Faculty of Computer and Information Sciences, Ain Shams University: Too Wrong to Pass Too Correct to Fail

Pillow, Isaac, Mostafa, Islam

2021

Contents

1 Combinatorics

1.1 Burnside Lemma

```

// |Classes|=sum (k ^C(pi))  / |G|

// C(pi)  the number of cycles in the permutation pi

// |G| the number of permutations
```

1.2 Catlan Numbers

```
const int MOD = ....
const int MAX = ....
int catalan[MAX];
void init() {
    catalan[0] = catalan[1] = 1;
    for (int i=2; i<=n; i++) {
        catalan[i] = 0;
        for (int j=0; j < i; j++) {
            catalan[i] += (catalan[j] * catalan[i-j-1]) % MOD;
            if (catalan[i] >= MOD) {
                catalan[i] -= MOD;
            }
        }
    }
}

// 1- Number of correct bracket sequence consisting of n opening and n closing brackets.
// 2- The number of rooted full binary trees with n+1 leaves (vertices are not numbered).
//    A rooted binary tree is full if every vertex has either two children or no children.
// 3- The number of ways to completely parenthesize n+1 factors.
// 4- The number of triangulations of a convex polygon with n+2 sides
//    (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
// 5- The number of ways to connect the 2n points on a circle to form n disjoint chords.
// 6- The number of non-isomorphic full binary trees with n internal nodes (i.e. nodes having at least one son).
// 7- The number of monotonic lattice paths from point (0,0) to point (n,n) in a square lattice of size nxn,
//    which do not pass above the main diagonal (i.e. connecting (0,0) to (n,n)).
// 8- Number of permutations of length n that can be stack sorted
//    (i.e. it can be shown that the rearrangement is stack sorted if and only if
//    there is no such index i<j<k, such that ak<ai<aj ).
// 9- The number of non-crossing partitions of a set of n elements.
// 10- The number of ways to cover the ladder 1..n using n rectangles
// (The ladder consists of n columns, where ith column has a height i).
```

2 Algebra

2.1 Primitive Roots

```
int powmod (int a, int b, int p) {
    int res = 1;
    while (b)
        if (b & 1)
            res = int (res * 1ll * a % p),  --b;
        else
            a = int (a * 1ll * a % p),  b >>= 1;
    return res;
}

int generator (int p) {
    vector<int> fact;
```

```cpp
    int phi = p - 1,  n = phi;
    for (int i = 2; i * i <= n; ++i)
        if (n % i == 0) {
            fact.push_back (i);
            while (n % i == 0)
                n /= i;
        }
    if (n > 1)
        fact.push_back (n);

    for (int res = 2; res <= p; ++res) {
        bool ok = true;
        for (size_t i = 0; i < fact.size() && ok; ++i)
            ok &= powmod (res, phi / fact[i], p) != 1;
        if (ok)  return res;
    }
    return -1;
}
```

2.2 Discrete Logarithm

```cpp
// Returns minimum x for which a ^ x % m = b % m, a and m are coprime.
int solve(int a, int b, int m) {
    a %= m, b %= m;
    int n = sqrt(m) + 1;

    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;

    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = (cur * 1ll * a) % m;
    }

    for (int p = 1, cur = 1; p <= n; ++p) {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur)) {
            int ans = n * p - vals[cur];
            return ans;
        }
    }
    return -1;
}

//When a and m are not coprime
// Returns minimum x for which a ^ x % m = b % m.
int solve(int a, int b, int m) {
    a %= m, b %= m;
    int k = 1, add = 0, g;
    while ((g = gcd(a, m)) > 1) {
        if (b == k)
            return add;
        if (b % g)
            return -1;
        b /= g, m /= g, ++add;
        k = (k * 1ll * a / g) % m;
    }

    int n = sqrt(m) + 1;
    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;

    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = (cur * 1ll * a) % m;
    }

    for (int p = 1, cur = k; p <= n; ++p) {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur)) {
            int ans = n * p - vals[cur] + add;
            return ans;
        }
    }
    return -1;
}
```

