Contents

Template

```
//Open PC^2
// Create Project , Auto complete , Enable Save build
// Wrtie Template
// put Samples in Same Folder
// Auto complete, save , ctrl +shift + F ( make format )
//Ctrl + i - Corrects indentation of the selected text - very useful in my
opinion
//enable-C++14 Project Prefrence>Setting>GCC C++>Misecellenaus > -std=c++11
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define endl "\n"
#define watch(x) cout << (#x) << " is " << (x) << endl
int dx[] = { 0, 0, 1, -1, 1, -1, 1, -1 };
int dy[] = { 1, -1, 0, 0, -1, 1, 1, -1 };
// horse moves
int lx[] = { 2, 2, -1, 1, -2, -2, -1, 1 };
int ly[] = { -1, 1, 2, 2, 1, -1, -2, -2 };
// inf
int inf = 0x3f3f3f3f;
ll llinf = 0x3f3f3f3f3f3f3f3f;
int main() {
       std::ios_base::sync_with_stdio(0);
       cin.tie(NULL);  cout.tie(NULL);
       #ifndef ONLINE_JUDGE
       freopen("input.txt", "r", stdin);
       #endif
       int t, nw = 1;  cin >> t;
       while (t--){
              cout << "Case " << nw++ << " : ";
       }
}
```

Graph

DSU

```
const int MAX = 1e5 + 10;
int n, par[MAX], rnk[MAX];
int find(int ind)
{
	if (par[ind] == ind) return ind;
	return par[ind] = find(par[ind]);
}
void merge(int x, int y)
{
	int xRoot = find(x);
	int yRoot = find(y);
	if (rnk[xRoot] > rnk[yRoot])
		par[yRoot] = xRoot;
	if (rnk[xRoot] < rnk[yRoot])
		par[xRoot] = yRoot;
	if (rnk[xRoot] == rnk[yRoot])
	{
		par[xRoot] = yRoot;
		rnk[yRoot]++;
	}
	return;
}
```

LCA

```
const int N = 1e6 + 10, M = 22;
// par[j][i]    = the (2^j)-th ancestor of node number i.
// dis[i]       = the distance to reach node i from the root (node 1).
// LOG[i]       = floor(log2(i)).
int n, m, u, v, dis[N], par[M][N], LOG[N];
vector<int> adj[N];
// Depth first traversal on the tree to fill par[j][i] and dis[i] arrays
// with the appropriate values .
// O(n.log(n))
void dfs(int u = 0, int p = 0, int lvl = 0) {
	//cout << u << " " << lvl << " " << p << endl;
	dis[u] = lvl; par[0][u] = p;
	for (int i = 1; (1 << i) <= lvl; ++i) {
		par[i][u] = par[i - 1][par[i - 1][u]];
	}
	for (int v : adj[u]) {
		if (v != p)
			dfs(v, u, lvl + 1);
	}
}
// Computes the floor of the log of integer from 1 to n.
// After calling this function, LOG[i] will be equals to floor(log2(i)).
// O(n)
void computeLog() {
	LOG[0] = -1;
	for (int i = 1; i < N; ++i) {
		LOG[i] = LOG[i - 1] + !(i & (i - 1));
	}
}
// Builds the LCA structure.
// O(n.log(n))
void buildLCA() {
	dfs();
	computeLog();
}
// Returns the k-th ancestor of node u.
// O(log(k))
int getAncestor(int u, int k) {
	while (k > 0) {
		int x = k & -k;
		k -= x;
		u = par[LOG[x]][u];
	}
	return u;
}
// Returns the lowest common ancestor of nodes u and v.
// O(log(n))
int getLCA(int u, int v) {
	if (dis[u] > dis[v]) {
		swap(u, v);
	}
	v = getAncestor(v, dis[v] - dis[u]);
	if (u == v)
		return u;
	for (int i = LOG[dis[u]]; i >= 0; --i) {
		if (par[i][u] != par[i][v]) {
			u = par[i][u]; v = par[i][v];
		}
	}
	return par[0][u];
}
// Returns the distance between any given pair of nodes
// O(getLCA(u, v)) = O(log(log(n)))
int getDistance(int u, int v) {
	int l = getLCA(u, v);
	return dis[u] + dis[v] - 2 * dis[l];
}
```

