```
// Chef And Easy Xor Queries (CodeChef)
// Sqrt Decomposition

// Operations:
// 1 : Update value x at pos i
// 2 : Find subarray XOR of value k from index 1 to r (All Subarray starts from 1)

// Approach:
// 1 : All segment consecutive xor is calculated in Seg aray
//   : All segment consecutive xor is also counted on SegMap
// 2 : Updates are done on each Decomposed segment array
// 3 : Queries are combined from all Decomposed array in the range

int BlockSize, Seg[1010][1010];                  // BlockSize is the size of each Block
int SegMap[330][1110007] = {0};

void Update(int v[], int l, int val) {           // Updates value in position l : val
   int idx = l/BlockSize;                        // Block Index
   int lft = (l/BlockSize)*BlockSize;            // The leftmost index of array v, which is the 0 position
                                                 //of Segment idx
   v[l] = val;                                   // Setting value to default array to ease

   // Clear full block and re-calculate          // Using memset in large array will cause TLE
   SegMap[idx][Seg[idx][0]]--;                   // Decreasing previous value
   Seg[idx][0] = v[lft];
   SegMap[idx][v[lft++]]++;                      // Increasing with new value
   for(int i = 1; i < BlockSize; ++i, ++lft) {
      SegMap[idx][Seg[idx][i]]--;
      Seg[idx][i] = Seg[idx][i-1] ^ v[lft];
      SegMap[idx][Seg[idx][i]]++;
   }
}

int Query(int l, int r, int k) {                 // Query in range l -- r for k
   int Count = 0, val = 0;
   while(l%BlockSize != 0 && l < r) {            // if l partially lies inside of a sqrt segment
      cout << "P1" << endl;
      Count += (Seg[l/BlockSize][l%BlockSize] == k);
      val = val^Seg[l/BlockSize][l%BlockSize];
      ++l;
   }
   while(l+BlockSize <= r) {                     // for all full sqrt segment
      Count += SegMap[l/BlockSize][k^val];
      val ^= Seg[l/BlockSize][BlockSize-1];
      l += BlockSize;
   }
   while(l <= r) {                               // for the rightmost partial sqrt segment values
      Count += (Seg[l/BlockSize][l%BlockSize] == (k^val));
      ++l;
   }

   return Count;
}

void SqrtDecompose(int v[], int len) {           // Builds Sqrt segments
   int idx, pos, val = 0;
   BlockSize = sqrt(len);                        // Calculating Block size
   for(int i = 0; i < len; ++i) {
      idx = i/BlockSize;                         // Index of block
```

```
        pos = i%BlockSize;                          // Index of block element
        if(pos == 0)
            val = 0;
        val ^= v[i];
        Seg[idx][pos] = val;
        SegMap[idx][val]++;
    }
}

int v[100100];

int main() {
    int n, q, idx, x, t;
    sf("%d %d", &n, &q);
    for(int i = 0; i < n; ++i)
        sf("%d", &v[i]);

    SqrtDecompose(v, n);

    while(q--) {
        sf("%d", &t);
        if(t == 1) {
            sf("%d %d", &idx, &x);
            Update(v, idx-1, x);
        }
        else {
            sf("%d %d", &idx, &x);
            pf("%d\n", Query(0, idx-1, x));
        }
    }

    return 0;
}
```