2.3 Iteration over submasks

```cpp
int s = m;
while (s > 0) {
 ... you can use s ...
 s = (s-1) & m;
}
```

2.4 Totient function

```cpp
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    phi[0] = 0;
    phi[1] = 1;
    for (int i = 2; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

2.5 CRT and EEGCD

```cpp
ll extended(ll a, ll b, ll &x, ll &y) {

    if(b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    ll x0, y0;
    ll g = extended(b, a % b, x0, y0);
    x = y0;
    y = x0 - a / b * y0;

    return g ;
}
ll de(ll a, ll b, ll c, ll &x, ll &y) {

    ll g = extended(abs(a), abs(b), x, y);
    if(c % g) return -1;

    x *= c / g;
    y *= c / g;

    if(a < 0)x = -x;
    if(b < 0)y = -y;
    return g;
}
pair<ll, ll> CRT(vector<ll> r, vector<ll> m) {

    ll r1 = r[0], m1 = m[0];

    for(int i = 1; i < r.size(); i++) {

        ll r2 = r[i], m2 = m[i];
        ll x0, y0;
        ll g = de(m1, -m2, r2 - r1, x0, y0);

        if(g == -1) return {-1, -1} ;

        ll nr = x0 * m1 + r1;
        ll nm = m1 / g * m2;

        r1 = (nr % nm + nm) % nm;
        m1 = nm;
    }
    return {r1, m1};
}
```

2.6 FFT

```cpp
#include<iostream>
#include <bits/stdc++.h>
#define ll long long
#define ld long double
#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
#define IO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
using namespace std;
typedef complex<double> C;
typedef vector<double> vd;
typedef vector<int> vi;
typedef pair<int, int> pii;
void fft(vector<C>& a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, 1);
    static vector<C> rt(2, 1); // (^ 10% fas te r i f double)
    for (static int k = 2; k < n; k *= 2) {
        R.resize(n);
        rt.resize(n);
        auto x = polar(1.0L, acos(-1.0L) / k);
        rep(i, k, 2 * k) rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
    }
    vi rev(n);
    rep(i, 0, n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i, 0, n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) rep(j, 0, k) {
            C z = rt[j + k] * a[i + j + k]; //
            a[i + j + k] = a[i + j] - z;
            a[i + j] += z;
        }
}
vd conv(const vd& a, const vd& b) {
    if (a.empty() || b.empty()) return {};
    vd res(sz(a) + sz(b) - 1);
    int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
    vector<C> in(n), out(n);
    copy(all(a), begin(in));
    rep(i, 0, sz(b)) in[i].imag(b[i]);
    fft(in);
    for (C& x : in) x *= x;
    rep(i, 0, n) out[i] = in[-i & (n - 1)] - conj(in[i]);
    fft(out);
    rep(i, 0, sz(res)) res[i] = imag(out[i]) / (4 * n);
    return res;
}

int main() {
    IO
    //Applications
    //1-All possible sums

    //2-All possible scalar products
    // We are given two arrays a[] and b[] of length n.
    //We have to compute the products of a with every cyclic shift of b.
    //We generate two new arrays of size 2n: We reverse a and append n zeros to it.
    //And we just append b to itself. When we multiply these two arrays as polynomials,
    //and look at the coefficients c[n-1], c[n], ..., c[2n-2] of the product c, we get:
    //c[k]=sum i+j=k  a[i]b[j]

    //3-Two stripes
    //We are given two Boolean stripes (cyclic arrays of values 0 and 1) a and b.
    //We want to find all ways to attach the first stripe to the second one,
    //such that at no position we have a 1 of the first stripe next to a 1 of the second stripe.
}
```

2.7 Fibonacci

```cpp


// F(n-1) * F(n+1) - F(n)^2 = (-1)^n

// F(n+k) = F(k) * F(n+1) + F(k-1) * F(n)

// F(2*n) = F(n) * (F(n+1) + F(n-1))