Tarjan

```
const int N = 1e6 + 2;
int n, m, component[N], cur_index;
int inde[N], low[N];
bool instack[N];
vector<int> stk, adj[N];
vector< vector<int> > comps;
void tarjan(int ind)
{
	inde[ind] = cur_index;
	low[ind] = cur_index;
	cur_index++;
	stk.push_back(ind);
	instack[ind] = 1;
	for (int i = 0; i<adj[ind].size(); i++)
	{
		int j = adj[ind][i];
		if (inde[j] == -1)
		{
			tarjan(j);
			low[ind] = min(low[ind], low[j]);
		}
		else if (instack[j]) low[ind] = min(low[ind], inde[j]);
	}
	if (inde[ind] == low[ind])
	{
		vector<int> temp;
		while (stk.back() != ind)
```

```
        {
            temp.push_back(stk.back());
            instack[stk.back()] = 0;
            component[stk.back()] = comps.size();
            stk.pop_back();
        }
        temp.push_back(stk.back());
        instack[stk.back()] = 0;
        component[stk.back()] = comps.size();
        stk.pop_back();
        comps.push_back(temp);
    }
    return;
}

void SCC()
{
    memset(inde, -1, sizeof inde);
    memset(low, 0, sizeof low);
    comps.clear(); cur_index = 0;
    for (int i = 1; i <= n; i++)
        if (inde[i] == -1) tarjan(i);
    return;
}
```

Articulation Point and Bridge

```
const int MAX_N = 1e3;
int dfsl[MAX_N], dfsn[MAX_N];
int dfscnt;
vector<int>adj[MAX_N];
int cnty;
set<int>cutpoint;
set<pair<int, int>>bridge;
void mem()
{
    for (auto& v : adj)
        v.clear();
    memset(dfsl, -1, sizeof dfsl);
    memset(dfsn, -1, sizeof dfsn);
    bridge.clear();
    cutpoint.clear();
    dfscnt = 0;
}
void dfs(int node, int p = -1) {
    dfsn[node] = dfsl[node] = dfscnt++;
    int child = 0;
    for (auto &e : adj[node]) {
        if (e == p) continue;
        if (dfsn[e] != -1)
            dfsl[node] = min(dfsl[node], dfsn[e]);
        else {
            dfs(e, node);
            dfsl[node] = min(dfsl[node], dfsl[e]);
            if (dfsl[e] >= dfsn[node] && p != -1)
                cutpoint.insert(node);
            child++;
            if (dfsl[e] > dfsn[node])
                bridge.insert({ node, e });
        }
    }
    if (p == -1 && child > 1)
        cutpoint.insert(node);
}
```

Bellmen ford

```
const int N = 1e3 + 10;
vector<vector<pair<int, int>>>adj(N);
int bellmanford(int n, int src, int dest) {
	vector<int>dist(n + 1, 1e9), prev(n + 1, -1);
	dist[src] = 0; prev[src] = -1;
	int modified = 1;
	for (int i = 0; modified&&i < n - 1; i++) {
		modified = 0;
		for (int i = 0; i < n; i++) { // iterate over graph o(E)
			for (auto nxt : adj[i]) {
				if (dist[i] != 1e9&&dist[i] + nxt.second < 
dist[nxt.first]) {
					dist[nxt.first] = dist[i] + nxt.second;
					prev[nxt.first] = i;
					modified = 1;
				}
			}
		}
	}
	//cout << dist[dest] << endl;
	int has_negcycle = -1;
	for (int i = 0; i < n; i++) {
		for (int i = 0; i < n; i++) { // iterate over graph o(E)
			for (auto nxt : adj[i]) {
				if (dist[i] != 1e9&&dist[i] + nxt.second < 
dist[nxt.first]) {
					dist[nxt.first] = -1e9;
					prev[nxt.first] = i;
					has_negcycle = nxt.first;
				}
			}
		}
	}
	return has_negcycle != -1;
	if (has_negcycle == -1) {
		cout << "No negative cycles\n";
	}
	else {
		int end = has_negcycle;
		for (int i = 0; i < n; i++) {
			end = prev[end];
		}
		vector<int> path;
		for (int cur = end;; cur = prev[cur])
		{
			path.push_back(cur);
			if (cur == end && path.size() > 1)
				break;
		}
		reverse(path.begin(), path.end());
		cout << "Negative cycle: ";
		for (int i = 0; i < path.size(); ++i)
			cout << path[i] << ' ';
	}
	return has_negcycle;
}
```

