Question 1

Problem Statement :

You are given an array a of N Integers and asked to split the array a into k consecutive segments such that each element of a belongs to exactly one segment and the sum of the cost of all segments is minimum.

We define the cost of some segment t as the sum of distances between the first and last occurrence for each unique element in the segment t.

Your task is to find the minimum possible total sum of the cost of all segments.

Input Format

- The first line contains an integer, n, denoting the number of elements in a.
- The next line contains an integer, k, denoting the Number of required consecutive segments..
- Each line i of the n subsequent lines (where 0 <=i<n) contains an integer describing a[i].

Constraints :

- 1 <= n<= 35000
- 1 <=k <=min(n,100)
- 1 <=a[i] <=n

Sample Input	Sample Output	Explanation
1 1 1	0	The only possible segment is [1] The cost is 1-1=0
7 2 1 6 6 4 6 6 6	3	We can divide the array into [1,6,6,4] and [6,6,6] Cost of [1,6,6, 4] will be (1-1)+(3 cost would be 1+2=3

C++

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

using namespace std;
const int N = 4e5, M = 110, inf = 0x3f3f3f3f;
int a[N], dp[N][M], lst[N], pre[N], nxt[N], i, j, n, m, L, R, sum;

int cal(int l, int r) {
    while (L < l) {
        if (nxt[L] <= R)sum -= nxt[L] - L; L++; } while (L > l) {
        --L;
        if (nxt[L] <= R)sum += nxt[L] - L;
    }
    while (R < r) { ++R; if (pre[R] >= L)sum += R - pre[R];
    }
    while (R > r) {
        if (pre[R] >= L)sum -= R - pre[R];
        R--;
    }
    return sum;
}

void solve(int l, int r, int L1, int R1, int now)
{
    if (l > r || L1 > R1)return;
    int mid = (l + r) / 2, val = inf, pos;
    for (int i = L1; i < mid && i <= R1; ++i)
    {
        int tmp = dp[i][now - 1] + cal(i + 1, mid);
        if (tmp < val)pos = i, val = tmp;
    }
    dp[mid][now] = val;
    solve(l, mid - 1, L1, pos, now);
    solve(mid + 1, r, pos, R1, now);
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);
    cin >> n >> m;
    for (i = 1; i <= n; ++i) cin >> a[i];
    for (i = 1; i <= n; ++i) dp[i][0] = inf, pre[i] = lst[a[i]], lst[a[i]] = i;
    for (i = 1; i <= n; ++i) lst[a[i]] = n + 1;
    for (i = n; i; --i) nxt[i] = lst[a[i]], lst[a[i]] = i;
    for (i = 1; i <= m; ++i) solve(1, n, 0, n, i);
    cout << dp[n][m] << "\n";
}
```

Question 2

Problem Statement : Wael wants to play Santa this year, so he prepared gifts for all the children of the neighborhood. He decided to pack the gifts in boxes and give each child a box. Let's define the Value of the box as the number of distinct types of gifts inside this box.

Wael has N gifts, such that the type of each gift i is A[i]. Wael wants to pack exactly K boxes, and he has to put in each box a sub-array of consecutive gifts

Wael wants to maximize the total value of all boxes with gifts. Your task is to help him determine this maximum possible total value

Notes

- Each gift should be put in exactly one box, and each box should contain a sub-array of consecutive gifts.
- A box cannot be left empty

Input Format

- The first line contains an integer, N, denoting the number of elements in A
- The next line contains an integer, K, denoting the number of boxes.
- Each line i of the N subsequent lines (where 0≤i<N) contains an integer describing Ai

Constraints

- 1 <= N <= 35000
- 1 <= K <= Min (N,50)
- 1 <= A[i] <= N

Sample Input	Sample Output	Explanation
1 1 1	1	Wael will put the only gift in a box so the total value will be 1.
4 1 1 1 2	2	Wael has only one box he has to put all gifts in it, so that there are two types of gifts

Sample Input	Sample Output	Explanation
2 1		
7 2 1 3 3 1 4 4 4	5	It is optimal to put the first two gifts in the first box, and all the rest in the second Th the second box then, so the total value is 5.

C++

```
#include <bits/stdc++.h>
using namespace std;
int ans = INT_MIN;

void solve(int a[], int n, int k, int index, int count, int maxval)
{

   if (k == 1) {
      maxval = max(maxval, count);
      count = 0;
      map<int, int>mp;
      for (int i = index; i < n; i++) {
         mp[a[i]]++;
      }
      count = mp.size();
      mp.clear();

      maxval = max(maxval, count);

      ans = max(ans, maxval);
      return;
   }
   count = 0;
      map<int, int>mp;
      for (int i = index; i < n; i++) {
         mp[a[i]]++;
         count = mp.size();


     maxval = max(maxval, count);

     solve(a, n, k - 1, i + 1, count, maxval);

   }
}
// Driver Code
```

```
int main()
{
   int arr[] = {1, 1};
   int k = 2; // K divisions
   int n = 2; // Size of Array
   solve(arr, n, k, 0, 0, 0);
   cout << ans << "\n";
}
```

Question 3

Problem Statement :

A subarray of array A is a segment of contiguous elements in array A.

Given an array A of N elements, you can apply the following operations as many times as you like:

– Choosing a subarray [L, R] and subtracting 1 from each element in this subarray. The cost of this operation is X.

– Choosing an index i such that A[i] is positive, and setting A[i] = 0. The cost of this operation in Y.

Your task is to make all the elements equal to 0 and find the minimum cost to do so.

Input Format

- The first line contains an integer, N., denoting the number of elements in A.
- The next line contains an integer, X, denoting the cost of the first operation.
- The next line contains an integer. Y, denoting the cost of the second operation
- Each line i of the N subsequent lines (where 1 <=i<= N) contains an Integer describing Ai.

Constraints

- 1<=N<=10^5
- 1<=X<=10
- 1<=Y<=10^4
- 1<=A[i]<=10^8

Sample Input 1

1

1

10

1

Sample Output 1

1

Explanation:

N=1 X=1 Y=10 A=[1]. The optimal solution is to perform one operation of the first type on the subarray [1,N].

Sample Input 2

3

1

1

1

1

1

Sample Output 2

1

Explanation:

N=3 X=1 Y=1 A=[1,1,1] The optimal solution is to perform one operation of the first type on the subarray[1,N];

C++
Java
Python

