

```

// Problem #      : snip
// Created on     : 2018-10-21 10:03:54
#include <bits/stdc++.h>
#define FR(i, n) for (int i = 0; i < (n); ++i)
using namespace std;

typedef long long ll;
typedef pair<int, int> ii;
typedef vector<int> vi;

// Binomial Coefficient
// Non-DP method, for when bottom-up DP with:
//  $C(n, k) = C(n - 1, k - 1) + C(n - 1, k)$ , is not feasible
ll C(int n, int k) {
    if (k == 0 || k == n) return 1;
    k = min(k, n - k); // Since  $C(n, k) == C(n, n - k)$ 
    ll ans = 1;
    for (ll i = 1; i <= k; i++) {
        ans *= (n - k + i) / i;
    }
    return ans;
}

// Count # of digits in number
int countDigits(long long n) {
    return n > 0 ? (int)log10((double)n) + 1 : 1;
}

// Dijkstra SSSP
vector< vector<ii> > g; vi D; int n;
void dijkstra(int s) {
    D.assign(n, -1);
    priority_queue<ii, vector<ii>, greater<ii> > pq;
    pq.emplace(0, s);
    while (!pq.empty()) {
        int u = pq.top().second, d = pq.top().first; pq.pop();
        if (D[u] != -1) continue;
        D[u] = d;
        for (auto &i : g[u]) {
            int v = i.second, w = i.first;
            pq.emplace(d + w, v);
        }
    }
}

// Prime factorization, n >= 0
// Complexity:  $O(\log(n))$ 
vi factor(int n) {
    vi f;
    if (n < 2) return vi(); // vi(1, n), if want 'n' for n < 2
    while (~n & 1) n >>= 1, f.push_back(2);
    for (ll p = 3; p * p <= n; p += 2)
        while (n % p == 0) n /= p, f.push_back((int)p);
    if (n > 1) f.push_back(n);
    return f;
}

```

```

// Compute nth Fibonacci number with matrix exponentiation
// Time complexity: O(log(n))
// Warning: 46th Fibonacci number (i.e. fib(46)) is largest
// that will fit into signed 32-bit integer; use long long if need
// longer.
int f[1000];
int fib(int n) {
    if (n < 2) return n;
    if (f[n]) return f[n];
    int k = (n + 1) / 2;
    f[n] = (n & 1) ? fib(k) * fib(k) + fib(k - 1) * fib(k - 1)
        : (2 * fib(k - 1) + fib(k)) * fib(k);
    return f[n];
}

// Floyd-warshall's APSP, initially g[i][j] is weight of path from i -> j
// for direct connections given; otherwise INF (0x3f3f3f3f ~ 1 bil).
// 'g' is represented as an adjacency matrix. This algorithm is generally
// useable as long as number of vertices, V <= 400 (approx.)
void floydWarshall_APSP(vector<vi> &g, int V) {
    FR(k, V) FR(i, V) FR(j, V) g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
}

// Least Common Multiple - O(log n), where n = max(a, b) (as in gcd)
int LCM(int a, int b) {
    return a * (b / __gcd(a, b));
}

// Longest Common Subsequence
const int MM = 20;
int lcsAndPath(int A[MM], int a, int B[MM], int b, int ans[MM]) {
    int L[MM + 1][MM + 1];
    for (int i = a; i >= 0; i--)
        for (int j = b; j >= 0; j--)
            if (i == a || j == b)
                L[i][j] = 0;
            else if (A[i] == B[j])
                L[i][j] = 1 + L[i + 1][j + 1];
            else
                L[i][j] = max(L[i + 1][j], L[i][j + 1]);

    int i = 0, j = 0, k = 0;
    while (i < a && j < b) {
        if (A[i] == B[j])
            ans[k++] = A[i], i++, j++;
        else if (L[i + 1][j] > L[i][j + 1])
            i++;
        else if (L[i + 1][j] < L[i][j + 1])
            j++;
        else
            j++; // tiebreaker
    }
    return L[0][0]; // len, ans has actual values as the path
}

