Contribute Lecture Notes

Lecture Notes

Question 1

Given an array where every element occurs three times, except one element which occurs only once. Find the element that occurs once.

Example:

1. {23, 5, 23, 4, 23, 4, 5, 3, 5, 4} => 3
2. {15, 12, 15, 9, 15, 9, 9} => 12

Solution Idea:

Method 1: Loop through each bit of all the elements and find the sum of bits at each place. If the sum of bits at each position is not divisible by 3, then the required answer has the bit set at that position.

```
#include<bits/stdc++.h>
typedef long long lli;
using namespace std;
lli mod=1e9+7;

int find_single_occurence(vector<int> &arr)
{
    int n=arr.size();
    int ans=0;
    for(int i=0;i<32;i++)   // looping through each digit
    {
```

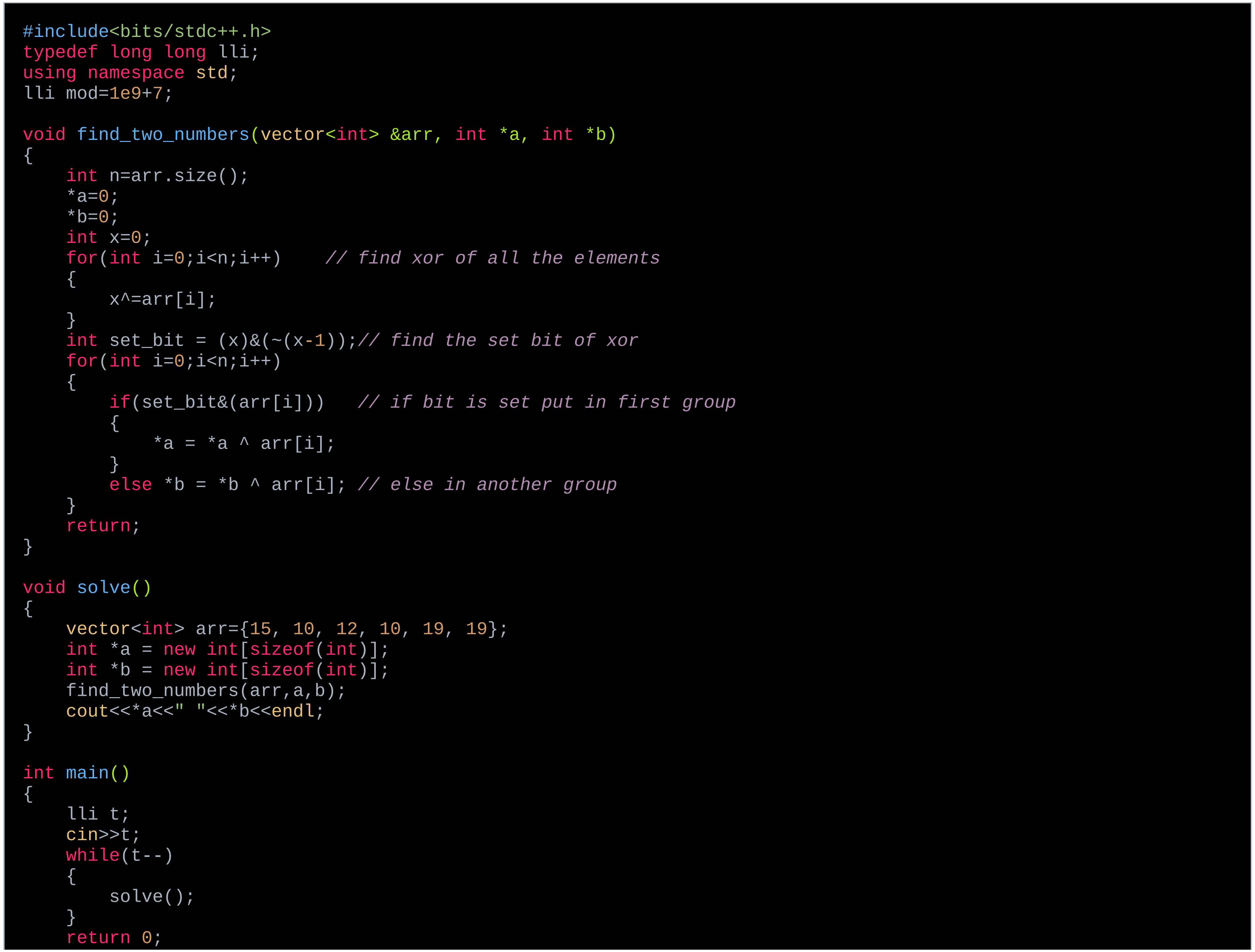
```cpp
#include<bits/stdc++.h>
typedef long long lli;
using namespace std;
lli mod=1e9+7;

int find_odd_occurence(vector<int> &arr)
{
    int n=arr.size();
    int ans=0;
    for(int i=0;i<n;i++)   // looping through each element
    {
        ans = ans^arr[i];  // XOR of all the elements
    }
    return ans;
}

void solve()
{
    vector<int> arr={23, 15, 23, 4, 23, 4, 15, 4, 23, 15, 15};
    int ans = find_odd_occurence(arr);
    cout<<ans<<endl;
}

int main()
{
    lli t;
    cin>>t;
    while(t--)
    {
        solve();
    }
    return 0;
}
```

