Contents

1 Sublime Text Setting

```
{
  "cmd": ["g++", "$file", "-o", "${file_path}/${file_base_name}"],
  "file_regex": "^(..[^:]*):([0-9]+):?([0-9]+)?:? (.*)$",
  "working_dir": "${file_path}",
  "selector": "source.c, source.c++, source.cxx, source.cpp",
  "variants":
  [
      {
          "name": "Run",
          "shell": true,
          "cmd": ["g++ -std=c++11 -O2 ₩"${file}₩" -o ₩"${file_path}/${file_base_name}₩" && gnome-terminal -e 'bash -c ₩"${file_path}/${file_base_name} < ₩"${file_path}/input₩";echo;echo; echo Press ENTER to continue; read line;exit; exec bash₩"'"]
      }
  ]
}
```

```
sudo /etc/init.d/ssh stop
sudo /etc/init.d/networking restart
sudo -i
sudo nautilus
```

2 Convex Hull Optimization

```
int N;
ll dp[100003], A[100003],B[100003];
pair<ll, ll> stck[100003];
int size = 0;
double cross(int a, int b){
    return (double)(stck[b].second - stck[a].second) / (stck[a].first - stck[b].first);
}
void insert(ll a, ll b){
    stck[size] = {a, b};
    while(size > 1 && cross(size - 2, size - 1) > cross(size - 1, size)){
        stck[size - 1] = stck[size];
        size--;
    }
    size++;
}
ll query(ll x){
    int i;
    for(i = 0; i < size - 1; i++){
        if(cross(i, i + 1) >= x) break;
    }
    return stck[i].first * x + stck[i].second;
}
int main() {
     fastio();
    cin >> N;
     for(int i = 1; i <= N; i++) cin >> A[i];
     for(int i = 1; i <= N; i++) cin >> B[i];
    dp[1] = 0;
    insert(B[1], 0);
     for(int i = 2; i <= N; i++){
        dp[i] = query(A[i]);
        insert(B[i], dp[i]);
    }
    cout << dp[N];
    return 0;
}
```

3 Divide And Conquer Optimization

```
int N, M;
int A[8003], K[803][8003];
ll pSum[8003], D[803][8003];
ll C(int l, int r){
    return (pSum[r] - pSum[l - 1]) * (r - l + 1);
}
void DC(int i, int l, int r, int p, int q){
    if(l > r) return;
    int mid = (l + r) >> 1;
    D[i][mid] = infl;
    for(int k = p; k <= q && k < mid; k++){
        if(D[i][mid] > D[i - 1][k] + C(k + 1, mid)) D[i][mid] = D[i - 1][k] + C(k + 1, mid),K[i][mid] = k;
    }
    DC(i, l, mid - 1, p, K[i][mid]);
    DC(i, mid + 1, r, K[i][mid], q);
}
int main() {
    memset(D,0x3c,sizeof(D));
```

```
    cin >> N >> M;
    for(int i = 1; i <= N; i++) {
        cin >> A[i];
        pSum[i] = pSum[i - 1] + A[i];
    }
    if(N<=M) return !printf("%lld", pSum[N]);
    D[0][0] = 0;
    for(int i = 1; i <= M; i++) DC(i, 1, N, 0, N - 1);
    cout << D[M][N];
    return 0;
}
```

4 Knuth Optimization

```
ll D[5003][5003];
int pSum[5003],K[5003][5003], f[5003];
int n, T;
int main() {
    fastio();
    for(cin >> T; T--;){
        cin >> n;
        for(int i = 1; i <= n; i++) {
            cin >> f[i];
            pSum[i] = pSum[i - 1] + f[i];
        }
        memset(D,0x3c,sizeof(D));
        for(int i = 1; i <= n; i++) D[i][i + 1] = 0, K[i][i + 1] = i + 1;
        for(int size = 2; size <= n; size++){
            for(int i = 1; i + size <= n + 1; i++){
                int j = i + size;
                for(int k = K[i][j-1]; k <= K[i+1][j]; k++){
                    if(D[i][j] > D[i][k] + D[k][j] + pSum[j-1] - pSum[i-1]) {
                        D[i][j] = D[i][k] + D[k][j] + pSum[j-1] - pSum[i-1];
                        K[i][j] = k;
                    }
                }
            }
        }
        cout << D[1][n + 1] << '₩n';
    }
    return 0;
}
```

5 Deque Optimization

```
ll psum[100003], A[100003], dp[100003];
deque<int> dq;
int N, K;
ll C(int i){
    return dp[i - 1] - psum[i];
}
int main() {
    fastio();
    cin >> N >> K;
    for(int i = 1 ; i <= N; i++) cin >> A[i];
    for(int i = 1 ; i <= N; i++) psum[i] = psum[i - 1] + A[i];
    for(int i = 1; i <= N; i++){
        while(!dq.empty() && dq.front() < i - K) dq.pop_front();
        while(!dq.empty() && C(dq.back()) <= C(i)) dq.pop_back();
        dq.push_back(i);
```

```
            dp[i] = psum[i] + C(dq.front());
            if(i <= K) dp[i] = max(dp[i], psum[i]);
        }
        cout << dp[N];
        return 0;
}
```

