Implementation for program 1:

- In program 1, every index of the array, is an element. The value stored in each of those indices, is the index of the parent of that element.
- Initially, every value is it's own index, indicating that each element is its own parent.
- To make a connection, the parent element's index value is stored as the child element's value.

Worst case scenario for program 1:

Naive initial trial:

- To generate a worst-case scenario, my initial thought was that any sample that would generate an output containing only "1"s, would be the slowest possible case, considering every iteration makes a connection.

- However, i then realised that, while generating a "1", meaning that a connection was made, that connection itself could take varying times depending on how many array indices are being modified.

Theoretically correct worst-case scenario:

- I then found from an extensive mathematic proof online that in union find, the worst-case (the slowest process) is when 2 trees of the same size are combined, and the sizes of the trees are a power of 2.

- This can be understood in simple terms like this:
    - take a sample with 8 realms, and 7 requests.
    - from my previous understanding, a sample that contains (0,1), (1,2), (2,3) and so on, would generate an output with only "1"s. However, for each connection, only one array index is being modified, namely the second integer in every line.
    - however, if we instead use the correct theory of worst-case:

     -> connect (0,1), (2,3) and so on to create pairs of 2,
     -> then connect the pairs of 2 using (1,2), (3,4) and so on to create sets of 4,
     -> then connect the sets of 4 to create sets of 8 and so on

    => every connection of pairs of size 2^n, causes 2^n elements to change value in the array.

    - Thus, for every connection in case 1, there is only 1 array element modification in case 1, but 2^n array element modifications in case 2, leading to significantly higher run-time.

Implementation for program 3:

- Each element is initiated with datatype “city” which was defined to have attributes “parent”, “depth” and “value”.
- An array stores pointers to each of these elements. When a connection is to be made, the “parent” attribute of the child element is set to be equal to a pointer to the parent element.
- The “root” is the only element in a tree which has parent = itself.
- To merge 2 trees, the “parent” of the root of the smaller tree is set to be equal to the root of the larger tree.
- A “find” function takes any element as input, and travels up the tree till it encounters the root (element with parent = itself).
- The program involves “path compression”, which basically does the following:

      - When find() travels up the tree, it finally results in setting all elements’ respective parents to be equal to the root of the tree.
      - This results in a compressed tree, and reduces the number of nodes encountered when travelling up from an element till its respective root.

Implementation for program 4:

- This program follows the same algorithm as program 3. The only difference is the implementation; instead of a datatype with attributes as parent, depth and value, this program has arrays parents[], depths[], values[].
- For every element ‘I’, its attributes can be found by <array-name>[I] (for example, parent[I])

The time taken for each of these programs to run, with different numbers of realms and connection requests, is tabulated below:

	10^5 realms 10^5 requests	10^6 realms 10^6 requests	10^3 realms 10^7 requests	100 realms 100 requests
Program #1	1263 ms	127258 ms	136 ms	1 ms
Program #3	58 ms	466 ms	1523 ms	1 ms
Program #4	1308 ms	185966 ms	5338 ms	2 ms