Question 3

Given an array where every element occurs twice, except two elements which occur only once. Find both the elements which occur only once.

Example:

1. {23, 23, 4, 3, 5, 3, 15, 15} => 4,5
2. {15, 10, 12, 10, 10, 10} => 12,15



```cpp
#include<bits/stdc++.h>
typedef long long lli;
using namespace std;
lli mod=1e9+7;

void find_two_numbers(vector<int> &arr, int *a, int *b)
{
    int n=arr.size();
    *a=0;
    *b=0;
    int x=0;
    for(int i=0;i<n;i++)    // find xor of all the elements
    {
        x^=arr[i];
    }
    int set_bit = (x)&(~(x-1));// find the set bit of xor
    for(int i=0;i<n;i++)
    {
        if(set_bit&(arr[i]))   // if bit is set put in first group
        {
            *a = *a ^ arr[i];
        }
        else *b = *b ^ arr[i]; // else in another group
    }
    return;
}

void solve()
{
    vector<int> arr={15, 10, 12, 10, 19, 19};
    int *a = new int[sizeof(int)];
    int *b = new int[sizeof(int)];
    find_two_numbers(arr,a,b);
    cout<<*a<<" "<<*b<<endl;
}

int main()
{
    lli t;
    cin>>t;
    while(t--)
    {
        solve();
    }
    return 0;
```

```
#include<bits/stdc++.h>
typedef long long lli;
using namespace std;
lli mod=1e9+7;

int find_xor_upto_n(int n)
{
    int rem=n%4;
    switch(rem)
    {
        case 0: return n;
        case 1: return 1;
        case 2: return n+1;
        case 3: return 0;
    }
    return 1;
}
void solve()
{
    int n=14;
    int ans = find_xor_upto_n(n);
    cout<<ans<<endl;
}

int main()
{
    lli t;
    cin>>t;
    while(t--)
    {
        solve();
    }
    return 0;
}
```

Question 5

One number is missing from the list of integers from 1 to n. Find that missing number.

Example:

1. {1, 2, 5, 6, 3, 7} => 4
2. {2, 4, 6, 5, 1} => 3

Given a positive integer n, find the count of positive integers i such that 0 <= i <= n and n+i=n^i.

