```
// In the name of Allah the merciful.
#include<bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef unsigned long long ull;
typedef vector<int> vi;
typedef vector<ll> vll;
typedef pair<int,int> pii;
typedef pair<ll,ll> pll;

#define endl            '\n'
#define yes             cout << "YES" << endl
#define no              cout << "NO" << endl
#define show(x)         cout << #x << " : " << x << endl
#define showtwo(x, y)   cout << #x << " : " << x << " " << #y << " : " << y << endl
#define all(a)          a.begin(),a.end()
#define input_arr(vec)  for(auto &&it:vec) {cin >> it;}
#define display_arr(vec) for (auto &&it : vec){cout << it << " ";} cout << endl;
#define files           freopen("input.txt", "r", stdin); freopen("output.txt", "w", stdout);
#define efficient()     ios_base::sync_with_stdio(0); cin.tie(0);
#define fraction()      cout.unsetf(ios::floatfield); cout.precision(10); cout.setf(ios::fixed,ios::floatfield);

ll gcd ( ll a, ll b ) { return __gcd ( a, b ); }
ll lcm ( ll a, ll b ) { return abs(a) * ( abs(b) / gcd ( a, b ) ); }

const ll MOD = 1e9 + 7; // 1000000007;
const double PI = acos(-1);

/*-------------------------------------------------------------------------------------------------------*/

int main(void)
{
   //efficient();
   int t = 1;
   cin >> t;
   for (int tc = 1; tc <= t; tc++)
   {
      // cout << "Case " << tc << ": ";
   }
   return 0;
}
```

```
// Conversion of whole String to uppercase or lowercase using STL in C++
transform(s1.begin(), s1.end(), s1.begin(), ::toupper);
transform(s2.begin(), s2.end(), s2.begin(), ::tolower);
```

```
//ASCII Code
A = 65 to Z = 90    |    a =  97 to z = 122    | 0 = 48 to 9 = 57
```

Subarray: (n * (n + 1)) / 2 Subdequence: 2^(n-1) non-emtry Subset: 2^n	Number to character: char c = 4 + '0'; Character to Number: int c = '4' – '0';

// A to the power B using Binary Exponential
// Time complexity O(log n)

Recursive Approch

```
long long binpow(long long a, long long b) {
    if (b == 0)
        return 1;
    long long res = binpow(a, b / 2);
    if (b % 2)
        return res * res * a;
    else
        return res * res;
}
```

Without Recursion

```
long long binpow(long long a, long long b) {
    long long res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a;
        a = a * a;
        b >>= 1;
    }
    return res;
}
```

// A Number is prime or not

```
int is_prime (int n) {
    if (n <= 1) return 0;
    for (int i = 2; i * i <= n; i++)
        if (n % i == 0)
            return 0;
    return 1;
    // ruturn 1 if prime, otherwaise 0
}
```

// Sieve of Eratosthenes

```
void SieveOfEratosthenes(int n)
{
    bool prime[n + 1];
    memset(prime, true, sizeof(prime));

    for (int p = 2; p * p <= n; p++) {
        if (prime[p] == true) {
            for (int i = p * p; i <= n; i += p)
                prime[i] = false;
        }
    }
    // Print all prime numbers
    for (int p = 2; p <= n; p++)
        if (prime[p])
            cout << p << " ";
}
```

```
// All prime factors of a given number
// Prime Factorization without using Sieve --> Time Complexity: O(sqrt(n))

vector<int> primeFactors(int n)
{
   vector<int> primeF;
   while (n % 2 == 0)  {
      primeF.push_back(2);
      n = n/2;
   }
   for (int i = 3; i <= sqrt(n + 1); i = i + 2)
   {
      while (n % i == 0) {
         primeF.push_back(i);
         n = n/i;
      }
   }
   if (n > 2) primeF.push_back(n);
   return primeF;
}
```

```
// Prime Factorization using Sieve O(log n) for multiple queries
// C++ program to find prime factorization of a
// number n in O(Log n) time with precomputation allowed.
#include "bits/stdc++.h"
using namespace std;

#define MAXN 100001
// stores smallest prime factor for every number
int spf[MAXN];

// Calculating SPF (Smallest Prime Factor) for every number till MAXN.
// Time Complexity : O(nloglogn)
void sieve()
{
        spf[1] = 1;
        for (int i = 2; i < MAXN; i++)

                // marking smallest prime factor for every
                // number to be itself.
                spf[i] = i;

        // separately marking spf for every even
        // number as 2
        for (int i = 4; i < MAXN; i += 2)
                spf[i] = 2;

        for (int i = 3; i * i < MAXN; i++) {
                // checking if i is prime
                if (spf[i] == i) {
                        // marking SPF for all numbers divisible by i
```