6 FFT

```
typedef complex<double> base;
const double PI = 2.0 * acos(0.0);
void fft(vector <base> &a, bool invert) {
    int n = sz(a);
    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) {
        double ang = 2 * PI / len * (invert ? -1 : 1);
        base wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            base w(1);
            for (int j = 0; j < len / 2; j++) {
                base u = a[i + j], v = a[i + j + len / 2] * w;
                a[i + j] = u + v;
                a[i + j + len / 2] = u - v;
                w *= wlen;
            }
        }
    }
    if (invert) {
        for (int i = 0; i < n; i++) a[i] /= n;
    }
}
void multiply(const vector<int> &a, const vector<int> &b, vector<int> &res) {
    vector<base> fa(all(a)), fb(all(b));
    int n = 1;
    while (n < max(sz(a), sz(b))) n <<= 1; n <<= 1;
    fa.resize(n); fb.resize(n);
    fft(fa, false); fft(fb, false);
    for (int i = 0; i < n; i++) fa[i] *= fb[i];
    fft(fa, true);
    res.resize(n);
    for (int i = 0; i < n; i++) res[i] = int(fa[i].real() + (fa[i].real() > 0 ? 0.5 :
-0.5));
    while(!res.empty() && res.back() == 0) res.pop_back();
}
const int MAX_L = (3 e5 + 9) * 2;
vector<int> A, B, res;
int ans[MAX_L];
int main() {
    fastio();
    string s1, s2;
    cin >> s1 >> s2;
    int s1sz = s1.size(), s2sz = s2.size();
    for(int i = 0; i < s1sz; i++){
        A.push_back(s1[i] - '0');
    }
    for(int i = 0; i < s2sz; i++){
        B.push_back(s2[i] - '0');
```

```
    }
    reverse(A.begin(), A.end());
    reverse(B.begin(), B.end());
    multiply(A, B, res);
    int ressz = res.size();
    for(int i = 0; i < ressz; i++){
        int v = (ans[i] + res[i]);
        ans[i] = v % 10;
        ans[i + 1] = v / 10;
    }
    int len = ressz;
    while(len >= 0 && ans[len] == 0) len--;
    if(len < 0) printf("0");
    for(int i = len; i >= 0; i--) printf("%d", ans[i]);
}
```

7 RNG

```
int main(){
    fastio();
    mt19937 rng(time(0));
    for(int i = 0; i < 100; i++) {
        cout << rng() << '₩n';
    }
}
```

8 Kth Number

```
int t[262145], A[250003];
int N, K, ans;
void update(int pos, int v){
    t[pos += (1<<16)] += v;
    for(;pos>1; pos/=2) t[pos/2] = t[pos] + t[pos^1];
}
int search(int k){
    int pos = 1;
    while(pos < (1<<16)){
        if(k <= t[2*pos]) pos*=2;
        else k-=t[2*pos], pos=2*pos+1;
    }
    return pos-(1<<16);
}
int main() {
    cin >> N >> K;
    for(int i = 0 ; i < N; i++) cin >> A[i];
    for(int i = 0 ; i < K; i++) update(A[i],1);
    long long sum = 0;
    for(int i = K; i <= N; i++){
        sum += search((K+1)/2);
        if(i==N) break;
        update(A[i],1);
        update(A[i-K],-1);
    }
    cout << sum;
    return 0;
}
```

9 Miller Rabin And Pollard Rho

```
ll mul(ll a, ll b, ll mod){
    ll res = 0;
    while(b){
        if(b & 1) res = (res + a) % mod;
        a = (a * 2) % mod;
        b /= 2;
    }
    return res;
}
ll Pow(ll a, ll b, ll mod){
    ll res = 1;
    ll x = a;
    while(b){
        if(b & 1) res = mul(res, x, mod);
        x = mul(x, x, mod);
        b /= 2;
    }
    return res;
}
bool isPrime(ll n){
    const static vector<int> arr = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
    if(n == 1) return false;
    for(int i = 0; i < (int)arr.size(); i++){
        if(arr[i] == n) return true;
        if(Pow(arr[i], n - 1, n) != 1) return false;
    }
    return true;
}
```

```
ll rho(ll n) {
    ll x, y, d, c = -1;
    if(n % 2 == 0) return 2;
    while(1){
        x = y = 2;
        while(1){
            x = mul(x, x, n);
            x = (x - c) % n;
            y = mul(y, y, n);
            y = (y - c) % n;
            y = mul(y, y, n);
            y = (y - c) % n;
            d = gcd(abs(n + y - x), n);
            if (d == n) break;
            else if (d > 1) return d;
        }
        c--;
    }
}
void fator(ll n, vector<ll> &arr){
    if(isPrime(n)){
        arr.push_back(n);
        return;
    }
    ll f = rho(n);
    fator(f, arr); fator(n / f, arr);
}
```

10 Chinese Remainder

```
ll xGCD(ll a, ll b, ll& x, ll& y){
    if(!b){
        x = 1, y = 0;
        return a;
    }
    ll x1, y1;
    int ret = xGCD(b, a % b, x1, y1);
    x = y1, y = x1 - (a / b) * y1;
    return ret;
}
ll chinese(vector<ll>& mods, vector<ll>& remains){
    ll lcm = mods[0];
    ll trash;
    for(int i = 1; i < (int)mods.size(); i++){
        lcm = lcm * mods[i] / xGCD(lcm, mods[i], trash, trash);
    }
    ll ret = 0;
    ll N = 1;
    for(ll num : mods) N *= num;
    for(int i = 0 ; i < (int)mods.size(); i++){
        ll n = mods[i];
        ll x, y;
        xGCD(n, N / n, x, y);
        if(y < 0) y %= N, y += N;
        ret = (ret + N / n * y % lcm * remains[i] % lcm) % lcm;
    }
    return ret;
}
```

