fill);

            cout << "value : "<< value << endl;

            if(odd_or_even%2 == 1){
                ones += value;
            }
            total_ones += ones;
            cout << "i : " << i << " fil : " << fill << " ones : " << ones << " total_ones : " << total_ones << endl;
            i++;
            // break;
        }
        cout << "returning : " << total_ones << endl;
        return total_ones;
    }
};

int main(){
    Solution sol;
    int n;
    cin >>n;
    int out = sol.countsetbits(n);
    cout << " final total ones : " << out << endl;
    return 0;
}
```

(9) power of 2 $0 \le n < 10^9$

8 ⇒ 2^3 ⇒ true

98 ⇒ ✗ ⇒ false

1 ⇒ 2^0 ⇒ true

TC ⇒ O(log n)
SC ⇒ O(1)

```
while (2^i < n){
    i++;
}
return 2^i == n;
```

(10) set the rightmost unset bit

n = 6 ⇒ 8 4 2 1 : 0 1 1 0

0 1 1 1 ⇒ 7

n = 16 ⇒ 16 8 4 2 1 : 1 1 1 1 1

1 1 1 1 1 ⇒ 31

TC = O(log n)
SC = O(1)

```
int p = 0;
while( n>>p &1 ==1){
    p++;
}
n = 1<<p & n;
return n;
```

(11) swap 2 numbers $1 \le a, b \le 10^6$

a = 13 b = 9

8 4 2 1

13 → 1 1 0 1

9 → 1 0 0 1

9 → 1 1 0 1

without temp var

```
a = a+b = a|b
b = a-b =
a = a-b
```

```
  1101
- 1001
------
  0100
   100
------
  1101
```

(12) Divide 2 Int without using multiplication, division, mod

$[-2^{31}, 2^{31}-1]$ return quotient

a 13 xor b 9 = S → a = S xor b = 13

S xor 9 = 13 → b

S xor 13 = 9 → b

43 → 43/8 ⇒ larger chunks (multiple of 8) can be subtracted
↳ not allowed so use <<

64 32 16 8 4 2 1

1 0 0 0	<<0	8×1=8	2^0	8<43
1 0 0 0 0	<<1	8×2=16	2^1	16<43
1 0 0 0 0 0	<<2	8×4=32	2^2	32<43
1 0 0 0 0 0 0	<<3	8×8=64	2^3	64>43 stop

quo: $2^2=4$ | divided: 43−32=11 | divisor: 8 | 11>8

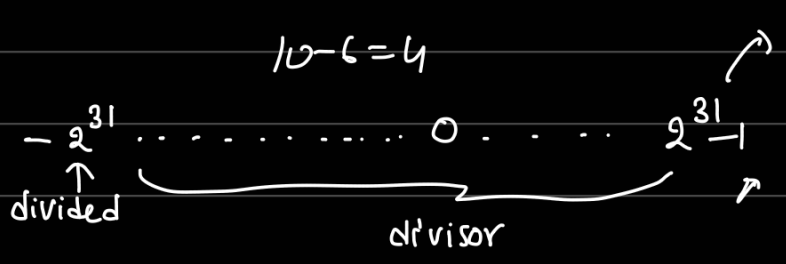
64 32 16 8 4 2 1 (2¹)

0 0 1 1	<<0	3<10 · <4	8<11 $2^0=1$
0 0 1 1	<<1	6<10 >4	16>11 stop
0 0 1 1	<<2	12>10 stop	4

quo: 4+1=5 | 11−8=3 | 8 | 3<8 stop & return

edge case:

10−6=4



-2147483648

-2147483647

2147483647

dividend = 22
divisor = 3

$3 \times 7 = 21$

$3 \times (2^2 + 2^1 + 2^0)$

$(3 \times 2^2) + (3 \times 2^1) + (3 \times 2^0)$

12 + 6 + 3

22 ⇒ $3 \times 2^0 = 3$ → 1
−12 → $3 \times 2^1 = 6$ → 2 } 7
10 → $3 \times 2^2 = 12$ → 4
−6
4
−3
1 < 3

$3 \times 2^3 = 24$

divide_two_integers.cpp

Striver_A2Z_DSA > bit_manipulation > divide_two_integers.cpp > ...

