



Frequent Questions in Interview

Question 1

Print the binary representation of the given non-zero integer.

Example:

1. 7 => 111

2. 4 => 100



Solution Idea:



Method 1: (Iterative) Consider each bit of unsigned int and check if a bit is set or not and print 1 or 0 accordingly.

```
#include<bits/stdc++.h>
typedef long long lli;
using namespace std;

int main()
{
    unsigned int x = 7;
    for(int i=31;i>=0;i--) // looping through each digit from MSB to LSB
    {
        if(x&(1<<i))      // checking current bit is set or not
        {
            cout << '1';
        }
        else cout << '0';
    }
    cout<<endl;
    return 0;
}
```


[Copy](#)

Method 2: (Recursive) For the given number print all the bits except the LSB(least significant bit) recursively then print the LSB.

```
#include<bits/stdc++.h>
typedef long long lli;
using namespace std;

void printbin(unsigned int x)
{
    if(x>1)
    {
        printbin(x/2);    // recursive call to rest of the bits
    }
    cout<<(x%2);          // print the last bit
}

int main()
{
    unsigned int x = 4;
    printbin(x);
    return 0;
}
```

 Copy

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
    {
```

 Copy

```

    int sum=0;
    for(int j=0;j<n;j++) // looping through each element
    {
        sum+=(arr[j] & (1<<i));
    }
    if(sum%3)
    {
        ans|=(1<<i);    // set the bit in answer
    }
}
return ans;
}

void solve()
{
    vector<int> arr={23, 5, 23, 4, 23, 4, 5, 3, 5, 4};
    int ans = find_single_occurence(arr);
    cout<<ans<<endl;
}

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

```

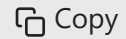
Question 2 Given an array where every element occurs an even number of times, except one element which occurs an odd number of times. Find the element that occurs an odd number of times.

Example:

1. {23, 15, 23, 4, 23, 4, 15, 4, 23, 15, 15} => 4
2. {15, 19, 12, 15, 19, 19, 19} => 12

Solution Idea:

Method 1: Simple way to find the element which occurs an odd number of times is to do XOR of all the elements in the array. Since XOR of two same numbers is zero so XOR of even occurrences of number will be zero.



```
#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, 19, 19} => 12,15

Solution Idea:

Method 1: Suppose two numbers are a and b. First find the XOR of all the elements in the array to obtain $a \oplus b$. Now any bit set in this $a \oplus b$ is set either in a or in b. Divide all the

elements in two groups based on the bit set at the same position of the set bit of $a \oplus b$. In this way a and b will be in two separated groups. Finally find the xor of each group separately to find a and b .

 Copy

```

#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;  
}
```

Question 4 Find the XOR of numbers from 1 to n.

Example:

1. 12 => 12

2. 15 => 0

Solution Idea:

Method 1: The XOR of numbers from 1 to any number just before multiple of 4 is 0 (i.e. 3,7,11).

```
#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--)  
    {
```

[Copy](#)

```
        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

Solution Idea:

Method 1: The simple way to find the missing number is to find the xor of numbers from 1 to n and then find the xor of this answer with the xor of all the elements of the given list.

```
#include<bits/stdc++.h>  
typedef long long lli.  
using namespace std;  
lli mod=1e9+7;  
  
int find_missing_number(vector<int> &arr)  
{  
    int m = arr.size();  
    int n=m+1;  
    int ans=0;  
    for(int i=1;i<=n;i++) // find the xor from 1 to n  
    {  
        ans = ans^i;  
    }  
    for(int i=0;i<m;i++) // find xor from a[0] to a[n-2]  
    {  
        ans = ans^arr[i];  
    }  
    return ans;  
}  
  
void solve()  
{  
    vector<int> arr = {1, 2, 5, 6, 3, 7};  
    int ans = find_missing_number(arr);  
    cout<<ans<<endl;  
}  
  
int main()
```

 Copy

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

Question 6 Given a positive integer n , find the count of positive integers i such that $0 \leq i \leq n$ and $n+i=n^i$.

Example:

1. $10 \Rightarrow 4$
2. $18 \Rightarrow 8$


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;
}
```

 Copy


```
int main()
{
    1

    li t;
    cin>>t;
    while(t--)
    {
        solve();
    }
    return 0;
}
```

Question 7 Given an array of integers, find the xor of elements of all the subarrays possible.

Example:


1. {12,15,6,7,9,14,18} => 17
2. {2,5,6,8,7,12} => 0

Solution Idea:

Method 1: Find the frequency of occurrence of each element in all the subarrays. Only elements whose frequency is odd will contribute to the final answer. The frequency of element at i th index is $(i+1)*(n-i)$.

```
#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;
}
```

 Copy

```

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>
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;
    if(n%2==0)
        return 0;
    for(int i=0;i<n;i+=2)
    {
        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;
}

```

 Copy

```

}

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

```

Question 8 Suppose $f(X,Y)$ = number of positions in binary representation where bits differ. Given an array of integers, find the sum of $f(a_i, a_j)$ for every possible i and j in range 1 to n . Find answer modulo 10^9+7 .

Example:

1. {2, 3, 7, 6, 4} => 32
2. {1,5,12,7,14,16} => 76

Solution Idea:


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 $2xy$ 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.

```

#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;
    }
}

```

 Copy

```

        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;
}

```

Question 9 Given an array of integers, find the minimum xor of a pair of elements of the array.

Example:

1. {2, 3, 7, 6, 4} => 1
2. {2, 5, 4, 2, 6, 8, 9} => 0

Solution Idea:


Method 1: Sort the array in increasing order. Then minimum xor will be xor of a_i and a_{i+1} for some i . This can be easily proved by contradiction.

```

#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;

```

 Copy

```

        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


Solution Idea:

Method 1: It is observed that any bit at i^{th} position from right gets inverted after 2^i numbers. So we can consider a group of size $2^{(i+1)}$ for each i where first 2^i numbers have bit 0 and another 2^i numbers have bit 1 at i^{th} position. Now we will count how many groups are perfectly there up to n , we can count this by dividing n by size of group i.e. $n/(2^{(i+1)})$. Next we will count if any group is partially there up to n , we can count this by taking modulo of n under group size i.e. $n\%(2^{(i+1)})$. If the last partially included group has size more than half of the original group size then only 1 bits will be included in the answer. We can count how many one bits are there.

```

#include<bits/stdc++.h>
typedef long long lli;

```

 Copy

```
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-->0)
    {
        solve();
    }
    return 0;
}
```