Floyed warshall

```
void warshall()
```

```
{
    for (int k = 1; k <= n; k++)
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
    return;
}
```

Dynamic Programming

LIS

```
// Returns the length of the longest increasing subsequence of the array.
// O(n.log(n))
const int N = 2e5 + 100;
int n, a[N];
int getLIS() {
    int len = 0;
    vector<int> LIS(n, 1e9);
    for (int i = 0; i < n; ++i) {
        // To get the length of the longest non decreasing subsequence
        // replace function "lower_bound" with "upper_bound"
        int idx = upper_bound(LIS.begin(), LIS.end(), a[i]) -
LIS.begin();
        LIS[idx] = a[i];
        len = max(len, idx);
    }
    return len + 1;
}
```

LIS NlogN print output

```
struct node {
    int elem, index;
};
inline bool operator<(const node& lhs, const node&rhs) {
    return lhs.elem < rhs.elem;
}
void print(int input[], map<int,int> parent, set<node>s) {
    stack<int>lis;
    int index = s.rbegin()->index;
    int n = s.size();
    while (n--) {
        lis.push(input[index]);
        index = parent[index];
    }
    cout << "LIS is : ";
    while (!lis.empty()) {
        cout << lis.top() << " ";
        lis.pop();
    }
    return;
}
void printLIS(int arr[], int n) {
    set<node>s;
    //parent[i] will store the predecessor of element with index i in the
    // LIS ending at element with index i
    map<int, int>parent;
    //process every element one by one
    for (int i = 0; i < n; i++) {
        // construct node from current element and its index
        node curr = { arr[i],i };
        //insert current node into the set and get iterator to the
        //inserted node
        auto it = s.insert(curr).first;
        //if the node not inserted at the end, then delete the next node
        if (++it != s.end())
            s.erase(it);
        //get iterator to the current node and update parent
        it = s.find(curr);
        parent[i] = (--it)->index;
    }
    // print LIS using parent map
    print(arr, parent, s);
}
```

Number of unique subsequence of size k

```
ll solution(vector<ll> A, int k){
    const int N = A.size();
    if (N < k || N < 1 || k < 1)
        return 0;
    if (N == k)
        return 1;
    vector<ll> v1(N, 0);
    vector<ll> v2(N, 0);
    vector<ll> v3(N, 0);
    v2[N - 1] = 1;
    v3[A[N - 1] - 1] = 1;
    for (int i = N - 2; i >= 0; i--) {
        v2[i] = v2[i + 1];
        if (v3[A[i] - 1] == 0) {
            v2[i]++;
            v3[A[i] - 1] = 1;
        }
    }
    for (int j = 1; j < k; j++) {
        fill(v3.begin(), v3.end(), 0);
        v1[N - 1] = 0;
        for (int i = N - 2; i >= 0; i--) {
            v1[i] = v1[i + 1];
            v1[i] = v1[i] + v2[i + 1];
            v1[i] = v1[i] - v3[A[i] - 1];
            v3[A[i] - 1] = v2[i + 1];
        }
        v2 = v1;
    }
    return v1[0];
}
ll solve(string str, int k){
    vector<ll> v;
    for (int i = 0; i < str.size(); i++)
        v.push_back(str[i] - 'a' + 1);
    return solution(v, k);
}
```

