Problem Analysis: Strong Vertices in a Directed Graph

Problem Statement

Given two arrays a and b of length n, construct a directed graph where an edge from vertex u to v ($u \neq v$) exists if:

au−av≥bu−bv

$au - av \geq bu - bv$

A vertex V is **strong** if there exists a path from V to all other vertices. The task is to find all strong vertices.

Input

- t test cases.
- For each test case:
  - n: Length of arrays.
  - Array a of integers.
  - Array b of integers.

Output

For each test case:

1. The number of strong vertices.
2. Their indices in ascending order.

Key Insight

The edge condition simplifies to:

$au-bu \geq av-bv$

$au-bu \geq av-bv$

Define $ci=ai-bi$

$ci=ai-bi$.

Then:

- An edge u → v exists if $cu \geq cvcu \geq cv$.
- A vertex V is strong if $cVcV$ is the **maximum** in c, because:
    - $cV \geq cicV \geq ci$ for all i ⇒ edges V → i exist.
    - Thus, V can reach all vertices directly.

Why This Works

- **Maxima in** c: Only vertices with maximum $cici$ can have outgoing edges to all others.
- **Multiple Maxima**: If multiple vertices share the max $cici$, all are strong since they can reach each other and every other vertex.

Solution Code

```
// Najmul Huda
#include <bits/stdc++.h>
using namespace std;
#define all(x) x.begin(), x.end()
#define SET(arr, a) memset(arr, a, sizeof(arr))
#define FOR(i, a, b) for (int i = a; i <= b; ++i)
#define ROF(i, a, b) for (int i = a; i >= b; --i)
#define yes cout << "YES" << endl;
#define no cout << "NO" << endl;
#define condition(flag) cout << (flag ? "YES" : "NO") << endl;
#define vec_min(v) min_element(all(v));
#define vec_max(v) max_element(all(v));
#define vec_minmax(v) minmax_element(all(v));
using ll = long long;
using vb = vector<bool>;
using vi = vector<int>;
using vl = vector<ll>;
using vc = vector<char>;
using vs = vector<string>;
const int N = 1e6 + 9;
const ll mod = 1e5 + 7, inf = 1e9;
```

```
template <typename Najmul>
void print(const vector<Najmul> &ans)
{
    for (const auto val : ans)
        cout << val << " ";
    cout << endl;
}
/*
 auto p = vec_minmax(a);
        int mx_a = *p.second;
        int mn_a = *p.first;
*/
void solve()
{
    int n;
    cin >> n;
    vi a(n), b(n), c(n), ans;

    FOR(i, 0, n - 1)
    cin >> a[i];

    FOR(i, 0, n - 1)
    cin >> b[i];
    /*
         ai-aj>=bi-bj
         => ai-bi>= aj-bj
         c[i] = ai-bi
         find maximum in c
         collect all indexs where c[i] == maximum
    */
    // Compute c[i] = a[i] - b[i]
    FOR(i, 0, n - 1)
    c[i] = (a[i] - b[i]);

    // Find the maximum value in c
    int maximum = *vec_max(c);

    // Collect all indices where c[i] == maximum
    FOR(i, 0, n - 1)
    if (maximum == c[i])
        ans.push_back(i + 1);
    // print size
    cout << ans.size() << endl;
    // print ans
    print(ans);
}

int32_t main()
```

```
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    int t;
    cin >> t;
    while (t--)
    {
        solve();
    }
    return 0;
}
```

Complexity Analysis

- **Time**: $O(n)$ per test case (due to linear scans and max_element).
- **Space**: $O(n)$ to store arrays and results.

Example Walkthrough

Input:

a = [3, 1, 2, 4]

b = [4, 3, 2, 1]

Steps:

1. Compute c = a - b → [-1, -2, 0, 3].
2. max_c = 3 (at index 4, 1-based).
3. Only vertex 4 has c_i = max_c ⇒ Strong vertex.

Output:

1

4

Conclusion

The solution efficiently identifies strong vertices by:

1. Simplifying the edge condition to $c_u \geq c_v$.

2. Checking for maxima in c, which guarantee reachability to all vertices.
   This approach ensures optimal performance for large inputs.