```

```

// LIS - O(n log(n))
vi LIS(vi &A) {
    int n = A.size(), len = 1;
    // Uncomment to find for descending subsequence
    // reverse(A.begin(), A.end());
    vi last(n + 1), pos(n + 1), pred(n);
    if (n == 0) return vi();
    last[1] = A[pos[1] = 0];
    for (int i = 1; i < n; i++) {
        int j = upper_bound(begin(last) + 1, begin(last) + len + 1, A[i]) -
            last.begin();
        // Uncomment (and comment above line) for STRICTLY asc (desc)
        // int j = lower_bound(begin(last) + 1, begin(last) + len + 1, A[i]) -
        //     last.begin();
        pred[i] = (j - 1 > 0) ? pos[j - 1] : -1;
        last[j] = A[pos[j] = i];
        len = max(len, j);
    }
    // Uncomment if only need length, and end here
    // return len;
    int start = pos[len];
    vi S(len);
    for (int i = len - 1; i >= 0; i--) {
        S[i] = A[start]; start = pred[start];
    }
    return S;
}

// LIS - O(n^2)
// TODO: TEST THIS FUNCTION (may need n+1 size, etc.)
// ---> Ascending
void LIS_n2_asc(int n) {
    vi asc(n, 0);
    for (int i = n - 1; i >= 0; i--) {
        asc[i] = 1;
        for (int j = i + 1; j < n; j++)
            if (A[i] < A[j]) asc[i] = max(asc[i], asc[j] + 1);
    }
    return asc[0];
}

// <--- Descending
void LIS_n2_desc(int n) {
    vi desc(n, 0);
    for (int i = n - 1; i >= 0; i--) {
        desc[i] = 1;
        for (int j = i + 1; j < n; j++)
            if (A[i] > A[j]) desc[i] = max(desc[i], desc[j] + 1);
    }
    return desc[0];
}

// Modular Exponentiation
// Compute x^n mod m
int modexp(int x, int n, int m) {
    if (n == 0) return 1;
    if (n & 1) return ((x % m) * modexp(x, n - 1, m)) % m;
    int y = modexp(x, n / 2, m);
    return (y * y) % m;
}

```

```

// Palindrome
// Check if a string is a palindrome
bool palindrome(string s) {
    return equal(s.begin(), next(s.begin(), s.size() / 2), s.rbegin());
}

// RMQ - O(n log(n)), O(1) lookup
class RMQ {
public:
    vector<int> A; vector< vector<int> > M;
    RMQ(const vector<int> &B) {
        A = B; int n = A.size();
        int m = 31 - __builtin_clz(n) + 1;
        M.assign(m, vector<int>(n));
        FR(j, n) M[0][j] = j;
        for (int i = 1; (1 << i) <= n; i++) {
            for (int j = 0; (j + (1 << i)) <= n; j++) {
                M[i][j] = (A[M[i - 1][j]] <=
                    A[M[i - 1][j + (1 << (i - 1))]])
                    ? M[i - 1][j]
                    : M[i - 1][j + (1 << (i - 1))];
            }
        }
    }
    int query(int L, int R) {
        int k = 31 - __builtin_clz(R - L + 1);
        return (A[M[k][L]] <= A[M[k][R - (1 << k) + 1]])
            ? M[k][L]
            : M[k][R - (1 << k) + 1];
    }
};

// Sieve + optimized prime testing
ll sz; bitset<10000010> p; vi primes;
void sieve(ll m) {
    sz = m + 1;
    p.set(); p[0] = p[1] = 0;
    for (ll i = 2; i <= sz; i++)
        if (p[i]) {
            for (ll j = i * i; j <= sz; j += i) p[j] = 0;
            primes.push_back((int)i);
        }
}
bool isPrime(ll x) {
    if (x <= sz) return p[x];
    for (int i = 0; i < (int)primes.size(); i++)
        if (x % primes[i] == 0) return false;
    return true;
}

```

```

// Union Find (Disjoint Set)
class UF {
public:
    vi p, r;
    UF(int n) {
        p.assign(n, 0); iota(begin(p), end(p), 0);
        r.assign(n, 0);
    }
    int find(int n) { return (p[n] == n ? n : (p[n] = find(p[n]))); }
    bool same(int i, int j) { return find(i) == find(j); }
    void merge(int i, int j) {
        if (!same(i, j)) {
            int x = find(i), y = find(j);
            if (r[x] > r[y]) p[y] = x;
            else {
                p[x] = y;
                if (r[x] == r[y]) r[y]++;
            }
        }
    }
};

int main() {

    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

}

```