11 Chinese Remainder for non coprime

```
ll xGCD(ll a, ll b, ll& x, ll& y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    ll x1, y1;
    int ret = xGCD(b, a % b, x1, y1);
    x = y1, y = x1 - (a / b) * y1;
    return ret;
}
ll chinese(vector<ll>& mods, vector<ll>& remains) {
    ll lcm = 1;
    for (ll num : mods) lcm *= num;
    ll ret = 0;
    ll N = 1;
    for (ll num : mods) N *= num;
    for (int i = 0; i < (int)mods.size(); i++) {
        ll n = mods[i];
        ll x, y;
        xGCD(n, N / n, x, y);
        if (y < 0) y %= N, y += N;
        ret = (ret + N / n * y % lcm * remains[i] % lcm) % lcm;
    }
    return ret;
}
void addMap(map<ll, pair<ll, ll> >& mp, ll idx, pair<ll, ll> pr) {
    if (mp.find(idx) != mp.end() && mp[idx].first < pr.first) mp.erase(idx);
    if (mp.find(idx) == mp.end()) mp[idx] = pr;
```

```
}
void makeVectors(vector<ll>& prevMods, vector<ll>& prevRem, vector<ll>& mods, vector<ll>& remains) {
    map<ll, pair<ll, ll> > mp;
    for (int i = 0; i < (int)prevMods.size(); i++) {
        ll back = prevMods[i];
        for (ll j = 2; j * j <= prevMods[i]; j++) {
            ll prev = back;
            int cnt = 0;
            while (back % j == 0) cnt++, back /= j;
            addMap(mp, j, { prev / back, prevRem[i] });
        }
        if (back != 1) addMap(mp, back, { back, prevRem[i] });
    }
    for (auto it : mp) {
        ll md = it.second.first;
        ll rem = it.second.second;
        rem %= md;
        mods.push_back(md);
        remains.push_back(rem);
    }
}
bool check(vector<ll>& mods, vector<ll>& remains) {
    int n = (int)mods.size();
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j || remains[i] < remains[j]) continue;
            ll trash;
            ll left = xGCD(mods[i], mods[j], trash, trash);
            ll right = remains[i] - remains[j];
            if (right % left) return false;
        }
    }
    return true;
}
ll solve(vector<ll>& prevMods, vectror<ll>& prevRem){
    if(!check(prevMods, prevRem)) return -1;
    vector<ll> mods, remains;
    makeVectors(prevMods, prevRem, mods, remains);
    return chinese(mods, remains);
}
```

12 LCA

```
/*노드 번호는 0번부터 시작*/
struct LCA{
    vector<vector<int> > G, par;
    vector<int> dep;
    int root, size, depLog;
    LCA(int size, int root){
        this->root = root;
        this->size = size;
        depLog = 1;
        int sz = 1;
        while(sz <= size) depLog++, sz *= 2;
        par = vector<vector<int> >(size, vector<int>(depLog + 1, -1));
        G = vector<vector<int> >(size);
        dep = vector<int>(size, 0);
    }
    void buildLCA(){
        dfs(root, -1);
```

```
    }
    void addEdge(int from, int to){
        G[from].push_back(to);
        G[to].push_back(from);
    }
    void dfs(int cur, int dad){
        for(int i = 1; (1<<i) <= dep[cur]; i++) par[cur][i] =
par[par[cur][i-1]][i-1];
        for(int next : G[cur]){
            if(next == dad) continue;
            par[next][0] = cur;
            dep[next] = dep[cur] + 1;
            dfs(next, cur);
        }
    }
    int getLCA(int a, int b){
        if(dep[a] < dep[b]) swap(a, b);
        for(int i = 0; dep[a] != dep[b]; i++){
            int diff = dep[a] - dep[b];
            if(diff & (1<<i)) a = par[a][i];
        }
        for(int i = depLog; i >= 0; i--){
            if(par[a][i] == -1) continue;
            if(par[a][i] != par[b][i]) a = par[a][i], b = par[b][i];
        }
        if(a == b) return a;
        return par[a][0];
    }
};
```

13 SCC & 2-SAT

```
/*1. 그래프 구성 뒤 답을 구하기 전 build 함수 호출
2. getAns 함수로 답을 구함(답이 존재하지 않으면 빈 벡터 반환)*/
struct SCC{
    vector<vector<int> > G;
    vector<int> sccId, order;
    int visitCnt, sccCnt, size;
    stack<int> stck;
    SCC(){}
    SCC(int size){
        this->size = size;
        G = vector<vector<int> >(size);
    }
    void buildSCC(){
        sccId = vector<int>(size, -1);
        order = vector<int>(size, -1);
        visitCnt = sccCnt = 0;
        for(int i = 0 ; i < size; i++) {
            if(order[i] == -1) dfs(i);
        }
        for(int i = 0 ; i < size; i++) sccId[i] = sccCnt - sccId[i] - 1;
    }
    void addEdge(int from, int to){
        G[from].push_back(to);
    }
    int dfs(int cur){
        stck.push(cur);
        int ret = order[cur] = visitCnt++;
        for(int next : G[cur]){
```

```
            if(order[next] == -1) ret = min(ret, dfs(next));
            else if(sccId[next] == -1) ret = min(ret, order[next]);
        }
        if(order[cur] == ret){
            while(1){
                int top = stck.top();
                stck.pop();
                sccId[top] = sccCnt;
                if(top == cur) break;
            }
            sccCnt++;
        }
        return ret;
    }
    vector<vector<int> > getCompGraph(){
        vector<vector<int> > compG = vector<vector<int> >(sccCnt);
        for(int cur = 0 ; cur < size; cur++){
            for(int next : G[cur]){
                if(sccId[cur] == sccId[next]) continue;
                compG[sccId[cur]].push_back(sccId[next]);
            }
        }
        sort(compG.begin(), compG.end());
        for(int i = 0; i < sccCnt; i++) {
            compG[i].erase(unique(compG[i].begin(), compG[i].end()),
compG[i].end());
        }
        return compG;
    }
};
struct TwoSAT{
    int size;
    SCC scc;
    vector<int> ans;
    bool isPossible;
    TwoSAT(int size){
        this->size = size;
        scc = SCC(size);
    }
    void buildTwoSAT(){
        scc.buildSCC();
        vector<int> sccId = scc.sccId;
        isPossible = true;
        for(int i = 0 ; i < size; i+=2){
            if(sccId[i] == sccId[i+1]) isPossible = false;
        }
        ans = vector<int>(size / 2, -1);
        for(int i = 0; i < size; i += 2){
            ans[i / 2] = sccId[i] > sccId[i + 1];
        }
    }
    void addEdge(int from, int to){
        scc.addEdge(from, to);
    }
    vector<int> getAns(){
        if(!isPossible) return vector<int>();
        return ans;
    }
};
int NOT(int x){
    return x^1;
```

```
}
int TRANS(int x){
    if(x < 0) return NOT((-x-1)*2);
    return (x-1)*2;
}
```