//GCD ( F(m) , F(n) ) = F(GCD(n,m))
```

2.8 Gauss Determinant

```cpp
const double EPS = 1E-9;
int n;
vector < vector<double> > a (n, vector<double> (n));

double det = 1;
for (int i=0; i<n; ++i) {
    int k = i;
    for (int j=i+1; j<n; ++j)
        if (abs (a[j][i]) > abs (a[k][i]))
            k = j;
    if (abs (a[k][i]) < EPS) {
        det = 0;
        break;
    }
    swap (a[i], a[k]);
    if (i != k)
        det = -det;
    det *= a[i][i];
    for (int j=i+1; j<n; ++j)
        a[i][j] /= a[i][i];
    for (int j=0; j<n; ++j)
        if (j != i && abs (a[j][i]) > EPS)
            for (int k=i+1; k<n; ++k)
                a[j][k] -= a[i][k] * a[j][i];
}

cout << det;
```

2.9 GAUSS SLAE

```cpp
const double EPS = 1e-9;
const int INF = 2; // it doesn't actually have to be infinity or a big number

int gauss (vector < vector<double> > a, vector<double> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where (m, -1);
    for (int col = 0, row = 0; col < m && row < n; ++col) {
        int sel = row;
        for (int i = row; i < n; ++i)
            if (abs (a[i][col]) > abs (a[sel][col]))
                sel = i;
        if (abs (a[sel][col]) < EPS)
            continue;
        for (int i = col; i <= m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;

        for (int i = 0; i < n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j = col; j <= m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }

    ans.assign (m, 0);
    for (int i = 0; i < m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i = 0; i < n; ++i) {
        double sum = 0;
        for (int j = 0; j < m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
            return 0;
    }

    for (int i = 0; i < m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}
```

2.10 Matrix Inverse

```cpp
// Sometimes, the questions are complicated - and the answers are simple. //
#pragma GCC optimize ("O3")
#pragma GCC optimize ("unroll-loops")
#include <bits/stdc++.h>
#define ll long long
#define ld  long double
#define IO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
using namespace std;
vector < vector<double> > gauss (vector < vector<double> > a) {

    int n = (int) a.size();
    vector<vector<double> > ans(n, vector<double>(n, 0));

    for(int i = 0; i < n; i++)
        ans[i][i] = 1;
    for(int i = 0; i < n; i++) {
        for(int j = i + 1; j < n; j++)
            if(a[j][i] > a[i][i]) {
                swap(a[j], a[i]);
                swap(ans[j], ans[i]);
            }
        double val = a[i][i];
        for(int j = 0; j < n; j++) {
            a[i][j] /= val;
            ans[i][j] /= val;
        }
        for(int j = 0; j < n; j++) {
            if(j == i)continue;
            val = a[j][i];
            for(int k = 0; k < n; k++) {
                a[j][k] -= val * a[i][k];
                ans[j][k] -= val * ans[i][k];
            }
        }
    }
    return ans;
}
int main() {

    IO
    vector<vector<double> > v(3, vector<double> (3) );
    for(int i = 0; i < 3; i++)
        for(int j = 0; j < 3; j++)
            cin >> v[i][j];

    for(auto i : gauss(v)) {
        for(auto j : i)
            cout << j << " ";
        cout << "\n";
    }
}
```

2.11 NTT

```cpp
struct NTT {
    int mod ;
    int root ;
    int root_1 ;
    int root_pw ;

    NTT(int _mod, int primtive_root, int NTT_Len) {

        mod = _mod;
        root_pw = NTT_Len;
        root = fastpower(primtive_root, (mod - 1) / root_pw);
        root_1 = fastpower(root, mod - 2);
    }
    void fft(vector<int> & a, bool invert) {
        int n = a.size();

        for (int i = 1, j = 0; i < n; i++) {
            int bit = n >> 1;
            for (; j & bit; bit >>= 1)
                j ^= bit;
            j ^= bit;

            if (i < j)
                swap(a[i], a[j]);
        }

        for (int len = 2; len <= n; len <<= 1) {
            int wlen = invert ? root_1 : root;
            for (int i = len; i < root_pw; i <<= 1)
                wlen = (int)(1LL * wlen * wlen % mod);