Convex hull trick

```
const int N = 1e7 + 10, M = 5000 + 5, mod = 1e9 + 7;
pair<ll, ll>ar[N];
ll dp[N];
vector<pair<ll, ll>>v;
ll get(int i, ll x)
{
    return v[i].first * x + v[i].second;
}
bool check(pair<ll, ll> v1, pair<ll, ll> v2, pair<ll, ll> v3)
```

```
{
	ll x1 = v1.second - v2.second, x2 = v3.first - v1.first;
	ll y1 = v2.first - v1.first, y2 = v1.second - v3.second;
	return x1 * x2 < y1*y2;
}
int main() {
	fast_in_out();
	//cout << fixed << setprecision(6);
	//kolo ray7 , ya 5raby
	int n;
	cin >> n;
	for (int i = 0; i < n; i++)
		cin >> ar[i].first;
	for (int i = 0; i < n; i++)
		cin >> ar[i].second;
	v.push_back({ ar[0].second, 0 });
	int p = 0;
	for (int i = 1; i < n; ++i)
	{
		while (p + 1 < v.size() && get(p + 1, ar[i].first) < get(p,
ar[i].first))++p;
		dp[i] = get(p, ar[i].first);
		pair<ll, ll>cur = { ar[i].second,dp[i] };
		while (v.size() >= 2 && check(cur, v[v.size() - 1], v[v.size() -
2]))
			v.pop_back();
		if ((v.size() - 1) < p)p = v.size() - 1;
		v.push_back({ ar[i].second, dp[i] });
	}
	cout << dp[n - 1];
	return 0;
}
```

Strings

Mancher

```
// finding all the palindrom substring
for (int i = 0; i < s.size(); i++) {
	int l = i - 1;
	int r = i + 1;
	pln[i][i] = true;
	while (l >= 0 && r < s.size() && s[l] == s[r]) {
		pln[l][r] = true;
		l--; r++;
	}
	l = i;
	r = i + 1;
	while (l >= 0 && r < s.size() && s[l] == s[r]) {
		pln[l][r] = true;
		l--; r++;
	}
}
```

Z-algorithm

```
int z[N];
string s;
void Z_algo(int n)
{
	int L = 0, R = 0;
	for (int i = 1; i < n; i++) {
		if (i > R) {
```

```
                        L = R = i;
                        while (R < n && s[R - L] == s[R]) R++;
                        z[i] = R - L; R--;
                }
                else {
                        int k = i - L;
                        if (z[k] < R - i + 1) z[i] = z[k];
                        else {
                                L = i;
                                while (R < n && s[R - L] == s[R]) R++;
                                z[i] = R - L; R--;
                        }
                }
        }
}
```

String hashing one

```
const int N = 1e6 + 100, p = 31;
const ll mod = 1e9 + 7;
ll P = 31LL, pwP[2][N], invP[2][N];
ll hsh[2][N], MOD[] = { 1000000007LL, 1000000009LL };
ll modx(ll base, ll ex, ll m)
{
        ll ans = 1LL;
        while (ex > 0LL)
        {
                if (ex & 1LL)
                        ans = (ans*base) % m;
                base = (base*base) % m;
                ex = ex >> 1LL;
        }
        return ans;
}

void pre()
{
        invP[0][0] = invP[1][0] = pwP[0][0] = pwP[1][0] = 1LL;
        for (int i = 0; i<2; i++)
        {
                invP[i][1] = modx(P, MOD[i] - 2LL, MOD[i]);
                pwP[i][1] = P;
        }
        for (int i = 2; i<N; i++)
        {
                for (int j = 0; j<2; j++)
                {
                        pwP[j][i] = (pwP[j][i - 1] * P) % MOD[j];
                        invP[j][i] = (invP[j][i - 1] * invP[j][1]) % MOD[j];
                }
        }
        return;
}

void calc_hsh(string t)
{
        hsh[0][0] = hsh[1][0] = 0LL;
        for (int i = 1; i <= t.size(); i++)
                for (int j = 0; j<2; j++)
                        hsh[j][i] = (hsh[j][i - 1] + (t[i - 1] - 'a' +
1LL)*pwP[j][i - 1]) % MOD[j];

}
```

```
pair<ll, ll> sub_hsh(int l, int r)
{
        pair<ll, ll> ans;
        ans.first = ((hsh[0][r] - hsh[0][l - 1] + 2LL * MOD[0])*invP[0][l - 1])
% MOD[0];
        ans.second = ((hsh[1][r] - hsh[1][l - 1] + 2LL * MOD[1])*invP[1][l - 1])
% MOD[1];
        return ans;
}
```

