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

**#include <ext/pb\_ds/assoc\_container.hpp>**

**#include <ext/pb\_ds/tree\_policy.hpp>**

**using namespace std;**

**using namespace \_\_gnu\_pbds;**

**typedef tree<int, null\_type, less<int>, rb\_tree\_tag,**

**tree\_order\_statistics\_node\_update>**

**new\_data\_set;**

**int main() {**

**new\_data\_set St;**

**St.insert(1);**

**for (int i = 0; i < St.size(); i++) {**

**cout << i << " -> " << \*St.find\_by\_order(i) << '\n';**

**}**

**for (int i = 0; i < 15; i++) {**

**cout << i << " -> " << St.order\_of\_key(i) << '\n';**

**}**

**}**

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

**#include <ext/pb\_ds/assoc\_container.hpp>**

**#include <ext/pb\_ds/tree\_policy.hpp>**

**using namespace std;**

**using namespace \_\_gnu\_pbds;**

**typedef tree<int, null\_type, less<int>, rb\_tree\_tag,**

**tree\_order\_statistics\_node\_update>**

**new\_data\_set;**

**int main() {**

**int n;**

**cin >> n;**

**int a[n];**

**for (int i = 0; i < n; i++) {**

**cin >> a[i];**

**}**

**int ans = 0;**

**new\_data\_set St;**

**for (int i = 0; i < n; i++) {**

**ans += (i - St.order\_of\_key(a[i]));**

**St.insert(a[i]);**

**}**

**cout << ans;**

**}**

**/\* =============================== \*/**

**/\* Try all 2^n subsets of n items \*/**

**/\* =============================== \*/**

**void all\_subsets(vector<int> items) {**

**int n = vals.size();**

**int times = (1 << n);**

**vector<int> bits(n, 0)**

**while(times-- > 0) {**

**do\_something(bits)**

**// generate next set's bit representation**

**int i = 0, carry = 1;**

**while (i < n) {**

**in[i] += carry;**

**if (in[i] <= 1)**

**carry = 0;**

**else**

**in[i] = 0;**

**i++;**

**}**

**}**

**}**

**/\* ========================================= \*/**

**/\* Split n items into k containers optimally \*/**

**/\* ========================================= \*/**

**int capacities[MAXN];**

**int N;**

**// Return cost of storing n items in i-th container**

**storage\_cost(int i, int n);**

**// Find best way to split n items among containers**

**// from index i to N-1. For simplicity, the total**

**// remaining capacity is carried along.**

**int search\_splits(int i, int n, int tot\_cap) {**

**if (i >= N) return 0;**

**int min\_k = max(0, n - (tot\_cap - capacities[i]));**

**int max\_k = min(n, capacities[i]);**

**int min\_cost = INT\_MAX;**

**rep(k, min\_k, max\_k) {**

**min\_cost = min(min\_cost,**

**storage\_cost(i, k) +**

**search\_splits(i+1, n-k, tot\_cap - capacities[i]);**

**)**

**}**

**}**

**int best\_split(int n) {**

**int tot\_cap = 0;**

**rep(i,0,N-1) tot\_cap += capacities[i];**

**return search\_splits(0,n,tot\_cap);**

**}**

* **Bitset**

**bitset<4> foo; // 0000**

**foo.size(); // 4**

**foo.set(); // 1111**

**foo.set(1,0); // 1011**

**foo.test(1); // false**

**foo.set(1); // 1111**

**foo.test(1); // true**

**// ===============**

**// trial division**

**//=================**

**// complexity: ~O( sqrt(x) )**

**bool isPrime(int x) {**

**for (int d = 2; d \* d <= x; d++) {**

**if (x % d == 0)**

**return false;**

**}**

**return true;**

**}**

* **Merg Sort.**

**void mergesort\_(vector<int>& a, int i, int j, vector<int>& b) {**

**if (i == j) return; // caso base**

**// caso general**

**int m = (i+j)/2;**

**mergesort\_(a, i, m, b);**

**mergesort\_(a, m+1, j, b);**

**// merge eficiente**

**int r = i, s = m+1;**

**int k = i;**

**while (r <= m and s <= j) b[k++] = a[r] <= a[s] ? a[r++] : a[s++];**

**while (r <= m) b[k++] = a[r++];**

**while (s <= j) b[k++] = a[s++];**

**for (int x = i; x <= j; ++x) a[x] = b[x];**

**}**

**void mergesort(vector<int>& a) {**

**vector<int> b(a.size());**

**mergesort\_(a, 0, a.size()-1, b);**

**}**

**// =======================================**

**// trial division with precomputed primes**

**// =======================================**

**// complexity: ~O( sqrt(x)/log(sqrt(x)) )**

**// + time of precomputing primes**

**bool isPrime(int x, vector<int>& primes) {**

**for (int p : primes) {**

**if (p\*p > x) break;**

**if (p % x == 0)**

**return false;**

**}**

**return true;**

**}**

**inline ll mod(ll x, ll m) { return ((x %= m) < 0) ? x+m : x; }**

**/\* ============================= \*/**

**/\* GCD (greatest common divisor) \*/**

**/\* ============================= \*/**

**// OPTION 1: using C++ builtin function \_\_gcd**

**\_\_gcd(a,b)**

**// OPTION 2: manually usings euclid's algorithm**

**int gcd (ll a, ll b) {**

**while (b) { a %= b; swap(a,b); }**

**return a;**

**}**

**//=====================**

**// Prime