            for (int i = 0; i < n; i += len) {
                int w = 1;
                for (int j = 0; j < len / 2; j++) {
                    int u = a[i + j], v = (int)(1LL * a[i + j + len / 2] * w %
                        mod);
                    a[i + j] = u + v < mod ? u + v : u + v - mod;
                    a[i + j + len / 2] = u - v >= 0 ? u - v : u - v + mod;
                    w = (int)(1LL * w * wlen % mod);
                }
            }
        }

        if (invert) {
            int n_1 = fastpower(n, mod - 2);
            for (int & x : a)
                x = (int)(1LL * x * n_1 % mod);
        }
    }
    vector<int> multiply(vector<int> &a, vector<int> &b) {
        vector<int> fa(a.begin(), a.end()), fb(b.begin(), b.end());
        int n = 1;
        while(n < a.size() + b.size())
            n <<= 1;

        fa.resize(n);
        fb.resize(n);

        fft(fa, 0);
        fft(fb, 0);

        for(int i = 0; i < n; i++)
            fa[i] = 1LL * fa[i] * fb[i] % mod;
        fft(fa, 1);
        return fa;
    }
};
```

2.12 NTT of KACTL

```
///(Note faster than the other NTT)
///If the mod changes don't forget to calculate the primitive root
using ll = long long;
const ll mod = (119 << 23) + 1, root = 3; // = 998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
// and 483 << 21 (same root). The last two are > 10^9.
typedef vector<ll> vl;

ll modpow(ll b, ll e) {
    ll ans = 1;
    for (; e; b = b * b % mod, e /= 2)
        if (e & 1) ans = ans * b % mod;
    return ans;
}
void ntt(vl &a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vl rt(2, 1);
    for (static int k = 2, s = 2; k < n; k *= 2, s++) {
        rt.resize(n);
        ll z[] = {1, modpow(root, mod >> s)};
        f(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
    }
    vector<int> rev(n);
    f(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    f(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) f(j,0,k) {
            ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
            a[i + j + k] = ai - z + (z > ai ? mod : 0);
            ai += (ai + z >= mod ? z - mod : z);
        }
}
vl conv(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s), n = 1 << B;
    int inv = modpow(n, mod - 2);
    vl L(a), R(b), out(n);
    L.resize(n), R.resize(n);
    ntt(L), ntt(R);
    f(i,0,n) out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
    ntt(out);
    return {out.begin(), out.begin() + s};
}
vector<int> v;
vector<ll> solve(int s, int e) {
    if(s==e) {
        vector<ll> res(2);
        res[0] = 1;
        res[1] = v[s];
        return res;
    }
    int md = (s + e) >> 1;
    return conv(solve(s,md),solve(md+1,e));
}
```

3 Data Structures

3.1 2D BIT

```
void upd(int x, int y, int val) {
    for(int i = x; i <= n; i += i & -i)
    for(int j = y; j <= m; j += j & -j)
    bit[i][j] += val;
}
int get(int x, int y) {
    int ans = 0;
    for(int i = x; i; i -= i & -i)
    for(int j = y; j; j -= j & -j)
    ans += bit[i][j];
}
```

3.2 2D Sparse table

```
/*
    note this isn't the best cache-wise version
    query O(1), Build O(NMlgNlgM)
    be careful when using it and note the he build a dimension above another
    i.e he builds a sparse table for each row
    the build sparse table over each row's sparse table
*/
const int N = 505, LG = 10;

int st[N][N][LG][LG];
int a[N][N], lg2[N];

int yo(int x1, int y1, int x2, int y2) {
  x2++;
  y2++;
  int a = lg2[x2 - x1], b = lg2[y2 - y1];
  return max(
         max(st[x1][y1][a][b], st[x2 - (1 << a)][y1][a][b]),
         max(st[x1][y2 - (1 << b)][a][b], st[x2 - (1 << a)][y2 - (1 << b)][a][b])
       );
}

void build(int n, int m) { // 0 indexed
  for (int i = 2; i < N; i++) lg2[i] = lg2[i >> 1] + 1;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
      st[i][j][0][0] = a[i][j];
    }
  }
  for (int a = 0; a < LG; a++) {
    for (int b = 0; b < LG; b++) {
      if (a + b == 0) continue;
      for (int i = 0; i + (1 << a) <= n; i++) {
        for (int j = 0; j + (1 << b) <= m; j++) {
          if (!a) {
            st[i][j][a][b] = max(st[i][j][a][b - 1], st[i][j + (1 << (b - 1))][a][b - 1]);
          } else {
            st[i][j][a][b] = max(st[i][j][a - 1][b], st[i + (1 << (a - 1))][j][a - 1][b]);
          }
        }
      }
    }
  }
}
```

3.3 hillbert Order