String hashing 2

```
char str[N];
int l;
unsigned ll hashArr[N];
unsigned ll power[N];
const int base = 29;
bool res[N];
void pre() {
        for (int i = 1; i<2 * l; i++) {
                hashArr[i] = hashArr[i - 1] * base + str[i] - 'a' + 1;
        }
}
unsigned ll getHash(int i, int j) {
        return hashArr[j] - hashArr[i - 1] * power[j - i + 1];
}
```

MATH

fib til 72 in o(1)

```
ld sq = sqrt(5);
ll fib(ll n) {
        return  (pow((1 + sq) / 2., n) - pow((1 - sq) / 2., n)) / sq;
}
```

Matrix Power

```
#define MAX_N 9 // Fibonacci matrix, increase/decrease this value as needed
struct Matrix { ll m[MAX_N][MAX_N]; };
// we will return a 2D array
Matrix matMul(Matrix a, Matrix b) { // O(n^3)
        Matrix ans; int i, j, k;
        for (i = 0; i < MAX_N; i++)
                for (j = 0; j < MAX_N; j++)
                        for (ans.m[i][j] = k = 0; k < MAX_N; k++) {
                // if necessary, use
                                ans.m[i][j] += (a.m[i][k] * b.m[k][j]) % mod;
                                ans.m[i][j] %= mod;
                        }
        return ans;
}
Matrix matPow(Matrix base, ll p) {// O(n^3 log p)
        Matrix ans; int i, j;
        for (i = 0; i < MAX_N; i++)
                for (j = 0; j < MAX_N; j++)
                        ans.m[i][j] = (i == j);
        // prepare identity matrix
        while (p) { // iterative version of Divide & Conquer exponentiation
                if (p & 1) ans = matMul(ans, base);
                // if p is odd (last bit is on)
                base = matMul(base, base); // square the base
                p >>= 1; // divide p by 2
        }
        return ans;
}
```

Mod inverse

```
const int N = 1e5 + 100;
const int mod = 1e9 + 7;
ll inv[N];
void inverse() {
        inv[0] = inv[1] = 1;
        for (int i = 2; i < N; i++) {
                inv[i] = (mod - (mod / i) * inv[mod % i] % mod) % mod;
        }
}
```

Moebius

```
// Returns value of mobius()
int mobius(int n){
        int p = 0;
        // Handling 2 separately
        if (n % 2 == 0){
                n = n / 2;
                p++;
                // If 2^2 also divides N
                if (!(n&1))
                        return 0;
        }
        // Check for all other prine factors
        for (int i = 3; i <= sqrt(n); i = i + 2){
                // If i divides n
                if (n%i == 0){
                        n = n / i;
                        p++;
                        // If i^2 also divides N
                        if (n % i == 0)
                                return 0;
                }
        }
        return (p % 2 == 0) ? -1 : 1;
}
```

Sieve Moebius

```
const int MAX = 1e5;
int moebius[MAX];
bool prime[MAX];
ll sieve_moebius(ll n){
        memset(prime, 1, sizeof prime);
        memset(moebius, -1, sizeof moebius);
        moebius[0] = moebius[1] = prime[0] = prime[1] = 0;
        for (ll i = 2; i <= MAX; i++){
                if (prime[i]){
                        moebius[i] = 1;
                        for (ll j = i + i; j <= MAX; j += i){
                                prime[j] = 0;
                                moebius[j] = j % (i*i) == 0 ? 0 : -moebius[j];
                        }
                }
        }
}
```