14 Segment Tree with lazy propagation

```
struct Segtree{
    vector<ll> tree, lazy;
    int size;
    Segtree(vector<ll>& arr){
        int tmpsize = (int)arr.size();
        size = 1;
        while(size < tmpsize) size *= 2;
        tree = vector<ll>(2 * size, 0);
        lazy = vector<ll>(2 * size, 0);
        for(int i = 0 ; i < (int)arr.size(); i++) tree[size + i] = arr[i];
        for(int i = size - 1; i > 0; i--) tree[i] = tree[2 * i] + tree[2 * i + 1];
    }
    ll query(int s, int e, int nl, int nr, int nd){
        updateLazy(nl, nr, nd);
        if(s <= nl && nr <= e) return tree[nd];
        if(e < nl || nr < s) return 0;
        int nm = (nl + nr) / 2;
        ll leftRes = query(s, e, nl, nm, 2 * nd);
        ll rightRes = query(s, e, nm + 1, nr, 2 * nd + 1);
        return leftRes + rightRes;
    }
    ll update(int s, int e, int nl, int nr, ll val, int nd){
        updateLazy(nl, nr, nd);
        if(s <= nl && nr <= e) {
            lazy[nd] += val;
            updateLazy(nl, nr, nd);
            return tree[nd];
        }
        if(e < nl || nr < s) return tree[nd];
        int nm = (nl + nr) / 2;
        ll leftRes = update(s, e, nl, nm, val, 2 * nd);
        ll rightRes = update(s, e, nm + 1, nr, val, 2 * nd + 1);
        return tree[nd] = leftRes + rightRes;
    }
    void updateLazy(int nl, int nr, int nd){
        if(lazy[nd]) {
            tree[nd] += (nr - nl + 1) * lazy[nd];
            if(nd < size) lazy[2 * nd] += lazy[nd], lazy[2 * nd + 1] += lazy[nd];
            lazy[nd] = 0;
        }
    }
    ll query(int s, int e){
        return query(s, e, 0, size - 1, 1);
    }
    ll update(int s, int e, ll val){
        return update(s, e, 0, size - 1, val, 1);
    }
};
```

15 Plain Sweep

```
struct Rect{
    ll x1, y1, x2, y2;
};
struct RNG{
    ll l, r;
};
Rect rs[200003];
ll tr[1600003], cnt[1600003];
vector<pair<int,int>> v;
vector<ll> ys;
vector<RNG> rngs;
int N;
void update(int l, int r, int nl, int nr, int nd, int v){
    if(rngs[nr].l < l || r < rngs[nl].r) return;
    if(l <= rngs[nl].l && rngs[nr].r <= r) {
        cnt[nd] += v;
        if(cnt[nd]) tr[nd] = rngs[nr].r - rngs[nl].l;
        else tr[nd] = (nl == nr ? 0 : tr[2 * nd] + tr[2 * nd + 1]);
        return;
    }
    int nm = nl + nr >> 1;
    update(l, r, nl, nm, 2 * nd, v);
    update(l, r, nm + 1, nr, 2 * nd + 1, v);
    if(cnt[nd]) tr[nd] = rngs[nr].r - rngs[nl].l;
    else tr[nd] = (nl == nr ? 0 : tr[2 * nd] + tr[2 * nd + 1]);
}
int main(){
    fastio();
    cin >> N;
    for(int i = 1; i <= N; i++){
        ll x1, x2, y1, y2;
        cin >> x1 >> x2 >> y1 >> y2;
        rs[i] = Rect{x1, y1, x2, y2};
        v.pb({x1, i});
        v.pb({x2, -i});
        ys.pb(y1);
        ys.pb(y2);
    }
    sort(all(ys));
    ys.erase(unique(all(ys)), ys.end());
    for(int i = 0; i < sz(ys) - 1; i++) rngs.pb(RNG{ys[i], ys[i + 1]});
    sort(all(v));
    ll prv = -1;
    ll ans = 0;
    for(auto p : v){
        ll cur = p.first;
        int isEnd = p.second < 0;
        int id = abs(p.second);
        ans += (cur - prv) * tr[1];
        update(rs[id].y1, rs[id].y2, 0, sz(rngs) - 1, 1, isEnd ? -1 : 1);
        prv = cur;
    }
    cout << ans << '₩n';
}
```