```
///Faster Sorting MO

const int infinity = (int)1e9 + 42;
const int64_t llInfinity = (int64_t)1e18 + 256;
const int module = (int)1e9 + 7;
const long double eps = 1e-8;

inline int64_t gilbertOrder(int x, int y, int pow, int rotate) {
    if (pow == 0) {
        return 0;
    }
    int hpow = 1 << (pow-1);
    int seg = (x < hpow) ? (
        (y < hpow) ? 0 : 3
    ) : (
        (y < hpow) ? 1 : 2
    );
    seg = (seg + rotate) & 3;
    const int rotateDelta[4] = {3, 0, 0, 1};
    int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
    int nrot = (rotate + rotateDelta[seg]) & 3;
    int64_t subSquareSize = int64_t(1) << (2*pow - 2);
    int64_t ans = seg * subSquareSize;
    int64_t add = gilbertOrder(nx, ny, pow-1, nrot);
    ans += (seg == 1 || seg == 2) ? add : (subSquareSize - add - 1);
    return ans;
}

struct Query {
    int l, r, idx;
    int64_t ord;

    inline void calcOrder() {
        ord = gilbertOrder(l, r, 21, 0);
    }
};

inline bool operator<(const Query &a, const Query &b) {
    return a.ord < b.ord;
}

signed main() {
    #ifndef USE_FILE_IO
        ios_base::sync_with_stdio(false);
    #endif

    mt19937 rnd(42);

    int n, m, k; cin >> n >> m; k = rnd() % 1048576;
    vector<int> p(n+1);
    for (int i = 0; i < n; i++) {
        int val = rnd() % 1048576;
        p[i+1] = p[i] ^ val;
    }

    vector<Query> qry(m);
    for (int i = 0; i < m; i++) {
        int l = rnd() % n + 1, r = rnd() % n + 1;
        if (l > r) {
            swap(l, r);
        }
        qry[i].l = l; qry[i].r = r;
        qry[i].idx = i;
        qry[i].calcOrder();
    }

    int64_t ans = 0;
    vector<int64_t> res(m);
    vector<int64_t> cnt((int)2e6, 0);
    sort(qry.begin(), qry.end());
    int l = 0, r = 1;
    ans = (p[1] == k);
    cnt[p[0]]++; cnt[p[1]]++;

    for (Query q: qry) {
        q.l--;
        while (l > q.l) {
            l--;
            ans += cnt[p[l] ^ k];
            cnt[p[l]]++;
        }
        while (r < q.r) {
            r++;
            ans += cnt[p[r] ^ k];
            cnt[p[r]]++;
        }
        while (l < q.l) {
            cnt[p[l]]--;
            ans -= cnt[p[l] ^ k];
            l++;
        }
        while (r > q.r) {
            cnt[p[r]]--;
            ans -= cnt[p[r] ^ k];
            r--;
        }
        res[q.idx] = ans;
    }

    uint64_t rhsh = 0;
    for (int i = 0; i < m; i++) {
        rhsh *= (uint64_t)1e9 + 7;
        rhsh += (uint64_t)res[i];
    }
    cout << rhsh << "\n";

    return 0;
}
```

3.4 Merge Sort Bit with updates

```
//O(log ^ 2 N) updates and queries


#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/rope>

using namespace std;
using namespace __gnu_pbds;
using namespace __gnu_cxx;

template<class T> using Tree = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;