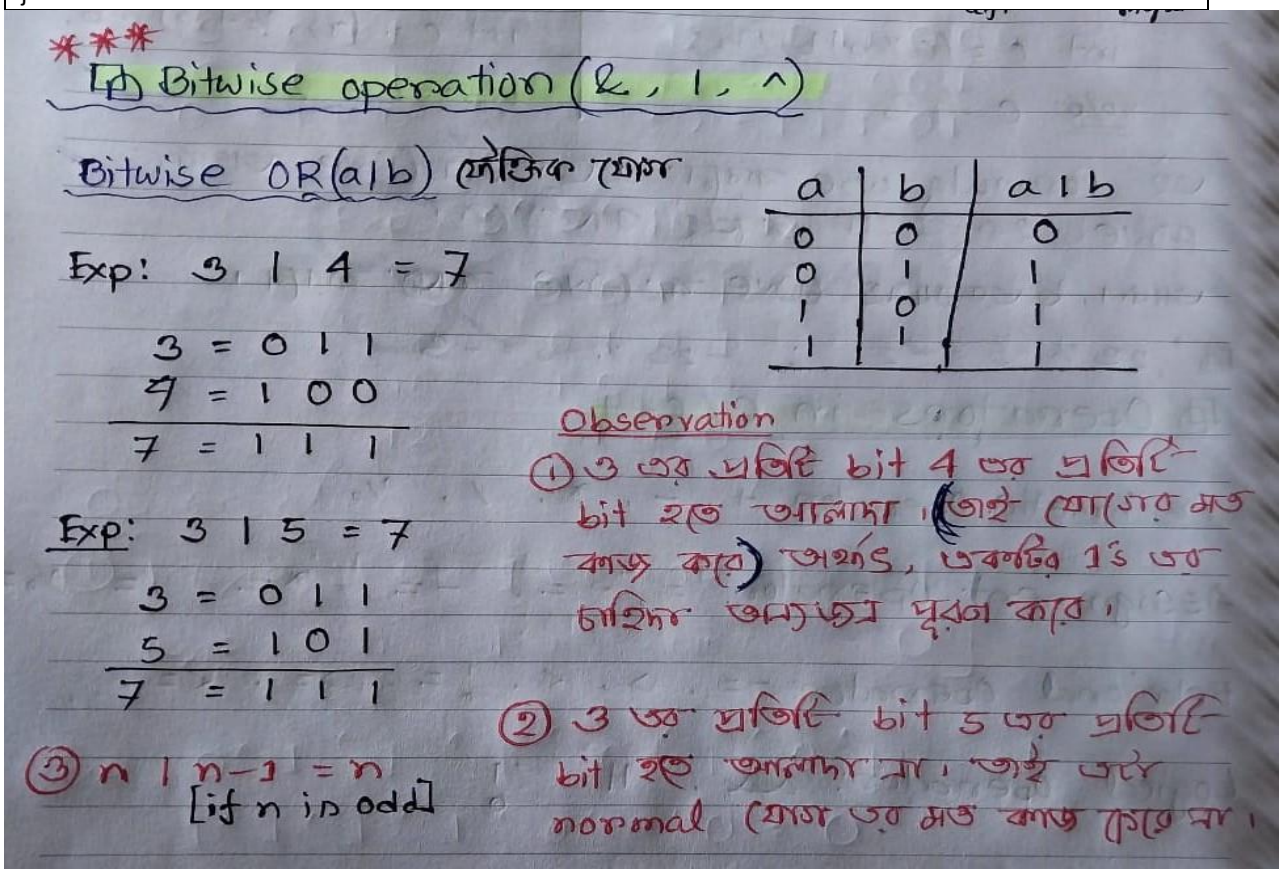
```
                        for (int j = i * i; j < MAXN; j += i)

                                // marking spf[j] if it is not
                                // previously marked
                                if (spf[j] == j)
                                        spf[j] = i;
                }
        }
}

// A O(log n) function returning primefactorization
// by dividing by smallest prime factor at every step
vector<int> getFactorization(int x)
{
        vector<int> ret;
        while (x != 1) {
                ret.push_back(spf[x]);
                x = x / spf[x];
        }
        return ret;
}

// driver program for above function
int main(int argc, char const* argv[])
{
        // precalculating Smallest Prime Factor
        sieve();
        int x = 12246;
        cout << "prime factorization for " << x << " : ";

        // calling getFactorization function
        vector<int> p = getFactorization(x);

        for (int i = 0; i < p.size(); i++) cout << p[i] << " ";
        cout << endl;
        return 0;
}
```

```
// All the possible subsequences
                Solution 1: Using bit manipulation | Time: O((2^n)*n) | Space: O(1)
vector<string> AllPossibleStrings(string s) {
        int n = s.length();
        vector<string>ans;
        for (int num = 0; num < (1 << n); num++) {
                string sub = "";
                for (int i = 0; i < n; i++) {
                        //check if the ith bit is set or not
                        if (num & (1 << i)) {
                                sub += s[i];
                        }
                }
```

```
                if (sub.length() > 0) {
                        ans.push_back(sub);
                }
        }
        sort(ans.begin(), ans.end());
        return ans;
}
```