NCR & NPR

```
ll fact[N], inv[N], invfact[N];
```

```
int npr(int n, int r) {
	if (n < r) return 0;
	return fact[n] * invfact[n - r] % mod;
}
int ncr(int n, int r) {
	if (n < r) return 0;
	return (fact[n] * invfact[r] % mod) * invfact[n - r] % mod;
}
void pre() {
	fact[0] = inv[1] = fact[1] = invfact[0] = invfact[1] = 1;
	for (long long i = 2; i < N; i++) {
		fact[i] = (fact[i - 1] * i) % mod;
		inv[i] = mod - (inv[mod % i] * (mod / i) % mod);
		invfact[i] = (inv[i] * invfact[i - 1] % mod);
	}
}
```

Fast Power

```
ll fast_power(ll base, ll power) {
	ll result = 1;
	while (power > 0) {

		if (power & 1) {
			result = (result*base) % mod;
		}
		base = (base * base) % mod;
		power >>= 1;
	}
	return result;
}
```

GCD & LCM

```
ll gcd(ll a, ll b)
{
	if (!b)return a;
	return gcd(b, a%b);
}
ll lcm(ll a, ll b)
{
	ll temp = gcd(a, b);

	return temp ? (a / temp * b) : 0;
}
```

Get & set bits

```
int getbit(int mask, int ind)
{
	return((1 << ind)&mask) == (1 << ind);
}
int setbit0(int mask, int ind)
{
	return mask & ~(1 << ind);
}
int setbit1(int mask, int ind)
{
	return mask | (1 << ind);
}
```

Merge Sort

```
int arr[200001];
ll solve2(int s, int mid, int e){
	ll ret = 0;
```

```
        vector<int>l, r;
        int c = s;
        while (c <= mid)
                l.push_back(arr[c++]);
        while (c <= e)
                r.push_back(arr[c++]);
        for (int i = 0, j = 0; i < l.size() || j < r.size();){
                if (i >= l.size()) { arr[s++] = r[j++]; continue; }
                if (j >= r.size()) { arr[s++] = l[i++]; continue; }
                if (l[i] <= r[j]) { arr[s++] = l[i++]; continue; }
                ret += l.size() - i;
                arr[s++] = r[j++];
        }
        return ret;
}
ll solve(int s, int e){
        ll ret = 0;
        if (s < e){
                int mid = (s + e) / 2;
                ret = solve(s, mid) + solve(mid + 1, e) + solve2(s, mid, e);
        }
        return ret;
}
```

Pascal

```
ll psk[M][M];
void fill_psky() {

        for (int i = 0; i < M; i++) { psk[i][0] = 1; psk[i][i] = 1; }
        for (int i = 1; i < M; i++)
                for (int j = 1; j <= i; j++)
                        psk[i][j] = psk[i - 1][j] + psk[i - 1][j - 1];
}
```

Ternary Search

```
long double solve() {
        long double l = 0, r = 50;
        for (int i = 0; i < 10000; ++i) {
                long double mid1 = l + (r - l) / 3;
                long double mid2 = r - (r - l) / 3;
                long double sum1 = calc(mid1);
                long double sum2 = calc(mid2);
                if (sum1 < sum2)    r = mid2;    else    l = mid1;
        }
        return l;
}
```

Data Structure

Segment tree update interval solve one

```
int tree[33554432], n; //2 power (log(10000000)+2)
void update(int ind, int s, int e, int x, int y, int val){
        if (x > e || y < s) return;
        if (s >= x && e <= y)
        {
                tree[ind] += val;
                return;
        }
        int mid = (s + e) / 2;
        update(ind * 2, s, mid, x, y, val);
```

```
		update(ind * 2 + 1, mid + 1, e, x, y, val);
		return;
}
int solve(int ind, int s, int e, int pos){
		if (pos > e || pos < s) return 0;
		if (s == e && s == pos) return tree[ind];
		int mid = (s + e) / 2, left, right;
		left = solve(ind * 2, s, mid, pos);
		right = solve(ind * 2 + 1, mid + 1, e, pos);
		return tree[ind] + left + right;
}
```