```cpp
#include <iostream>
using namespace std;
int main ()
{
 int arr[] = { 1, 1, 1 };

 int X = 1;
 int Y = 1;
 int ans = 0;
 int arrSize = sizeof (arr) / sizeof (arr[0]);

 for (int i = 0; i < arrSize; i++)
  {
   arr[i]--;
  }
 ans = ans + X;

 for (int i = 0; i < arrSize; i++)
  {
   if (arr[i] != 0)
                {
                 arr[i] = 0;
                ans = ans + Y;
              }
  }
 cout << ans;
 return 0;
}
```

Question 4

Problem Statement :

Given an array A of N elements. You should choose a value B such that (B>=0), and then for each element in A set A[i]=A[i](+)B where is the bitwise XOR.
Print the minimum number of inversions in array A that you can achieve after choosing the value of B optimally and setting A[i] = A[i] (+) B. Since the answer might be large, print it modulo (10^9+7)

Input Format

- The first line contains an integer, N. denoting the number of elements in A
- Then the next line contains N elements, denoting the elements in A.

Input :

4

1 0 3 2

Output

1

C++

```
#include <bits/stdc++.h>
#define sz(x) ((int)(x).size())
#define inf mod

using namespace std;

const int maxn = (int) 5e6 + 100;
int n, t[2][maxn], id = 1;
int dp[2][30];
vector < int >g[maxn];

void add (int x, int pos)
{
  int v = 0;
  for (int i = 29; i >= 0; i--)
```

```
    {
      int bit = ((x >> i) & 1);
      if (!t[bit][v])
                t[bit][v] = id++;
      v = t[bit][v];
      g[v].push_back (pos);
    }
}

void go (int v, int b = 29)
{
  int l = t[0][v], r = t[1][v];
  if (l)
    go (l, b - 1);
  if (r)
    go (r, b - 1);
  if (!l || !r)
    return;
  int res = 0;
  int ptr = 0;
for (auto x:g[l])
    {
      while (ptr < sz (g[r]) && g[r][ptr] < x)
                ptr++;
      res += ptr;
    }
  dp[0][b] += res;
  dp[1][b] += sz (g[l]) * 1ll * sz (g[r]) - res;
}


int main ()
{
  cin >> n;
  for (int i = 1; i <= n; i++)
    {
      int x;
      cin >> x;
      add (x, i);
    }
  go (0);
  int inv = 0;
  int res = 0;
  for (int i = 0; i <= 29; i++)
    {
      inv += min (dp[0][i], dp[1][i]);
      if (dp[1][i] < dp[0][i])
                res += (1 << i);
    }
  cout << inv;
}
```

Question 5

Problem Statement :

Wael is well-known for how much he loves the bitwise XOR operation, while kaito is well known for how much he loves to sum numbers, so their friend Resli decided to make up a problem that would enjoy both of them. Resil wrote down an array A of length N, an integer K and he defined a new function called Xor- sum as follows

- Xor-sum(x)=(x XOR A[1])+(x XOR A[2])+(x XOR A[3])+……..+(x XOR A[N])

Can you find the integer x in the range [0,K] with the maximum Xor-sum (x) value?

Print only the value.

Input format

- The first line contains integer N denoting the number of elements in A.
- The next line contains an integer, k, denoting the maximum value of x.
- Each line i of the N subsequent lines(where 0<=i<=N) contains an integer describing Ai.

Constraints

- 1<=N<=10^5
- 0<=K<=10^9
- 0<=A[i]<=10^9

Sample Input 1

1

0

989898

Sample Output 1

989898

Explanation:

Xor_sum(0)=(0^989898)=989898

Sample Input 2

3
7
1
6
3

Sample Output 2

14

Explanation

Xor_sum(4)=(4^1)+(4^6)+(4^3)=14.

Sample Input 3

4
9
7
4
0
3

Sample Output 3

46

Explanation:

Xor_sum(8)=(8^7)+(8^4) +(8^0)+(8^3)=46.

C++
Java
Python

```
#include<bits/stdc++.h>
using namespace std;
unordered_map < int, int >L;

int main ()
{

 int n, k, m;
 cin >> n >> k;
 vector < int >v (n);

 for (int i = 0; i < n; i++) { cin >> m;
    v[i] = m;
    int j = 0;
    while (m)
              {
                L[j] += (m & 1);
                m >>= 1;
                j++;
              }
  }

 int j = 0, K = k, ans = 0, ans2 = 0;
 while (K)
  {
   j++;
   K >>= 1;
  }

 for (int i = j; i > 0; i--)
  {
   if (L[i - 1] < n - L[i - 1])
                 ans != 1;
   ans <<= 1; } ans >>= 1;
 while (ans > k)
  {
   ans &= 0;
   ans <<= 1;
   k <<= 1;
  }

for (auto i:v)
   ans2 += ans ^ i;
```

```
  cout << ans2;

}
```