Solution 2: Using recursion(Backtracking) | Time: O(2^n) | Space: O(n)

```
void solve(int i, string s, string &f) {
        if (i == s.length()) {
                cout << f << " ";
                return;
        }
        f = f + s[i];
        solve(i + 1, s,  f);
        f.pop_back();
        solve(i + 1, s,  f);
}
int main() {
        string s = "abc";
        string f = "";
        cout<<"All possible subsequences are: "<<endl;
        solve(0, s, f);
}
```



Bitwise AND (a&b) যৌক্তিক গুণ

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

Exp: 3 & 4 = 0

```
3 = 0 1 1
4 = 1 0 0
0 = 0 0 0
```

Exp: 3 & 5 = 1

```
3 = 0 1 1
5 = 1 0 1
1 = 0 0 1
```

Exp: 13 & 9 = 9

```
13 = 1101
 9 = 1001
 9 = 1001
```

Exp: 55 & 49 = 49

```
55 = 110111
49 = 110001
49 = 110001
```

Observation

① 3 এর প্রতিটি bit 4 এর প্রতিটি bit হতে আলাদা, অর্থাৎ, একটির ১'s এর সাথে অন্যটি গুণন করে। (তাই Answer always 0.)

② যদি bit আলাদা আলাদা না হয়ে same হয় কিন্তু (কমে অন্তত একটি value. দিবে।

③ কিন্তু, same হওয়ার একটা pattern আছে।

a = 13 ; b = 9 ; এখন (b হচ্ছে a এর আরেকটা subset এর মত) মানে, b এর যে যে bit '1' a এরও ঐ bit গুলাই '1', কিন্তু, a এর যে bit '1' b এর ঐ bit '1' না, এমন হলে always answer = b

④ Same value এর and ই value হয়

Bitwise XOR (a^b)