Segment tree update one solve interval

```
long long tree[262146]; //2 power (log(100000)+2)
void update(int ind, int s, int e, int pos, int val){
		if (pos > e || pos < s) return;
		if (s == e && s == pos){
			tree[ind] = val;
			return;
		}
		int mid = (s + e) / 2;
		update(ind * 2, s, mid, pos, val);
		update(ind * 2 + 1, mid + 1, e, pos, val);
		tree[ind] = tree[ind * 2] + tree[ind * 2 + 1];
		return;
}
long long solve(int ind, int s, int e, int x, int y){
		if (x > e || y < s) return 0;
		if (s >= x && e <= y) return tree[ind];
		int mid = (s + e) / 2;
		long long left, right;
		left = solve(ind * 2, s, mid, x, y);
		right = solve(ind * 2 + 1, mid + 1, e, x, y);
		return left + right;
}
```

segment tree update interval solve interval

```
const int N = 4e5 + 40, M = 1e5 + 10, mod = 1e9 + 7;
ll tree[N], tmp[N], mn[N], h[M];
ll lft[N];
void pushup(int ind) {
		int l = ind << 1, r = l | 1;
		mn[ind] = min(mn[l], mn[r]);
		lft[ind] = lft[l] + lft[r];
		tree[ind] = tree[l] + tree[r];
}
void pushdown(int ind) {
		if (!tmp[ind])return;
		int l = ind << 1, r = l | 1;
		tmp[l] += tmp[ind];
		tmp[r] += tmp[ind];
		mn[l] += tmp[ind];
		mn[r] += tmp[ind];
		tree[l] += tmp[ind] * lft[l];
		tree[r] += tmp[ind] * lft[r];
		tmp[ind] = 0;
}
void build(int ind, int l, int r) {
		tmp[ind] = 0;
		if (l == r) {
			mn[ind] = h[l]; lft[ind] = 1; tree[ind] = h[l];
			return;
```

```
	}
	int mid = l + r >> 1;
	build(ind << 1, l, mid);
	build(ind << 1 | 1, mid + 1, r);
	pushup(ind);
}
void update(int ind, int s, int e, int x, int y, ll val)
{
	if (x > e || y < s) return;
	if (s >= x && e <= y && mn[ind] - val >= 0)
	{
		mn[ind] -= val;
		tmp[ind] -= val;
		tree[ind] -= lft[ind] * val;
		return;
	}
	if (s == e && mn[ind] - val <= 0) {
		lft[ind] = 0;
		tree[ind] = 0;
		mn[ind] = 1e18;
		return;
	}
	pushdown(ind);
	int mid = s + e >> 1, l = ind << 1;
	update(l, s, mid, x, y, val);
	update(l | 1, mid + 1, e, x, y, val);
	pushup(ind);
	return;
}
long long solve(int ind, int s, int e, int x, int y)
{
	if (x > e || y < s) return 0;
	if (s >= x && e <= y)
		return tree[ind];
	int mid = s + e >> 1, l = ind << 1;
	long long left, right;
	pushdown(ind);
	left = solve(l, s, mid, x, y);
	right = solve(l | 1, mid + 1, e, x, y);
	return left + right;
}
```

segment tree update interval solve interval 2

```
long long tree[270000], tmp[270000];

void update(int ind, int s, int e, int x, int y, int val){
	if (x > e || y < s) return;
	if (s >= x && e <= y){
		tmp[ind] += val;
		tree[ind] += (long long)(e - s + 1) * val;
		return;
	}
	int mid = (s + e) / 2;
	update(ind * 2, s, mid, x, y, val);
	update(ind * 2 + 1, mid + 1, e, x, y, val);
	tree[ind] = tree[ind * 2] + tree[ind * 2 + 1] + (e - s + 1)*tmp[ind];
	return;
}
long long solve(int ind, int s, int e, int x, int y)
{
	if (x > e || y < s) return 0;
	if (s >= x && e <= y)
```

```
                return tree[ind];
        int mid = (s + e) / 2;
        long long left, right;
        left = solve(ind * 2, s, mid, x, y);
        right = solve(ind * 2 + 1, mid + 1, e, x, y);
        return left + right + tmp[ind] * (min(y, e) - max(s, x) + 1);
}
```