```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution {
public:
    int divide(int dividend, int divisor) {
        // handle overflow case
        if (dividend == INT_MIN && divisor == -1) return INT_MAX;

        // convert to long to prevent overflow
        long a = labs(dividend);
        long b = labs(divisor);
        long q = 0;

        // determine the sign of the result
        bool sign = (dividend < 0) ^ (divisor < 0);  // XOR for sign

        while (a >= b) {
            int i = 0;
            while ((i < 31) && ((b << i) <= a)) {
                i++;
            }
            i--; // go back to the last valid shift
            a -= (b << i);
            q += (1 << i); // add 2^i to quotient
        }

        return sign ? -q : q;
    }
};
```

we can merge 2 loops by iterating from 31 to 0

(13) Find pos of set bit

2 ⇒ 10 ⇒ 1's ⇒ in pos 2 ⇒ 2
5 ⇒ 101 ⇒ 2 1's ⇒ −1

```
i=0 (1<<i)
while (2^i < n){
    i++;
}
if (2^i == n) return i+1;
else return -1;
```

(14) Copy set bits in a range

x y $1 \le l, r \le 32$

10 13 consider [l, r]

l=2 r=3 set in x

4 3 2 1

10 ⇒ 1 0 1 0

13 ⇒ 1 0 0 1

⇒ 1 1 1 0 ⇒ 14

traverse ← count position and take y bit if 1 & keep in x

(15) sq of number without using *, / & pow()

0010	$2^2 \Rightarrow 4$	0100	2+2	
0011	$3^2 \Rightarrow 9$	1001	3+3+3	
0100	$4^2 \Rightarrow 16$	10000	4+4+4+4	
	$5^2 \Rightarrow 25$		5+5+5+5+5	

if n is even

$n = 2*x$

$n^2 = (2*x)^2 = 2^2 * x^2$

$n^2 = 4*x^2$

if n is odd

$n = 2x+1$

$n^2 = (2x+1)^2$

$n^2 = 4*x^2 + 4*x + 1$

$(a+b)^2 = a^2 + b^2 + 2ab$

$(2x+1)^2 = 4x^2 + 1 + 4x = n^2$

n/2 ⇒ [n>>1]

>> /2

<< *2

```
sq (n){
  if (n==0) return 0;
  if (n<0) n=-n;
  int x = n>>1;      [x = n/2]
  if (n&1) return ((sq (x)<<2) + (x<<2) + 1);   ⇒ if [n is odd]
  else return (sq (x) <<2);
}
```

$x^2 * 4$ (*2*2) $x*2+1$

(16) Power set

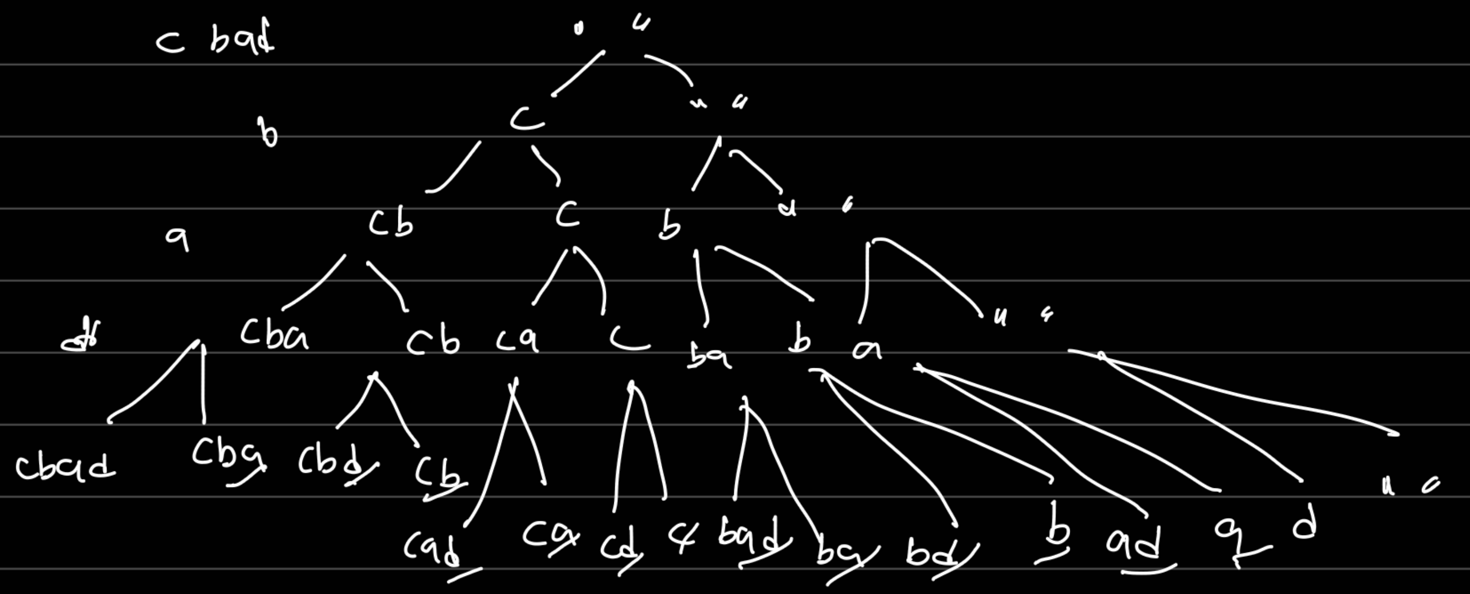
String → len n find all subsequences in lexicographically-sorted order

s = abc ⇒ a ab abc b bc c

cbad

cbad

a ad b ba bad bd c ca cad cb cba cbd cd d



nums [1 2 3] → 3 pos

$2^3 = 8$ powerset → $2^4 = 16$ powerset

No. of subsets = 2^n = (1<<n) ans = [ ]

for(num=0 → subsets-1) { → vector[ ]

  for (j=0 → n-1) {

    if (num & (1<<j)) L push_back(nums[j]) (check num is set or not)

2	1	0	
0	0	0	0 → x
0	0	1	1 → ✓
0	1	0	2
1	0	0	3
0	1	1	4
1	0	0	5
1	0	1	6
1	1	0	7
1	1	1	8

