Misc

// \_\_int128

inline bll read()

{

    bll x=0, f=1; char c=getchar();

    while (c<'0' || c>'9')

    {

        if (c=='-') f=-1;

        c=getchar();

    }

    while (c>='0' && c<='9')

    {

        x=x\*10+(c-'0');

        c=getchar();

    }

    return x\*f;

}

inline void print(bll x)

{

    if (x<0)

    {

        putchar('-');

        x=-x;

    }

    if (x>9) print(x/10);

    putchar(x%10+'0');

}

// Custom Hash

struct CHASH

{

    int h[N];

    CHASH() { fill(h, h+N, -1); }

    int add(int x)

    {

        int p=x%N;

        while (h[p]!=-1)

        {

            if (h[p]==x) return p;

            p=(p+1)%N;

        }

    }

    int find(int x)

    {

        int p=x%N;

        while (h[p]!=-1)

        {

            if (h[p]==x) return p;

            p=(p+1)%N;

        }

    }

} \*chash;

// Discretization

void discretize()

{

    vector<int> t; for (int i=0; i<n; i++) t.push\_back(s[i]);

    sort(t.begin(), t.end());

    t.erase(unique(t.begin(), t.end()), t.end());

    for (int i=0; i<n; i++) s[i]=lower\_bound(t.begin(), t.end(), s[i])-t.begin();

}

// Optimizations

#define getchar() getchar\_unlocked()

#pragma GCC optimize("O3,unroll-loops")

#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")

using namespace std;

inline int read()

{

    int x=0, f=1; char c=getchar();

    while (c<'0' || c>'9')

    {

        if (c=='-') f=-1;

        c=getchar();

    }

    while (c>='0' && c<='9')

    {

        x=x\*10+(c-'0');

        c=getchar();

    }

    return x\*f;

}

inline void print(int x)

{

    if (x<0)

    {

        putchar('-');

        x=-x;

    }

    if (x>9) print(x/10);

    putchar(x%10+'0');

}

// PBDS

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

using namespace \_\_gnu\_pbds;

#define pbds\_map gp\_hash\_table<int, int>;

#define pbds\_set tree<int, null\_type, less<int>, rb\_tree\_tag, tree\_order\_statistics\_node\_update>;

#define pbds\_multiset tree<int, null\_type, less\_equal<int>, rb\_tree\_tag, tree\_order\_statistics\_node\_update>; // cannot erase()

// Stress Testing

#include <chrono>

#include <random>

mt19937\_64 rng(chrono::steady\_clock::now().time\_since\_epoch().count());

struct WRONG

{

    signed main() {}

} \*wrong;

struct CORRECT

{

    signed main() {}

} \*correct;

signed main()

{

    for (int \_=0; \_<100013; \_++)

    {

        wrong=new WRONG; correct=new CORRECT;

        int out\_wrong=wrong->main();

        int out\_correct=correct->main();

        if (out\_wrong!=out\_correct)

        {

            cerr<<endl;

            cerr<<"WRONG: "<<out\_wrong<<endl;

            cerr<<"CORRECT: "<<out\_correct<<endl;

            return 0;

        }

        cerr<<"PASSED "<<\_<<endl;

    }

    cerr<<"PASSED ALL"<<endl;

    return 0;

}