16 Dinic

```
struct Dinic{
    struct edge{
        int to, cap, rev;
    };
    int size, src, sink;
    vector<vector<edge> > G;
    vector<int> level, iter;
    Dinic(){}
    Dinic(int size, int src, int sink) {
        this->size = size;
        this->src = src;
        this->sink = sink;
        G = vector<vector<edge> >(size);
        level = vector<int>(size, -1);
        iter = vector<int>(size, 0);
    }
    void addEdge(int from, int to, int cap){
        G[from].push_back({to, cap, (int)G[to].size()});
        G[to].push_back({from, 0, (int)G[from].size()-1});
    }
    bool bfs(int src, int sink){
        queue<int> q;
        q.push(src);
        level[src] = 0;
        while(!q.empty() && level[sink] == -1){
            int here = q.front();
            q.pop();
            for(auto e : G[here]){
                if(e.cap <= 0 || level[e.to] != -1) continue;
                level[e.to] = level[here] + 1;
                q.push(e.to);
            }
        }
        return level[sink] != -1;
    }
    int dfs(int here, int minFlow, int sink){
        if(here == sink) return minFlow;
        for(int& i = iter[here]; i < (int)G[here].size(); i++){
            auto& e = G[here][i];
            if(e.cap == 0 || level[e.to] != level[here] + 1) continue;
            int f = dfs(e.to, min(minFlow, e.cap), sink);
            if(f > 0){
                e.cap -= f;
                G[e.to][e.rev].cap += f;
                return f;
            }
        }
        return 0;
    }
    int getMaxflow(){
        int ret = 0;
        while(1){
            level = vector<int>(size, -1);
            iter = vector<int>(size, 0);
            if(!bfs(src, sink)) break;
            while(int f = dfs(src, INF, sink)) ret += f;
        }
        return ret;
```

```
    }
};
```

17 Hopcroft

```
struct Hopcroft{
    int asize, bsize;
    vector<vector<int> > G;
    vector<int> level, iter;
    vector<int> aMatch, bMatch;
    Hopcroft(int asize, int bsize){
        this->asize = asize;
        this->bsize = bsize;
        G = vector<vector<int> >(asize);
        level = vector<int>(asize, -1);
        iter = vector<int>(asize, 0);
        aMatch = vector<int>(asize, -1);
        bMatch = vector<int>(bsize, -1);
    }
    void addEdge(int from, int to){
        G[from].push_back(to);
    }
    void bfs(){
        queue<int> q;
        for(int a = 0 ; a < asize; a++) {
            if(aMatch[a] == -1) {
                level[a] = 0;
                q.push(a);
            }
        }
        while(!q.empty()) {
            int a = q.front();
            q.pop();
            for(int& i = iter[a]; i < (int)G[a].size(); i++){
                int b = G[a][i];
                if(bMatch[b] == -1 || level[bMatch[b]] != -1) continue;
                level[bMatch[b]] = level[a] + 1;
                q.push(bMatch[b]);
            }
        }
    }
    bool dfs(int a){
        for(int b : G[a]){
            if(bMatch[b] == -1 || (level[bMatch[b]] == level[a] + 1 &&
dfs(bMatch[b]))) {
                aMatch[a] = b;
                bMatch[b] = a;
                return true;
            }
        }
        return false;
    }
    int getMaxMatch(){
        int ret = 0;
        while(1){
            level = vector<int>(asize, -1);
            iter = vector<int>(asize, 0);
            bfs();
            int add = 0;
            for(int a = 0 ; a < asize; a++) {
                if(level[a] == 0 && dfs(a)) add++;
```

```
            }
            if(!add) break;
            ret += add;
        }
        return ret;
    }
};
```

18 MCMF

```
struct MCMF{
    struct edge{
        int to, cap, cost, rev;
    };
    int size, src, sink;
    vector<vector<edge> > G;
    vector<int> dist, par, edgeIdx;
    MCMF(int size, int src, int sink){
        G = vector<vector<edge> >(size);
        par = vector<int>(size);
        edgeIdx = vector<int>(size);
        this->size = size;
        this->src = src;
        this->sink = sink;
    }
    bool spfa(){
        dist = vector<int>(size, inf);
        vector<bool> inQ = vector<bool>(size, false);
        queue<int> q;
        q.push(src);
        inQ[src] = true;
        dist[src] = 0;
        while(!q.empty()){
            int here = q.front();
            q.pop();
            inQ[here] = false;
            for(int i = 0 ; i < (int)G[here].size(); i++){
                auto e = G[here][i];
                if(e.cap > 0 && dist[here] + e.cost < dist[e.to]) {
                    dist[e.to] = dist[here] + e.cost;
                    par[e.to] = here;
                    edgeIdx[e.to] = i;
                    if(!inQ[e.to]) q.push(e.to), inQ[e.to] = true;
                }
            }
        }
        return dist[sink] != inf;
    }
    pair<int,int> getMCMF(){
        int maxFlow = 0;
        int minCost = 0;
        while(1){
            if(!spfa()) break;
            int minFlow = inf;
            int costSum = 0;
            for(int p = sink; p != src; p = par[p]){
                auto& e = G[par[p]][edgeIdx[p]];
                minFlow = min(minFlow, e.cap);
                costSum += e.cost;
            }
            for(int p = sink; p != src; p = par[p]){
```

```
            auto& e = G[par[p]][edgeIdx[p]];
            e.cap -= minFlow;
            G[e.to][e.rev].cap += minFlow;
        }
        maxFlow += minFlow;
        minCost += costSum * minFlow;
    }
    return {maxFlow, minCost};
}
void addEdge(int from, int to, int cap, int cost){
    G[from].push_back({to, cap, cost, (int)G[to].size()});
    G[to].push_back({from, 0, -cost, (int)G[from].size()-1});
}
};
```