Tree<int> t[N];

void add(int idx, int v){
    for(int x = ++idx; x < N; x += x & -x){
        t[x].insert(v);
    }
}
void erase(int idx, int v){
    for(int x = ++idx; x < N; x += x & -x)
        t[x].erase(v);
}
int get(int idx,  int limit){
  int ret = 0;
  for(int x = ++idx; x; x -= x & -x)
    ret += (t[x].order_of_key(limit+1));
  return ret;
}
```

3.5 Mo's

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

int n, qq, arr[N], sz = 1000; // sz  is the size of the bucket
int co[N],ans = 0, ansq[N];
int cul = 1, cur = 1;

void add(int x) {
  co[arr[x]]++;
  if (co[arr[x]] == 1)
    ans++;
  else if (co[arr[x]] == 2)
    ans--;
}

void remove(int x) {
  co[arr[x]]--;
  if (co[arr[x]] == 1)
    ans++;
  else if (co[arr[x]] == 0)
    ans--;
}

void solve(int l, int r,int ind) {
  r+=1;
  while (cul < l) remove(cul++);
  while (cul > l) add(--cul);
  while (cur < r) add(cur++);
  while (cur > r) remove(--cur);
  ansq[ind] = ans;
}


int main() {
  FIO
  cin >> qq;
 //                      {l/sz,r},     { l , ind}
 priority_queue<pair<pair<int, int>, pair<int, int>>, vector<pair<pair<int, int>, pair<int, int>>>, greater<pair<pair<int, int>, pair<int, int>>>> q;
  for (int i = 0; i < qq; i++) {
    int l, r;
    cin >> l >> r;
    q.push({{l / sz, r},{l,i}});
  }
  while (q.size()) {
    int ind=q.top().second.second,l=q.top().second.first,r=q.top().first.second;
    solve(l, r,ind);
    q.pop();
  }
  for (int i = 0; i < qq; i++)
    cout << ansq[i] << endl;


  return 0;
}
```

3.6 Mo With Updates

```

///O(N^5/3) note that the block size is not a standard size

#pragma GCC optimize ("O3")
#pragma GCC target ("sse4")

#include <bits/stdc++.h>

using namespace std;

using ll = long long;

const int N = 1e5 +5;
const int M = 2 * N;
const int blk = 2155;
const int mod = 1e9 + 7;
struct Query{
 int l, r, t, idx;
 Query(int a = 0,int b = 0,int c = 0,int d = 0){l=a,r=b,t=c,idx = d;}
 bool operator < (Query o){
  if(r / blk == o.r / blk && l / blk == o.l / blk)return t < o.t;
  if(r / blk == o.r / blk)return l < o.l;
  return r < o.r;
 }
} Q[N];

int a[N], b[N];
int cnt1[M], cnt2[N];
int L = 0, R = -1, K = -1;
void add(int x){ ///add item to range
//  cout << x << '\n';
 cnt2[cnt1[x]]--;
 cnt1[x]++;
 cnt2[cnt1[x]]++;
}
void del(int x){ ///delete item from range
 cnt2[cnt1[x]]--;
 cnt1[x]--;
 cnt2[cnt1[x]]++;
}
map<int,int>id;
int cnt;
int ans[N];
int p[N], nxt[N];
int prv[N];
void upd(int idx){ ///update item value
 if(p[idx] >= L && p[idx] <= R)
  del(a[p[idx]]), add(nxt[idx]);
 a[p[idx]] = nxt[idx];
}
void err(int idx){
 if(p[idx] >= L && p[idx] <= R)
  del(a[p[idx]]), add(prv[idx]);
 a[p[idx]] = prv[idx];
}
int main(){

 int n, q, l, r, tp;

 scanf("%d%d", &n, &q);

 for(int i = 0; i < n; i++){
  scanf("%d", a + i);
  if(id.count(a[i]) == 0)
   id[a[i]] = cnt++;
  a[i] = id[a[i]];
  b[i] = a[i];
 }
 int qIdx = 0;
 int ord = 0;
 while(q--){

  scanf("%d", &tp);
  if(tp == 1){
   /// ADD Query
   scanf("%d%d", &l, &r);  --l, --r;
   Q[qIdx] = Query(l,r,ord-1,qIdx); qIdx++;
```

```
    } else{
      /// ADD Update
      scanf("%d%d",p + ord, nxt + ord); --p[ord];
      if(id.count(nxt[ord]) == 0)
        id[nxt[ord]] = cnt++;
      nxt[ord] = id[nxt[ord]];
      prv[ord] = b[p[ord]];
      b[p[ord]] = nxt[ord];
      ++ord;
    }

  }
  sort(Q,Q+qIdx);
  for(int i = 0; i < qIdx; i++){
    while(L < Q[i].l)del(a[L++]);
    while(L > Q[i].l)add(a[--L]);
    while(R < Q[i].r)add(a[++R]);
    while(R > Q[i].r)del(a[R--]);
    while(K < Q[i].t)upd(++K);
    while(K > Q[i].t)err(K--);
    ///Solve Query I
  }
  for(int i = 0; i < qIdx; i++)
    printf("%d\n", ans[i]);


  return 0;
}
```