Factorization**

**//=====================**

**/ complexity: ~ O( sqrt(n) + log\_2(n) )**

**vector<int> trial\_division(int n) {**

**vector<int> factors;**

**for (int d = 2; d\*d <= n; d++) {**

**while (n % d == 0) {**

**factors.push\_back(d);**

**if ((n /= d) == 1) return factors;**

**}**

**}**

**if (n > 1) factors.push\_back(n);**

**return factors;**

**}**

**// compute a^b (mod m)**

**int binary\_exp(int a, int b, int m) {**

**a %= m;**

**int res = 1;**

**while (b > 0) {**

**if (b&1) res = (res \* a) % m;**

**a = (a \* a) % m;**

**b >>= 1;**

**}**

**return res;}**

**//=============================================**

**// Sieve of Eratosthenes (all primes up to N)**

**//=============================================**

**// O ( n log log n )**

**vector<int> get\_primes\_up\_to(int n) {**

**vector<bool> is\_prime(n + 1, true);**

**int limit = floor(sqrt(n));**

**rep (i,2,limit+1)**

**if (is\_prime[i])**

**for (int j = i\*i; j <= n; j += i)**

**is\_prime[j] = false;**

**vector<int> primes;**

**rep(i,2,n) if (is\_prime[i]) primes.push\_back(i);**

**return primes;**

**}**

**void palindrome(int n){**

**int val = n;**

**int num = 0;**

**while(n){**

**int temp = n % 10;**

**num = num \* 10 + temp;**

**n = n / 10;**

**}**

**Cout << (num == val ? "PALINDROME\n" : "NOT A PALINDROME\n");**

**}**

**int palindrome(char\* str, int s, int e){**

**if(s==e)**

**return 1;**

**if(str[s] != str[e])**

**return 0;**

**if (s != e)**

**palindrome(str , s + 1, e - 1);**

**}**

**void swap(char\* str, int n){**

**int s = 0;**

**while(s != n / 2){**

**char l = str[s];**

**str[s] = str[n-1-s];**

**str[n-1-s] = l;**

**s++;**

**}**

**}**

**// It's Power of N**

**bool isPowerOfN(int number, int N) {**

**if(number == 0 || (number!=1 && N == 0)) return false;**

**else if(N == 1) return true;**

**while(number != 1) {**

**if(number % N != 0) {**

**return false;**

**}**

**number = number / N;**

**}**

**return true;**

**}**

**// Sum of Multiples of a number up to N**

**// sum\_multiple(3, 12) = 3+6+9+12 = 30**

**int sum\_multiple(int number, int N) {**

**int m = N / number;**

**int sum = m\*(m + 1) / 2;**

**return number\*sum;**

**}**

**// All divisors of a number**

**template<class T>**

**vector<T> divisors(T number) {**

**vector<T> solutions;**

**for (T i = 1; i <= sqrt(number); ++i) {**

**if (number % i == 0) {**

**if (number/i == i) {**

**solutions.push\_back(i);**

**} else {**

**solutions.push\_back(i);**

**solutions.push\_back(number/i);**

**} } }**

**return solutions;**

**}**

**/\*\***

**\* Description: Calculates n^m (Binary exponentiation)**

**\* Usage: power O(lg(M))**

**\* Source: https://github.com/dragonslayerx**

**\*/**

**#define MOD 1000000007**

**long long power(long long n, long long m){**

**if (m == 0) return 1;**

**long long x = power(n, m / 2);**

**if (!(m & 1)) return (x \* x) % MOD;**

**else return (((x \* x) % MOD) \* n) % MOD;**

**}**

**int merge(int a[], int b[], int l, int r, int mid){**

**int lptr = l;**

**int rptr = mid+1;**

**int current\_ptr = 0;**

**int count = 0;**

**for (int i = l; i <= r; i++) {**

**if (lptr > mid) {**

**b[current\_ptr++] = a[rptr++];**

**count += (mid - lptr + 1);**

**} else if (rptr > r) {**

**b[current\_ptr++] = a[lptr++];**

**} else {**

**if (a[lptr] < a[rptr]) {**

**b[current\_ptr++] = a[lptr++];**

**} else {**

**b[current\_ptr++] = a[rptr++];**

**count += (mid - lptr + 1);**

**}**

**}**

**}**

**for (int i = l, j = 0; i <= r; i++, j++) {**

**a[i] = b[j];**

**}**

**return count;**

**}**

**int merge\_sort(int a[], int b[], int l, int r){**

**if (l == r) {**

**return 0;**

**}**

**int mid = (l + r)/2;**

**int inversion\_left = merge\_sort(a, b, l, mid);**

**int inversion\_right = merge\_sort(a, b, mid+1, r);**

**int inversion\_count = merge(a, b, l, r, mid);**

**return inversion\_left + inversion\_right + inversion\_count;**

**}**

**int main(){**

**int a[N], b[N];**

**int n;**

**cin >> n;**

**for (int i = 0; i < n; i++) {**

**cin >> a[i];**

**//a[i] = rand() % 32;**

**}**

**cout << merge\_sort(a, b, 0, n-1) << endl ;**

**cout << endl;**

**}**

**// Recursive function to return gcd of a and b**

**long long gcd(long long int a, long long int b){**

**if (b == 0)**

**return a;**

**return gcd(b, a % b);**

**}**

**long long lcm(int a, int b){**

**return (a / gcd(a, b)) \* b;**

**}**

**int power(int x, int n)**

**{**

**// If x^0 return 1**

**if (n == 0)**

**return 1;**

**// If we need to find of 0^y**

**if (x == 0)**

**return 0;**

**// For all other cases**

**return x \* power(x, n - 1);**

**}**