19 LR Max Flow

```
/*
    새로운 소스와 싱크 s' 와 t' 를 만든다.
    새로운 유량 그래프에서 최대 유량을 구할 땐
    s' 가 소스가 되고 t'가 싱크가 된다.
    기존의 그래프의 각각의 노드 v에 대하여
    s' -> v 로 간선을 하나씩 이어주는데
    이 때의 용량은 v로 들어가는 모든 엣지들의
    demand 유량의 합이다.
    기존의 그래프의 각각의 노드 v에 대하여
    v -> t' 로 간선을 하나씩 이어주는데
    이 때의 용량은 v에서 나가는 모든 엣지들의
    demand 유량의 합이다.
    t -> s 로 무한대 용량의 간선을 만들어준다.
    기존의 모든 엣지들의 용량들에 대해
    자신의 demand 유량만큼 빼준다.
    maximum-flow 돌린 후에
    각 간선에 흐르는 유량에 그 간선의 demand 유량을
    더하면 원래 그래프에서 실제로 흐르는 유량이 됨
    그리고 LR-circulation 이 feasible 한지 판별하는 것은
    위와 같이 maximum-flow 를 돌린 후 최대유량이
    모든 간선들의 demand 유량의 합과 같으면 feasible,
    다르면 unfeasible!
*/
struct Dinic{
    struct edge{
        int to, cap, rev;
    };
    int size, src, sink;
    vector<vector<edge> > G;
    vector<int> level, iter;
    Dinic(){}
    Dinic(int size, int src, int sink) {
        this->size = size;
        this->src = src;
        this->sink = sink;
        G = vector<vector<edge> >(size);
        level = vector<int>(size, -1);
        iter = vector<int>(size, 0);
    }
    void addEdge(int from, int to, int cap){
        G[from].push_back({to, cap, (int)G[to].size()});
        G[to].push_back({from, 0, (int)G[from].size()-1});
    }
```

```
bool bfs(int src, int sink){
    queue<int> q;
    q.push(src);
    level[src] = 0;
    while(!q.empty() && level[sink] == -1){
        int here = q.front();
        q.pop();
        for(auto e : G[here]){
            if(e.cap <= 0 || level[e.to] != -1) continue;
            level[e.to] = level[here] + 1;
            q.push(e.to);
        }
    }
    return level[sink] != -1;
}
int dfs(int here, int minFlow, int sink){
    if(here == sink) return minFlow;
    for(int& i = iter[here]; i < (int)G[here].size(); i++){
        auto& e = G[here][i];
        if(e.cap == 0 || level[e.to] != level[here] + 1) continue;
        int f = dfs(e.to, min(minFlow, e.cap), sink);
        if(f > 0){
            e.cap -= f;
            G[e.to][e.rev].cap += f;
            return f;
        }
    }
    return 0;
}
```

```
    int getMaxflow(){
        int ret = 0;
        while(1){
            level = vector<int>(size, -1);
            iter = vector<int>(size, 0);
            if(!bfs(src, sink)) break;
            while(int f = dfs(src, inf, sink)) ret += f;
        }
        return ret;
    }
};
struct LRMaxFlow{
    Dinic dinic;
    int size, src, sink, fsrc, fsink;
    vector<int> inSum, outSum;
    LRMaxFlow(int size, int src, int sink){
        this->size = size;
        this->src = src;
        this->sink = sink;
        fsrc = size;
        fsink = size + 1;
        dinic = Dinic(size + 2, fsrc, fsink);
        inSum = vector<int>(size, 0);
        outSum = vector<int>(size, 0);
    }
    void addEdge(int from, int to, int lower, int upper){
        dinic.addEdge(from, to, upper - lower);
        inSum[to] += lower;
        outSum[from] += lower;
    }
```

```
    int getMaxflow(){
        for(int i = 0 ; i < size; i++) if(inSum[i]) dinic.addEdge(fsrc, i, inSum[i]);
        for(int i = 0 ; i < size; i++) if(outSum[i]) dinic.addEdge(i, fsink, outSum[i]);
        dinic.addEdge(sink, src, inf);
        return dinic.getMaxflow();
    }
};
```

20 Fast IO(Kth element)

```
static char s[75000000];
static inline void init(){fread(s, 1, sizeof(s), stdin);}
static inline void readN(int &r){
    static char *p = s;
    while (*p < 45) p++;
    int m = 0;
    for (r = 0; *p >= '-'; p++){
        if(*p == '-') {
            m = 1;
            continue;
        }
        r = r * 10 + (*p & 15);
    }
    if(m) r = -r;
}
int A[5000003];
int N, K;
int main(){
    init();
    readN(N);
    readN(K);
    for(int i = 0; i < N; i++) {
        readN(A[i]);
    }
    nth_element(A, A + K - 1, A + N);
    printf("%d", A[K - 1]);
}
```

21 SA & LCP

```
/*SA는 0 base고 LCP는 1 base이다*/
void countingSort(vector<int>& SA, vector<int>& g, int t){
    int n = sz(SA);
    vector<int> tSA(n), c(max(330, n + 1), 0);
    for(int i = 0 ; i < n; i++) c[i + t < n ? g[i + t] : 0]++;
    for(int i = 0, sum = 0, tmp ; i < max(330, n + 1); tmp = c[i], c[i] = sum, sum += tmp, i++);
    for(int i = 0 ; i < n; i++) tSA[c[SA[i] + t < n ? g[SA[i] + t] : 0]++] = SA[i];
    SA = tSA;
}
vector<int> getSA(string& S){
    int n = (int)S.size();
    vector<int> SA(n), g(n + 1), tg(n + 1);
    for(int i = 0 ; i < n; i++) SA[i] = i;
    for(int i = 0 ; i < n; i++) g[i] = S[i];
    for(int t = 1; t < n; t*=2){
        g[n] = 0;
        countingSort(SA, g, t);
        countingSort(SA, g, 0);
        tg[SA[0]] = 1;
```

```
        for(int i = 1 ; i < n; i++) {
            int bigger = g[SA[i - 1]] < g[SA[i]] || g[SA[i - 1] + t] < g[SA[i] +
t];
            tg[SA[i]] = tg[SA[i-1]] + bigger;
        }
        if(tg[SA[n - 1]] == n) break; // 속도 향상 주요 컷팅
        g = tg;
    }
    return SA;
}
vector<int> getLCP(string& S, vector<int>& SA){
    int n = (int)S.size();
    vector<int> LCP(n), rank(n);
    for(int i = 0 ; i < n; i++) rank[SA[i]] = i;
    int k = 0;
    for(int i = 0 ; i < n; i++){
        if(k > 0) k--;
        if(!rank[i]) continue;
        int j = SA[rank[i]-1];
        while(i+k < n && j+k < n && S[i+k] == S[j+k]) k++;
        LCP[rank[i]] = k;
    }
    return LCP;
}
```

