Increasing Sequences

In a vector of *n* integers, find the length of the longest increasing subsequence. A subsequence is a sequence of adjacent elements in the array, and an increasing subsequence is one where the elements are non-decreasing – each element is *greater than or equal to*the elements preceding it.

As an example, consider the following vector with 12 elements:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 8 | 10 | 5 | 2 | 1 | 12 | 12 | 83 | 30 | 40 | 65 |

We can consider this vector as a concatenation of increasing subsequences:

* (5, 8, 10), with length 3
* (5), with length 1
* (2), with length 1
* (1, 12, 12, 83), with length 4
* (30, 40, 65), with length 3

Since the longest of these subsequences has length 4, we want this function to return 4.

**Do not worry about writing an optimal solution to this problem!**One that works is good enough!

Function Requirements

Your 2 function prototypes for this section are:

int increasing\_sequences\_iterative(std::vector<int> &numbers);  
int increasing\_sequences\_recursive(std::vector<int> &numbers);

Similar to the previous problems, you will want to have your recursive function immediately call a helper function that takes an additional parameter that is initially equal to the first element of the array.

int increasing\_sequences\_recursive\_helper(const std::vector<int>& numbers, int startIdx);

(You can name this helper function whatever you want, and its prototype and definition should be in the .cpp file ONLY.) The value of *n* you initially pass to it will be 0 – the index of the first element – and that will change with the following recursive calls.