```
public:
    vector<string> AllPossibleStrings(string s){
        // Code here
       //  string n = "";
       //  vector<string> ans = powerset(s, n, 0);
       //  sort(ans.begin(), ans.end());
       //  return ans;
        int n = s.size();
        vector<string> ans{""};
        for(int num = 0; num<(1<<n); num++){
            string v="";
            for(int i=0; i<n; i++){
                if(num&(1<<i)) {
                    v+=s[i];
                }
            }
            ans.push_back(v);
        }
        sort(ans.begin(), ans.end());
        return ans;
    }
};
```

num < 2^n

for every num check set bits & add to V

(17) Find XOR of numbers from L to R

XOR pattern for every 4 numbers

L = 4 R = 8

4 5 6 7 8

XOR of all elements.

4 ⇒	0100	0001 → 6
5 ⇒	0101	0110
6 ⇒	0110	0111
7 ⇒	0111	0000 → 7
8 ⇒	1000	1000 → 8

0	1	2	3
n	1	n+1	0

1	0001	⇒	0011	3	2 ⇒ n+1	6 ⇒ n+1	10 %4 = 2 → n+1
2	0010		0011		3 ⇒ 0	7	4 %4 = 3 → 0
3	0011		0000		4 ⇒ n	8	12 %4 = 0 → n
4	0100		0100		5 ⇒ 1	9	13 %4 = 1 → 1
5	0101		0101				
6	0110		0001				
7			0110				
8			0111				
9							
10							

xor(4) = 0

xor(8) = 0

$n_1 \wedge n_2$

cancel out

we will get from xor of l to r

```
class Solution {
  public:
    int xorfrom1ton(int n){
        if(n%4 == 0) return n;
        else if(n%4 ==1) return 1;
        else if(n%4 == 2) return n+1;
        else if(n%4 == 3) return 0;
    }
    int findXOR(int l, int r) {
        // complete the function here
        return xorfrom1ton(l-1)^xorfrom1ton(r);
    }
};
```

18. Two numbers with odd occurrences

Arr = [4 2 4 5 2 3 3 1] N=8

5 1

1 7 5 7 5 4 7 4

all Numbers xor = x

find right most bit

x & -x

then 2 grps & find 2 numbers

19. Prime factors → given a number N

N=100 out ⇒ 2 5 ⇒ are unique prime factors of 100

N=35 out ⇒ 5 7 ⇒ " " " of 35

100 = 64 32 25