3.7 Ordered Set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

#define ordered_set tree<int, null_type,less<int>, rb_tree_tag, tree_order_statistics_node_update>

// order_of_key(k): returns the number of elements in the set strictly less than k
// find_by_order(k): returns an iterator to the k-th element (zero-based) in the set
```

3.8 Persistent Seg Tree

```

int val[ N * 60 ], L[ N * 60 ], R[ N * 60 ], ptr, tree[N]; /// N * lgN
int upd(int root, int s, int e, int idx) {
    int ret = ++ptr;
    val[ret] = L[ret] = R[ret] = 0;
    if (s == e) {
        val[ret] = val[root] + 1;
        return ret;
    }
    int md = (s + e) >> 1;
    if (idx <= md) {
        L[ret] = upd(L[root], s, md, idx), R[ret] = R[root];
    } else {
        R[ret] = upd(R[root], md + 1, e, idx), L[ret] = L[root];
    }
    val[ret] = max(val[L[ret]], val[R[ret]]);
    return ret;
}
int qry(int node, int s, int e, int l, int r){
  if(r < s || e < l || !node)return 0; //Punishment Value
  if(l <= s && e <= r){
    return val[node];
  }
  int md = (s+e)>>1;
  return max(qry(L[node], s, md, l, r), qry(R[node],md+1,e,l,r));
}
int merge(int x, int y, int s, int e) {
    if(!x||!y)return x | y;
    if(s == e) {
        val[x] += val[y];
        return x;
    }
    int md = (s + e) >> 1;
    L[x] = merge(L[x], L[y], s, md);
    R[x] = merge(R[x], R[y], md+1,e);
    val[x] = val[L[x]] + val[R[x]];
    return x;
}
```

3.9 Sqrt Decomposition

```
// Source: https://cp-algorithms.com/data_structures/sqrt_decomposition.html

// input data
int n;
vector<int> a (n);

// preprocessing
int len = (int) sqrt (n + .0) + 1; // size of the block and the number of blocks
vector<int> b (len);
for (int i=0; i<n; ++i)
    b[i / len] += a[i];

// answering the queries
for (;;) {
    int l, r;
  // read input data for the next query
    int sum = 0;
    for (int i=l; i<=r; )
        if (i % len == 0 && i + len - 1 <= r) {
            // if the whole block starting at i belongs to [l, r]
            sum += b[i / len];
            i += len;
        }
        else {
            sum += a[i];
            ++i;
        }
}

// If you're getting TLE and can't optimize more, you could reduce the number of slow division operations using the following code:

int sum = 0;
int c_l = l / len,   c_r = r / len;
if (c_l == c_r)
    for (int i=l; i<=r; ++i)
        sum += a[i];
else {
    for (int i=l, end=(c_l+1)*len-1; i<=end; ++i)
        sum += a[i];
    for (int i=c_l+1; i<=c_r-1; ++i)
        sum += b[i];
    for (int i=c_r*len; i<=r; ++i)
        sum += a[i];
}
```

3.10 Treap