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

Exp: 3 ^ 4 = 7

```
3 = 0 1 1
4 = 1 0 0
7 = 1 1 1
```

Exp: 3 ^ 5 = 6

```
3 = 0 1 1
5 = 1 0 1
6 = 1 1 0
```

Exp: 12 ^ 17 = 29

```
12 = 0 1 1 0 0
17 = 1 0 0 0 1
29 = 1 1 1 0 1
```

Exp: 3 ^ 5 ^ 10 = 12

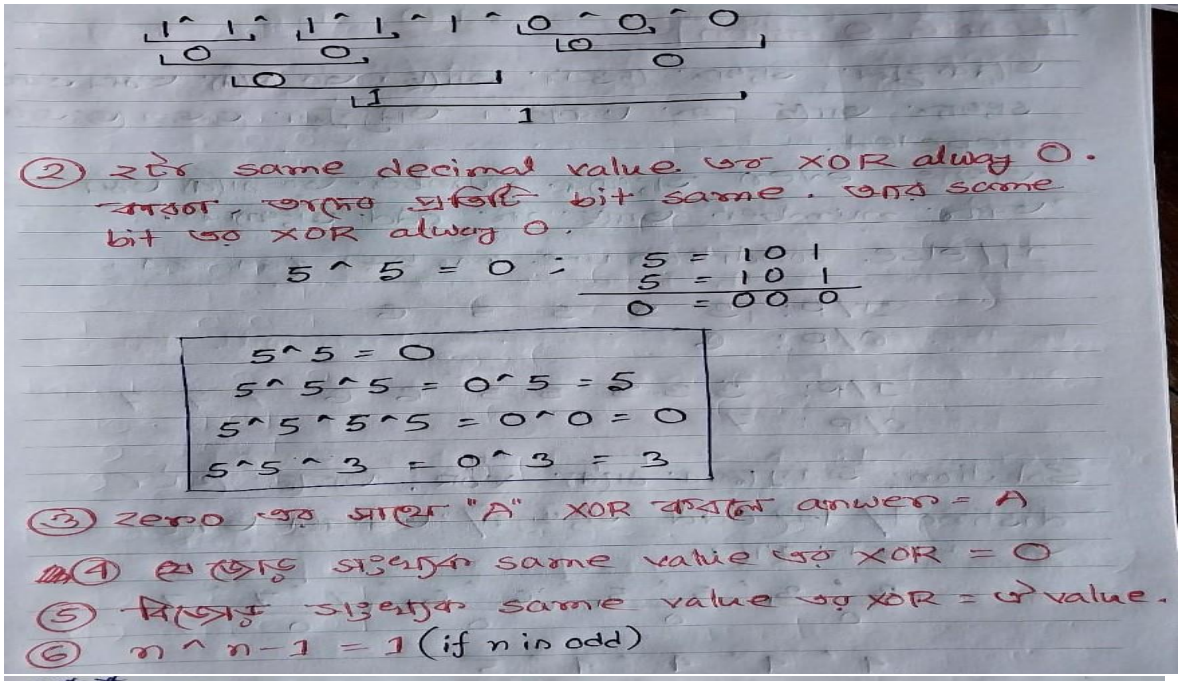
```
 3 = 0 0 1 1
 5 = 0 1 0 1
10 = 1 0 1 0
12 = 1 1 0 0
```

Observation

① বিজোড় সংখ্যক 1 এর XOR always 1 } 1 এর সাথে
জোড় " 1 " XOR " 0 } 0 কখনো আই-
এর matter
করে না।

Exp: 0 ^ 1 ^ 1 ^ 1 ^ 0 ^ 1 ^ 0 ^ 1 = 1

1 ^ 1 ^ 1 ^ 1 ^ 0 ^ 0 ^ 0 = 0



② 2টা same decimal value. এর XOR always 0.
কারণ, তাদের প্রতিটি bit same. তাই same bit এর XOR always 0.

5 ^ 5 = 0 ; 5 = 101
5 = 101
0 = 000

5^5 = 0
5^5^5 = 0^5 = 5
5^5^5^5 = 0^0 = 0
5^5^3 = 0^3 = 3

③ zero এর সাথে "A" XOR করলে answer = A
④ এ জাতীয় সংখ্যক same value এর XOR = 0
⑤ বিজোড় সংখ্যক same value এর XOR = ঐ value.
⑥ n ^ n-1 = 1 (if n is odd)

Tips/Tricks

1 to n পর্যন্ত x দ্বারা Divisable সংখ্যা পাওয়ার জন্য $\frac{n}{x}$.

Ex: 1 to 8 পর্যন্ত 2 এবং 8 দ্বারা Divisable সংখ্যা কয়টি?

$\frac{8}{2} = 4$ টি ; [2, 4, 6, 8]

Exp: 1 to 10^5 পর্যন্ত 3 দ্বারা Divisable সংখ্যা কয়টি?

$\frac{10^5}{3} = 33333$ টি

এ ভাবে কাজ করে,
2 এর ক্ষেত্রে : 0 + 2 + 2 + 2 + ... = 0 2 4 6 + ...
3 এর ক্ষেত্রে : 0 + 3 + 3 + 3 + ... = 0 3 6 9 ...

Tips / Tricks

কোন একটা number এ কয়টা digit আছে তা বের করার Shortcut

$\log_{10}(n) + 1$

Exp: n = 100000 ; $\log_{10}(100000) + 1 = 5 + 1 = 6$

কোন একটা number এর Binary representation এ কতগুলো bit আছে তা বের করার shortcut.

$\log_2(n) + 1$

Exp: n = 100000 ; $\log_2(100000) + 1$
$= 16.609.. + 1$
$= 17.609$
অর্থাৎ, 17 টি Bit আছে।