22 SA(nlg^2n)

```
vector<int> getSA(string& S){
    int n = (int)S.size();
    vector<int> SA(n), g(n+1), tg(n+1);
    for(int i = 0 ; i < n; i++) SA[i] = i;
    for(int i = 0 ; i < n; i++) g[i] = S[i];

    for(int t = 1; t < n; t*=2){
        g[n] = -1;
        auto cmp = [&](int a, int b)->bool{
            if(g[a] == g[b]) return g[a+t] < g[b+t];
            return g[a] < g[b];
        };
        sort(SA.begin(), SA.end(), cmp);
        tg[SA[0]] = 0;
        for(int i = 1 ; i < n; i++) {
            tg[SA[i]] = tg[SA[i-1]] + cmp(SA[i-1], SA[i]);
        }
        if(tg[SA[n - 1]] == n) break; // 속도 향상 주요 컷팅
        g = tg;
    }
    return SA;
}
```

23 KMP

```
vector<int> getPartial(string& S){
    int len = (int)S.size();
    int matched = 0;
    vector<int> pi = vector<int>(len, 0);
    for(int i = 1 ; i < len; i++){
        while(matched > 0 && S[i] != S[matched]) matched = pi[matched - 1];
        if(S[i] == S[matched]) pi[i] = ++matched;
    }
    return pi;
```

```
}
vector<int> KMP(string& A, string& B){
    int matched = 0;
    int aLen = (int)A.size();
    int bLen = (int)B.size();
    vector<int> pi = getPartial(B);
    vector<int> found;
    for(int i = 0 ; i < aLen; i++) {
        while(matched > 0 && A[i] != B[matched]) matched = pi[matched - 1];
        if(A[i] == B[matched]) matched++;
        if(matched == bLen) found.push_back(i - matched + 1), matched =
pi[matched - 1];
    }
    return found;
}
```

24 Hashing

```
const int base = 257;
ll mod[2] = {(ll)1e9+9999, (ll)1e9+99999}, poww[2];
int main() {
    fastio();
    string A, B;
    getline(cin, A);
    getline(cin, B);
    int aLen = (int)A.size();
    int bLen = (int)B.size();
    for(int k = 0 ; k < 2; k++){
        poww[k] = 1;
        for(int i = 0 ; i < bLen - 1; i++) {
            poww[k] = (poww[k] * base) % mod[k];
        }
    }
    ll h[2] = {0, 0};
    for(int k = 0 ; k < 2; k++) {
        for(int i = 0 ; i < bLen; i++) {
            h[k] = (h[k] * base + B[i]) % mod[k];
        }
    }
    ll acc[2] = {0, 0};
    for(int k = 0 ; k < 2; k++){
        for(int i = 0 ; i < bLen; i++){
            acc[k] = (acc[k] * base + A[i]) % mod[k];
        }
    }
    vector<int> ans;
    for(int i = bLen; i <= aLen; i++){
        if(acc[0] == h[0] && acc[1] == h[1]) ans.push_back(i - bLen);
        if(i == aLen) break;
        for(int k = 0 ; k < 2; k++){
            acc[k] = (acc[k] - A[i - bLen] * poww[k] + base * mod[k]) %
mod[k];
            acc[k] = (acc[k] * base + A[i]) % mod[k];
        }
    }
    printf("%d₩n", (int)ans.size());
    for(int a : ans) printf("%d₩n", a+1);
    return 0;
}
```

25 Aho Crosick

```
const int MAX_C = 26;
class AhoCorasick{
public:
    struct trie{
        trie* next[MAX_C];
        trie* fail;
        int output;
        trie(){
            for(int i = 0; i < MAX_C; i++) next[i] = NULL;
            fail = NULL;
            output = 0;
        }
        ~trie(){
            for(int i = 0; i < MAX_C; i++){
                if(next[i]) delete next[i];
            }
        }
        void insert(string& s, int index){
            trie *curr = this;
            int sz = s.size();
            for(int i = 0; i < sz; i++){
                int v = s[i] - 'a';
                if(!curr->next[v]) curr->next[v] = new trie();
                curr = curr->next[v];
            }
            curr->output = index;
        }
    };
    trie *root;
    AhoCorasick(vector<string>& arr){
        root = new trie();
        root->fail = root;
        int sz = arr.size();
        for(int i = 0; i < sz; i++) root->insert(arr[i], i + 1);
        queue<trie*> q;
        q.push(root);
        while(!q.empty()){
            trie *curr = q.front();
            q.pop();
            for(int i = 0; i < MAX_C; i++){
                trie *next = curr->next[i];
                if(!next) continue;
                trie *dest = curr->fail;
                while(dest != root && !dest->next[i]) dest = dest->fail;
                if(curr != root && dest->next[i]) dest = dest->next[i];
                next->fail = dest;
                if(next->fail->output) next->output = next->fail->output;
                q.push(next);
            }
        }
    }
    ~AhoCorasick(){
        delete root;
    }
    bool isExist(string &s){
        trie *curr = root;
        int sz = s.size();
```

```
        for(int i = 0; i < sz; i++){
            int v = s[i] - 'a';
            while(curr != root && !curr->next[v]) curr = curr->fail;
            if(curr->next[v]) curr = curr->next[v];
            if(!curr->output) continue;
            return true;
        }
        return false;
    }
    vector<pair<int, int>> getPos(string &s){
        vector<pair<int, int>> res;
        trie *curr = root;
        int sz = s.size();
        for(int i = 0; i < sz; i++){
            int v = s[i] - 'a';
            while(curr != root && !curr->next[v]) curr = curr->fail;
            if(curr->next[v]) curr = curr->next[v];
            if(!curr->output) continue;
            res.push_back({i, curr->output});
        }
        return res;
    }
};
```