```
vector Primes; int i=2;          TC ⇒ O(N)
While (N>1){                     SC ⇒ O(N)
    if (N%i==0) Primes.push_back(i);
    while (N%i==0) N=N/i;
} i++;
```

20. All divisors of a Number in ascending order.

Inp ⇒ 20 Out ⇒ 1 2 4 5 10 20

TC = √100 = 10 = SC

N/2 Sieve of eratosthenes

```
set <int> S;
for(int i=1; i<=sqrt(n); i++){
    if (n%i==0){                 TC → O(sqrt(N))
        S.insert(i);             SC →
        S.insert(n/i);           O(sqrt(N))
    }
}
for(int i:S){
    cout<<i;
}
```

21. count primes

n = 10 out = 4

2 3 5 7

n=0 out=0

n=1 out=0

Unmark all the multiples of prime NO. or primes < n of composite not prime

TC ⇒ O(n log log n)

SC ⇒ O(n)

```
class Solution {
public:
    int countPrimes(int n) {

        if(n<2) return 0;

        vector<int> primes(n+1, 1);
        int cnt=0;
        //1 - prime, 0 - not prime;
        primes[0] = 0;
        primes[1] = 0; // 1  2 numbers are not prime
        for(int i=2; i<n; i++) {
            if(primes[i]){
                cnt++;
                //  i=2 - 4 6 8 10 12 14 16...
                //  j =   2 3 4 5  6  7  8
                // j*i =  4 6 8 10.............
                for(int j=2; j*i<n; j++){
                    primes[j*i] = 0;
                }
            }
        }
        return cnt;
    }
};
```

22. Prime factorization

using sieve TC ⇒ O(N log(log N))

SC ⇒ O(N)

N = 12246

out ⇒ 2 3 13 157

while applying sieve algo we can check if it dividy the given n? & store it in vector? ⇒ Iterate again precomputed primes & do factorization

(or)

we can also do it in O(N) → in same way Prime factors

23. Power of x^n — find it Implement it

$x = 2.00000 \quad n = 10$

out: 1024.00000

$x \times x \times x \times x \cdot x \cdot x \cdot x \cdot x \cdot x = x^{10}$

$2^{10} \Rightarrow (1010)_2 \quad 2^8 \times 2^2 = 2^{10}$

pow = 1 n = 10 n != 0

1 = 2 10 → even

if (n is odd)
pow * x

x *= x;
n >>= 1; (=) n = n/2

$x = 2 \quad n = 20 \quad 2^{21}$

$2^{21} \Rightarrow 2 \times 2^{20} \quad 2^{-21} \Rightarrow$

$2 \times (4)^{10}$

$2 \times (16)^5$

$2 \times 16 \times 16^4$

$2 \times 16 \times (256)^2$

$2 \times 16 \times (65536)^1$

$2 \times 16 \times 65536 \times (65536)^0$

1

bool sign = n<0;
d any = 1;
while (n > 0) {
if (n%2 == 1) { //odd
any *= x;
n = n-1;
}
else {
x *= x;
n = n/2
}
}
if(sign) return 1/any;
returning any;

$2^{-1} \times 2^{-20} \quad \frac{1}{2^{21}}$

$2^{-1} \times 2$

at last if n<0
return 1/ans

Implement_pow_x_n.cpp

Striver_A2Z_DSA > bit_manipulation > Implement_pow_x_n.cpp > ...

```cpp
#include<bits/stdc++.h>
using namespace std;
class Solution {
public:
    double myPow(double x, int n) {
        double ans = 1;
        bool sign = n<0;
        if(sign){
            while(n<0){
                // if(n%2 == -1){
                if(n&-1 == -1){
                    ans *= x;
                    n = n+1;
                }
                else{
                    x *= x;
                    n >>= 1;// n = n/2;
                }
            }
        }
        else{
            while(n>0){
                // if(n%2 == 1){
                if(n&1){
                    ans *= x;
                    n = n-1;
                }
                else{
                    x *= x;
                    n >>= 1;// n = n/2;
                }
            }
        }
        if(sign) return 1/ans;
        return ans;
    }
};
```