Cambios

Autor: Manuel Torres Cid

Ordenemos el vector de c de forma creciente e iteramos por este, mantenemos dos conjuntos A y B, en la iteración i-ésima del vector c, A contiene los indices j de forma que $a_j \leq c_i$, y B, contiene los indices j de forma que $b_j \leq c_i$, observemos que para emparejar el valor c_i, con alguna pareja a_j, b_j, primero tendremos que intentar emparejar algun elemento de A (coste 0) y si no podemos intentaremos usar un elemento de B, si tomamos un indice j de A, cualquiera es válido, ya que el indice j seguirá siendo valido para $c_{i+1}, c_{i+2}, \ldots$. Si tomamos un indice j de B, debemos tomar aquel que tenga un valor a_j mayor, ya que nos dejará en una mejor posicion para procesar $c_{i+1}, c_{i+2}, \ldots$, la razón es que todos los indices de B seguirán siendo validos en $c_{i+1}, c_{i+2}, \ldots$, pero con coste 1, pero sus valores de a correspondientes no estan en A, por lo que eliminamos el índice j con un valor a_j mayor ya que es el índice que más "tarde"se insertará en A.

Para calcular todo esto podemos ordenar a, b, y c (sin perder los índices originales de a y b, podeis crear dos vectores de pairs donde el i-ésimo valor son $\{a_i, i\}$ y $\{b_i, i\}$ y ordenarlos), guardar punteros sobre estos vectores para saber cuando insertarlos en A y B, y implementar Ay B como dos priority queues de pairs, donde ser guarda $\{a_j, j\}$ en ambas colas (en ambas colas se ordenan los indices por valor de a).

```cpp
#include <bits/stdc++.h>
using namespace std;
#define int long long

int test_case() {
  int ans = 0;
  int n;
  cin >> n;
  // Nos guardamos los valores y su indice
  vector<pair<int, int>> a(n), b(n), copya(n);;
  vector<int> c(n);
  vector<bool> used(n, false);
  for (int i = 0; i < n; i++) {
    cin >> a[i].first;
    a[i].second = i;
  }
  for (int i = 0; i < n; i++) {
    cin >> b[i].first;
    b[i].second = i;
  }
  for (int i = 0; i < n; i++)
    cin >> c[i];
  copya = a;
  // ordenamos todo
  sort(a.begin(), a.end());
  sort(b.begin(), b.end());
  sort(c.begin(), c.end());
  // Mantenemos estas priority_queues a lo largo
    de todo el algoritmo
  // Al iterar c_i, am contiene los indices j de
    los elementos de a que
  // no hemos usado y que a_j <= c_i. bm contiene
    los indices j
  // de los elementos de a que no hemos usado y
    b_j <= c_i
  // ordenaos por a_j.
  // Como mucho introducimos cada elemento de a y
    b una vez en am y bm
  priority_queue<int> am;
  priority_queue<pair<int, int>> bm;
  int ita = 0, itb = 0;
  for (int i = 0; i < n; i++) {
    bool paired = false; // Si hemos emparejado a
      c_i
    while (ita < n and a[ita].first <= c[i]) {
      am.push(a[ita].second);
      ita++;
    }
    while (itb < n and b[itb].first <= c[i]) {
      bm.push(make_pair(copya[b[itb].second].first
      , b[itb].second));
      itb++;
    }
    // quitamos los usados
    while (not am.empty() and used[am.top()]) {
      am.pop();
    }
    while (not bm.empty() and used[bm.top().second
    ]) {
      bm.pop();
    }
    if (not am.empty()) {
      used[am.top()] = true;
      paired = true;
    }
    if (not bm.empty() and not paired) {
      used[bm.top().second] = true;
      paired = true;
      ans++; // usar un elemento de bm tiene coste
       1
    }
    if (not paired) {
      return -1;
    }
  }
  return ans;
}
signed main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  int T;
  cin >> T;
  while (T--) {
    cout << test_case() << "\n";
  }
  return 0;
}
```