```
typedef struct item * pitem;
struct item {
    int prior, value, cnt;
    bool rev;
    pitem l, r;
    item(int x, int y, int z){
        value = x;
        prior = y;
        cnt = z;
        rev = 0;
        l = r = NULL;
    }
};

int cnt (pitem it) {
    return it ? it->cnt : 0;
}

void upd_cnt (pitem it) {
    if (it)
        it->cnt = cnt(it->l) + cnt(it->r) + 1;
}

void push (pitem it) {
    if (it && it->rev) {
        it->rev = false;
        swap (it->l, it->r);
        if (it->l)  it->l->rev ^= true;
        if (it->r)  it->r->rev ^= true;
    }
}

void merge (pitem & t, pitem l, pitem r) {
    push (l);
    push (r);
    if (!l || !r)
        t = l ? l : r;
    else if (l->prior > r->prior)
        merge (l->r, l->r, r),  t = l;
    else
        merge (r->l, l, r->l),  t = r;
    upd_cnt (t);
}

void split (pitem t, pitem & l, pitem & r, int key, int add = 0) {
    if (!t)
        return void( l = r = 0 );
    push (t);
    int cur_key = add + cnt(t->l);
    if (key <= cur_key)
        split (t->l, l, t->l, key, add),  r = t;
    else
        split (t->r, t->r, r, key, add + 1 + cnt(t->l)),  l = t;
    upd_cnt (t);
}

void reverse (pitem t, int l, int r) {
    pitem t1, t2, t3;
    split (t, t1, t2, l);
    split (t2, t2, t3, r-l+1);
    t2->rev ^= true;
    merge (t, t1, t2);
    merge (t, t, t3);
}

void output (pitem t) {
    if (!t)  return;
    push (t);
    output (t->l);
    printf ("%c", char(t->value));
    output (t->r);
}

pitem gettreap(string s){
        pitem ret=NULL;
    int i;
       for(i=0;i<s.size();i++)merge(ret,ret,new item(s[i],(rand()<<15)+rand(),1));
    return ret;
}
```

3.11 Wavelet Tree

```
// remember your array and values must be 1-based
struct wavelet_tree {
    int lo, hi;
    wavelet_tree *l, *r;
    vector<int> b;

    //nos are in range [x,y]
    //array indices are [from, to)
    wavelet_tree(int *from, int *to, int x, int y) {
        lo = x, hi = y;
        if (lo == hi or from >= to)
            return;
        int mid = (lo + hi) / 2;
        auto f = [mid](int x) {
            return x <= mid;
        };
        b.reserve(to - from + 1);
        b.pb(0);
        for (auto it = from; it != to; it++)
            b.pb(b.back() + f(*it));
        //see how lambda function is used here
        auto pivot = stable_partition(from, to, f);
        l = new wavelet_tree(from, pivot, lo, mid);
        r = new wavelet_tree(pivot, to, mid + 1, hi);
    }

    //kth smallest element in [l, r]
    int kth(int l, int r, int k) {
        if (l > r)
            return 0;
        if (lo == hi)
            return lo;
        int inLeft = b[r] - b[l - 1];
        int lb = b[l - 1]; //amt of nos in first (l-1) nos that go in left
        int rb = b[r]; //amt of nos in first (r) nos that go in left
        if (k <= inLeft)
            return this->l->kth(lb + 1, rb, k);
        return this->r->kth(l - lb, r - rb, k - inLeft);
    }

    //count of nos in [l, r] Less than or equal to k
    int LTE(int l, int r, int k) {
        if (l > r or k < lo)
            return 0;
        if (hi <= k)
            return r - l + 1;
        int lb = b[l - 1], rb = b[r];
        return this->l->LTE(lb + 1, rb, k) + this->r->LTE(l - lb, r - rb, k);
    }

    //count of nos in [l, r] equal to k
    int count(int l, int r, int k) {
        if (l > r or k < lo or k > hi)
            return 0;
        if (lo == hi)
            return r - l + 1;
        int lb = b[l - 1], rb = b[r], mid = (lo + hi) / 2;
        if (k <= mid)
            return this->l->count(lb + 1, rb, k);
        return this->r->count(l - lb, r - rb, k);
    }
};
```

4 Matching

5 Trees

6 Strings

7 Geometry

8 Number Theory

9 DP

10 Misc.