Example:

1. 10 => 4
2. 18 => 8

Solution Idea:

Method 1: For any integer i if n+i = n^i, then n&i=0. So we will count the number of zero bits in n and calculate how many numbers can be formed using these bits as set bits.

```
#include<bits/stdc++.h>
typedef long long lli;
using namespace std;
lli mod=1e9+7;

int calculate(int n)
{
    int count_unset=0;
    while(n>0)
    {
        if((n&1)==0)    // count number of zero bits
            count_unset++;
        n>>=1;
    }
    return (1<<count_unset);// There can be 2^count numbers
}

void solve()
{
    int n=18;
    int ans = calculate(n);
    cout<<ans<<endl;
}

int main()
{
    lli t;
    cin>>t;
    while(t--)
    {
        solve();
    }
```

```
#include<bits/stdc++.h>
typedef long long lli;
using namespace std;
lli mod=1e9+7;

int find_xor_of_subarrays(vector<int> &arr)
{
    int n=arr.size();
    int ans=0;
    for(int i=0;i<n;i++)
    {
        int freq=(i+1)*(n-i);
        if(freq%2)
        {
            ans = ans^arr[i];
        }
    }
    return ans;
}

void solve()
{
    vector<int> arr={12,15,6,7,9,14,18};
    int ans=find_xor_of_subarrays(arr);
    cout<<ans<<endl;
}

int main()
{
    lli t;
    cin>>t;
    while(t--)
    {
        solve();
    }
    return 0;
}
```

***Method 2:** It can be observed that if the size of the array is even then frequency of each element is even else if number of elements is odd then elements at even position have odd frequency and elements at odd position have even frequency. So elements at odd positions will contribute to the final answer.*

```
#include<bits/stdc++.h>
```

Method 1: *If at any position in binary representation there are x numbers with bit set and y numbers with bit unset then this position will contribute 2*x*y to the answer. Since all bits will act as independent in this question so we will add answers for each bit individually to the final answer.

```cpp
#include<bits/stdc++.h>
typedef long long lli;
using namespace std;
lli mod=1e9+7;

int func(vector<int> &A) {
    int n=A.size();
    int ans=0;
    for(int i=0;i<31;i++)   // loop through each bit
    {
        long long int cnt1=0,cnt2=0;
        for(int j=0;j<n;j++)    // loop through each element
        {
            cnt1+= ( (A[j]&(1<<i)) > 0 ); // count number of set bits
        }
        cnt2=n-cnt1;        // number of unset bits
        cnt1%=mod;
        cnt2%=mod;
        ans = (ans%mod +cnt1*cnt2)%mod;
    }
    return (ans+ans)%mod; // double the answer for every pair
}

void solve()
{
    vector<int> arr={2, 3, 7, 6, 4};
    int ans=func(arr);
    cout<<ans<<endl;
}

int main()
{
    lli t;
    cin>>t;
    while(t--)
    {
        solve();
    }
    return 0;
}
```