BIT

```
int  tree[30001];
int solve(int ind) {
        int ret = 0;
        for (int i = ind; i>0; i -= (i&(-i)))
                ret += tree[i];
        return ret;
}
void update(int ind) {
        if (ind == 0) return;
        for (int i = ind; i <= n; i += (i&(-i)))
                tree[i]++;
        return;
}
```

Mo-algorithm

```
const int N = 1e5 + 100; // don't forget that
// for each query (L, R) to find the number of distinct values in the array
from L to R.
const int  Q = 100100;
int n, m, a[N], cnt[N], ans[N], curL, curR, curAns, blockSize;
struct query {
        int l, r, i;
        bool operator<(const query& rhs) const {
                if (l / blockSize != rhs.l / blockSize) {
                        return l < rhs.l;
                }
                return r < rhs.r;
        }
}queries[Q];
// Inserts the given index into our current answer
void insert(int k) {
        curAns += (++cnt[a[k]] == 1);
}
// Removes the given index from our current answer
void remove(int k) {
        curAns -= (--cnt[a[k]] == 0);
}
// Solves all queries according to Mo's algorithm.
void solveMO() {
        // Set Mo's algorithms variables
        blockSize = sqrt(n) + 1; curL = 0, curR = -1, curAns = 0;
        // Sort queries
        sort(queries, queries + m);
        // Solve each query and save its answer
        for (int i = 0; i < m; ++i) {
                query& q = queries[i];
                while (curL < q.l) remove(curL++);
                while (curL > q.l) insert(--curL);
                while (curR < q.r) insert(++curR);
                while (curR > q.r) remove(curR--);
                ans[q.i] = curAns;
        }
}
```

```
void read() {
        cin >> n >> m;
        for (int i = 0; i < n; ++i) {
                scanf("%d", a + i);
        }
        for (int i = 0; i < m; ++i) {
                query& q = queries[i];
                q.i = i;
                scanf("%d %d", &q.l, &q.r);
        }
        solveMO();
}
```

Ordered Set

```
#include <bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;
template <typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
int main()
{
        ordered_set<int>os;
        *os.find_by_order(index)
                os.order_of_key(x)// the number of element less that x
}
```

Tower Of Hanoi

```
vector<pair<char, char>>v;
void too(int n, char f, char t, char c) {
        if (n == 1) {
                v.push_back({ f,t });
                return;
        }
        too(n - 1, f, c, t);
        v.push_back({ f,t });
        too(n - 1, c, t, f);
}
```

2-sat

```
int n, m;
vector<int> adj[N];
stack<int> st;
pair<int, int> edges[N];
int idx[N], compID[N];
int assign[N], compHead[N];
int low[N], vis[N];
int T, cmp;

void dfs(int u) { // find the comp in reverse topo order
	idx[u] = low[u] = ++T;
	vis[u] = 1;
	st.push(u);
	for (auto e : adj[u]) {
		if (!idx[e]) dfs(e);
		if (vis[e]) low[u] = min(low[u], low[e]);
	}
	if (idx[u] == low[u]) {
		compHead[cmp] = u;
		while (true) {
			int v = st.top();
			st.pop();
			vis[v] = 0;
			compID[v] = cmp;
			if (v == u) break;
		}
		++cmp;
	}
}
int NOT(int u) {
	if (u >= m)  return u - m;
	return u + m;
}
void add(int u, int v) {// add (a v b) and (!a v !b)
	adj[u].emplace_back(NOT(v));
	adj[v].emplace_back(NOT(u));
	adj[NOT(u)].emplace_back(v);
	adj[NOT(v)].emplace_back(u);
}
```