26 Z algorithm(+pattern search)

```
struct Z{
    vector<int> za;
    Z(string S){
        int N = (int)S.size();
        za = vector<int>(N);
        int L = -1, R = -1;
        for(int i = 1; i < N; i++){
            if(i <= R){
                if(za[i - L] < R - i + 1) za[i] = za[i - L];
                else {
                    L = i;
                    while(R < N && S[R] == S[R - i]) R++;
                    za[i] = R - L; R--;
                }
            }
            else {
                L = R = i;
                while(R < N && S[R] == S[R - i]) R++;
                za[i] = R - L; R--;
            }
        }
    }
};
vector<int> findPos(string& S, string& P){
    vector<int> ret;
    string S2 = P + '#' + S;
    Z z(S2);
    for(int i = 0 ; i < sz(S2); i++) if(z.za[i] == sz(P)) ret.pb(i - sz(P) - 1);
    return ret;
}
```

27 Manacher

```
/* res[i] = i번째 문자를 중심으로 하는 팰린드롬의 반지름(문자열은 0부터 시작한다고 가정)
a # b # a # b # b #
0 0 2 0 3 0 1 2 1 0*/
vector<int> manacher(string &S){
    int sLen = (int)S.size();
    vector<int> res(sLen, 0);
    int r = 0, c = 0;
    for(int i = 0; i < sLen; i++){
        if(i < r) res[i] = min(r - i, res[c * 2 - i]);
        while(i - res[i] - 1 >= 0 && i + res[i] + 1 < sLen && S[i - res[i] - 1]
== S[i + res[i] + 1]) res[i]++;
        if(r < i + res[i]){
            r = i + res[i];
            c = i;
        }
    }
    return res;
}
```

28 Convex Hull

```
/*addPoint 함수로 점들을 추가하고 makeConvex 함수를 호출하면
poly 멤버 변수에 컨벡스 헐을 구성하는 점들이 들어간다.*/
struct Point{
    ll x, y;
    Point operator +(Point& rhs){
        return Point{x + rhs.x, y + rhs.y};
    }
    Point operator -(Point& rhs){
        return Point{x - rhs.x, y - rhs.y};
    }
    bool operator <(Point& rhs){
        if(x != rhs.x) return x < rhs.x;
        return y < rhs.y;
    }
    ll cross(Point rhs){
        return x * rhs.y - y * rhs.x;
    }
    ll size(){
        return x * x + y * y;
    }
};
int ccw(Point a, Point b, Point c){
    ll cw = (c - a).cross(c - b);
    if(!cw) return 0;
    cw /= abs(cw);
    return cw;
}
struct Convexhull{
    vector<Point> poly, ps;
    void addPoint(ll x, ll y){
        ps.pb(Point{x, y});
    }
    void makeConvex(){
        int n = sz(ps);
        sort(all(ps));
        Point st = ps[0];
        ps.erase(ps.begin());
```

```
        auto cmp = [&](Point& a, Point& b){
            int cw = ccw(a, b, st);
            if(!cw) (a - st).size() < (b - st).size();
            return cw > 0;
        };
        sort(all(ps), cmp);
        vector<Point> poly;
        poly.pb(st);
        for(int i = 0; i < sz(ps); i++){
            while(sz(poly) >= 2 && ccw(poly.back(), ps[i], poly[sz(poly) - 2])
<= 0) poly.pop_back();
            poly.pb(ps[i]);
        }
        this->poly = poly;
    }
};
int N;
int main(){
    fastio();
    cin >> N;
    Convexhull convex;
    for(int i = 0; i < N; i++){
        ll x, y;
        cin >> x >> y;
        convex.addPoint(x, y);
    }
    convex.makeConvex();
    auto poly = convex.poly;
    if(sz(poly) <= 2) return !printf("0");
    cout << sz(poly);
}
```

29 Line Overlap

```
struct Point{
    ll x, y;
    Point operator-(Point rhs){
        return Point{x - rhs.x, y - rhs.y};
    }
    ll cross(Point rhs){
        return x * rhs.y - y * rhs.x;
    }
};
struct Line{
    Point p1, p2;
};
int ccw(Point a, Point b, Point c){
    ll cw = (a - c).cross(b - c);
    if(!cw) return 0;
    cw /= abs(cw);
    return (int)cw;
}
int isOver(Line l1, Line l2){
    int cw1 = ccw(l2.p1, l1.p2, l1.p1);
    int cw2 = ccw(l1.p2, l2.p2, l1.p1);
    int cw3 = ccw(l1.p2, l2.p2, l2.p1);
    int cw4 = ccw(l2.p2, l1.p1, l2.p1);
    if(cw1 * cw2 > 0 && cw3 * cw4 > 0) return 1;
    return 0;
}
```