```cpp
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <vector>
using namespace std;

using LL = long long;
const int MOD = 1e9 + 7;
const int MAXN = 1e5 + 5;
// Euler sieve
int prime[MAXN * 10], cnt_prime;
bool noprime[MAXN * 10];
int mu[MAXN * 10];
vector<int> nozero_mu;
void Euler_sieve(int n);
// main
int N, M, K;
int a[MAXN], b[MAXN], p[MAXN], coef[MAXN * 10];
LL discret[MAXN];
vector<int> fac[MAXN * 10], buck[MAXN * 10];
int solve(int d);

int main() {
  //
  cin >> N;
  for (int i = 1; i <= N; i++) {
    cin >> a[i];
    discret[i] = 1LL * (N + 1) * a[i] + N - i;
  }
  sort(discret + 1, discret + 1 + N);
  // K = unique(discret + 1, discret + 1 + N) - discret - 1;
  K = N;
  for (int i = 1; i <= N; i++)
    a[i] = lower_bound(discret + 1, discret + 1 + K,
                       1LL * (N + 1) * a[i] + N - i) -
           discret;
  // for (int i = 1; i <= N; i++) cout << a[i] << endl;

  int maxb = 0;
  for (int i = 1; i <= N; i++) {
    cin >> b[i];
    buck[b[i]].push_back(i);
    maxb = max(maxb, b[i]);
  }
  Euler_sieve(maxb + 5);
  for (int i = 1; i <= maxb; i++) {
    for (int k : nozero_mu) {
      int j = i * k;
      if (j > maxb) break;
      coef[j] = (1LL * coef[j] + mu[k] * i + MOD) % MOD;
```

```
    }
  }

  for (int i = 1; i <= maxb; i++) {
    for (int j = i; j <= maxb; j += i)
      for (int id : buck[j]) fac[i].push_back(id);
    sort(fac[i].begin(), fac[i].end());
  }

  int ans = 0;
  for (int i = 1; i <= maxb; i++)
    if (coef[i]) {
      int tmp = 1LL * coef[i] * solve(i) % MOD;
      // cout << solve(i) << endl;
      ans = (ans + tmp) % MOD;
    }
  cout << ans << endl;
  return 0;
}

class FenwickTree {
 private:
  int n;
  int a[MAXN];
  inline int lowbit(int x) { return x & (-x); }

 public:
  void init(int n0);
  void add(int pos, int key);
  int query(int pos);
};
FenwickTree FT;
int solve(int d) {
  M = 0;
  for (int id : fac[d]) {
    p[++M] = a[id];
    discret[M] = p[M];
  }
  sort(discret + 1, discret + 1 + M);
  for (int i = 1; i <= M; i++)
    p[i] = lower_bound(discret + 1, discret + 1 + M, p[i]) - discret;
  int cnt = 0;
  FT.init(M);
  for (int i = 1; i <= M; i++) {
    int tmp = FT.query(p[i]);
    cnt = (cnt + tmp + 1) % MOD;
    FT.add(p[i], tmp + 1);
  }
  return cnt;
}
```

```
void FenwickTree::init(int n0) {
  n = n0;
  fill(a, a + n + 3, 0);
}

void FenwickTree::add(int pos, int key) {
  while (pos <= this->n) {
    this->a[pos] = (this->a[pos] + key) % MOD;
    pos += lowbit(pos);
  }
}

int FenwickTree::query(int pos) {
  int res = 0;
  while (pos) {
    res = (res + this->a[pos]) % MOD;
    pos -= lowbit(pos);
  }
  return res;
}

void Euler_sieve(int n) {
  mu[1] = 1;
  for (int i = 2; i <= n; i++) {
    if (!noprime[i]) {
      prime[++cnt_prime] = i;
      mu[i] = -1;
    }
    for (int j = 1; j <= cnt_prime && i * prime[j] <= n; j++) {
      noprime[i * prime[j]] = true;
      if (i % prime[j] == 0) break;
      mu[i * prime[j]] = -mu[i];
    }
  }
  for (int i = 1; i <= n; i++)
    if (mu[i]) nozero_mu.push_back(i);
}
```