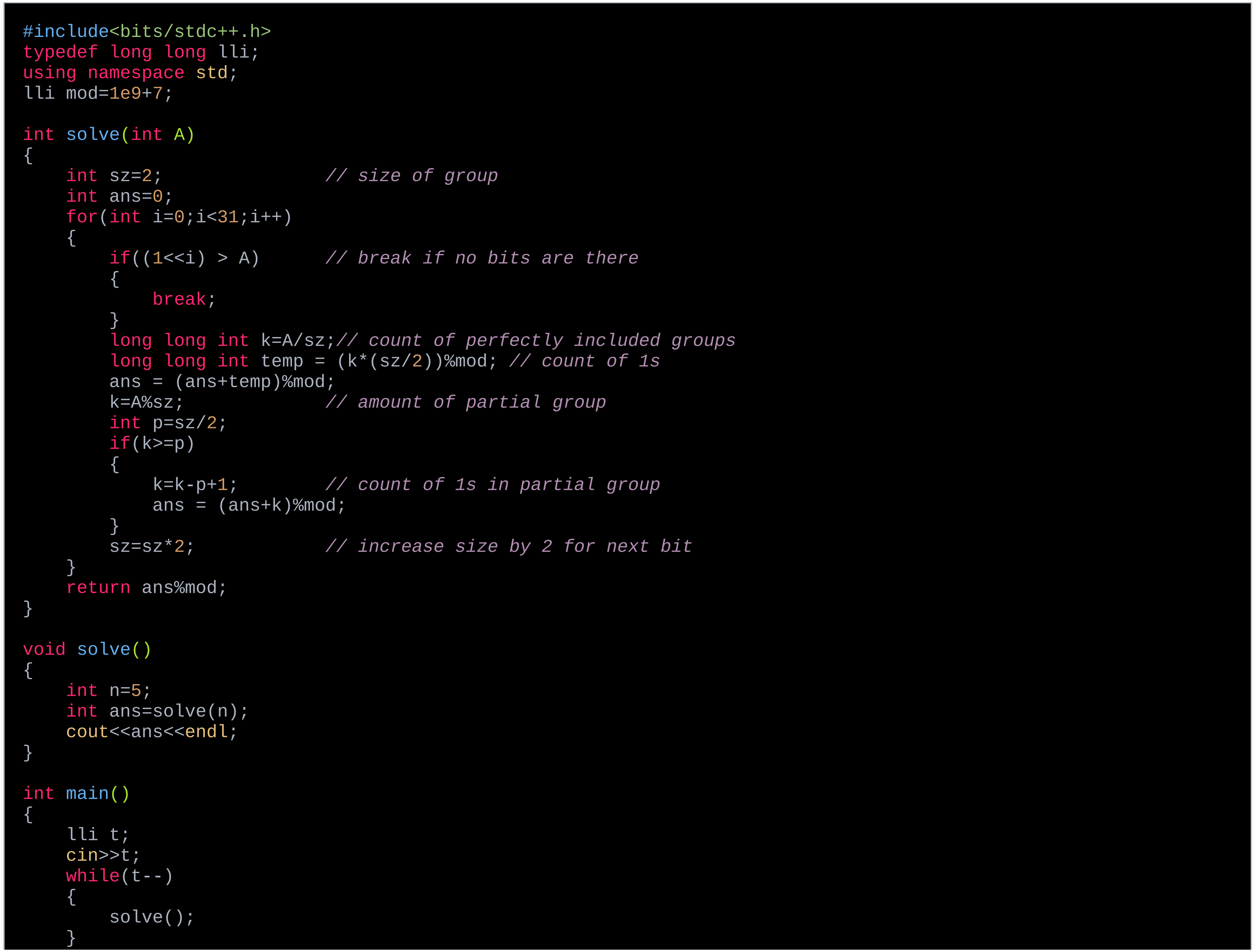
```cpp
#include<bits/stdc++.h>
typedef long long lli;
using namespace std;
lli mod=1e9+7;

int minimum_xor(vector<int> &A) {
    int n=A.size();
    int res=INT_MAX;
    sort(A.begin(),A.end());
    for(int i=1;i<n;i++)
    {
        res=min(res,A[i]^A[i-1]);
    }
    return res;
}

void solve()
{
    vector<int> arr={2, 3, 7, 6, 4};
    int ans=minimum_xor(arr);
    cout<<ans<<endl;
}

int main()
{
    lli t;
    cin>>t;
    while(t--)
    {
        solve();
    }
    return 0;
}
```

Question 10

Find the total number of set bits of all the numbers from 0 to n.

Example:

1. 4 => 5
2. 5 => 7



```cpp
#include<bits/stdc++.h>
typedef long long lli;
using namespace std;
lli mod=1e9+7;

int solve(int A)
{
    int sz=2;                // size of group
    int ans=0;
    for(int i=0;i<31;i++)
    {
        if((1<<i) > A)       // break if no bits are there
        {
            break;
        }
        long long int k=A/sz;// count of perfectly included groups
        long long int temp = (k*(sz/2))%mod; // count of 1s
        ans = (ans+temp)%mod;
        k=A%sz;              // amount of partial group
        int p=sz/2;
        if(k>=p)
        {
            k=k-p+1;         // count of 1s in partial group
            ans = (ans+k)%mod;
        }
        sz=sz*2;             // increase size by 2 for next bit
    }
    return ans%mod;
}

void solve()
{
    int n=5;
    int ans=solve(n);
    cout<<ans<<endl;
}

int main()
{
    lli t;
    cin>>t;
    while(t--)
    {
        solve();
    }
```