**Part 1. (100 points)**

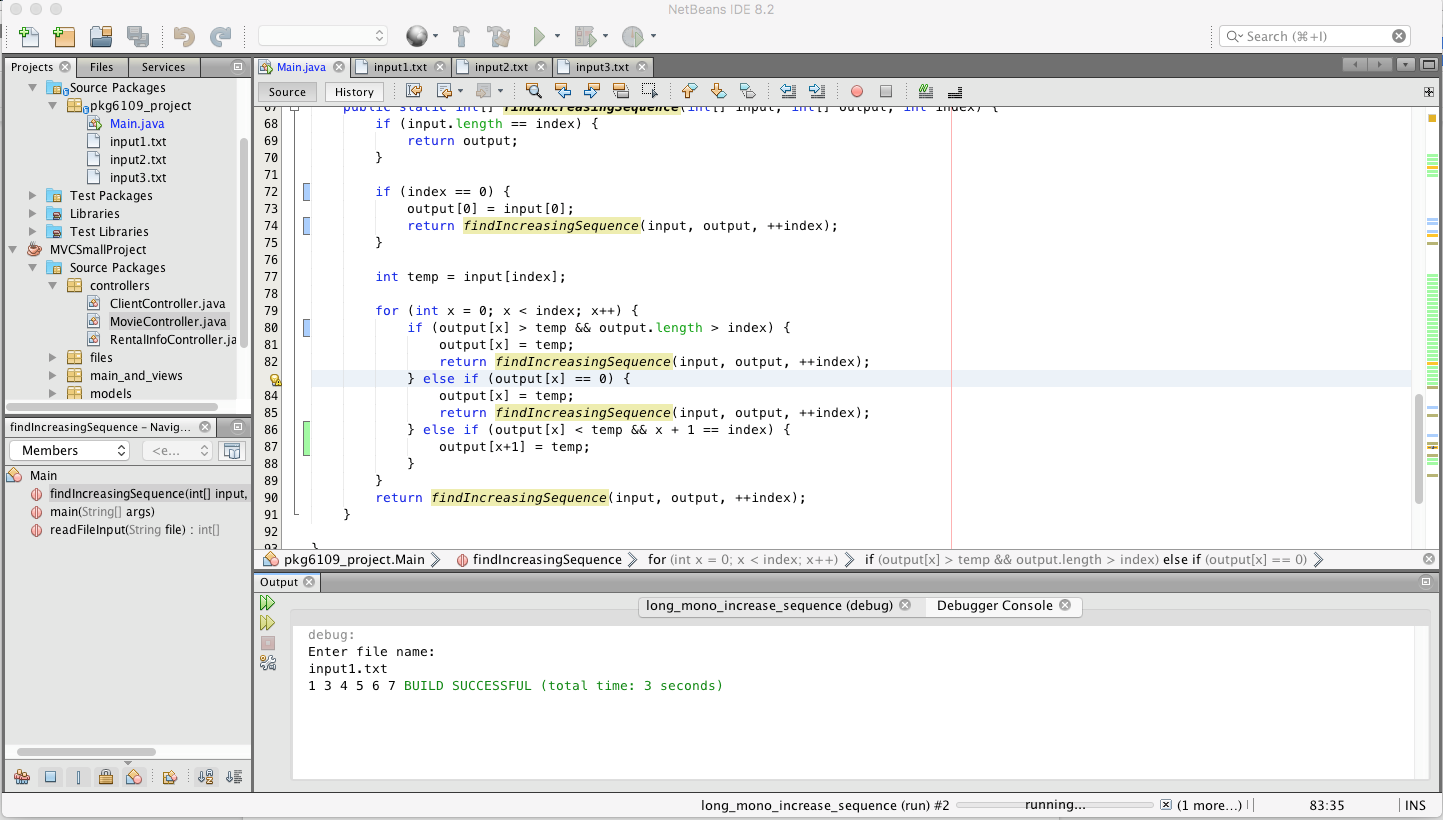
Develop an **O(*n*2)-**time dynamic programming algorithm to compute the Longest Monotonically Increasing Subsequence (LMIS) of a given sequence of ***n*** numbers. For example, given a sequence of **10** numbers <1 20 3 19 4 18 5 17 6 7 >, the LMIS of this sequence is 1 3 4 5 6 7.

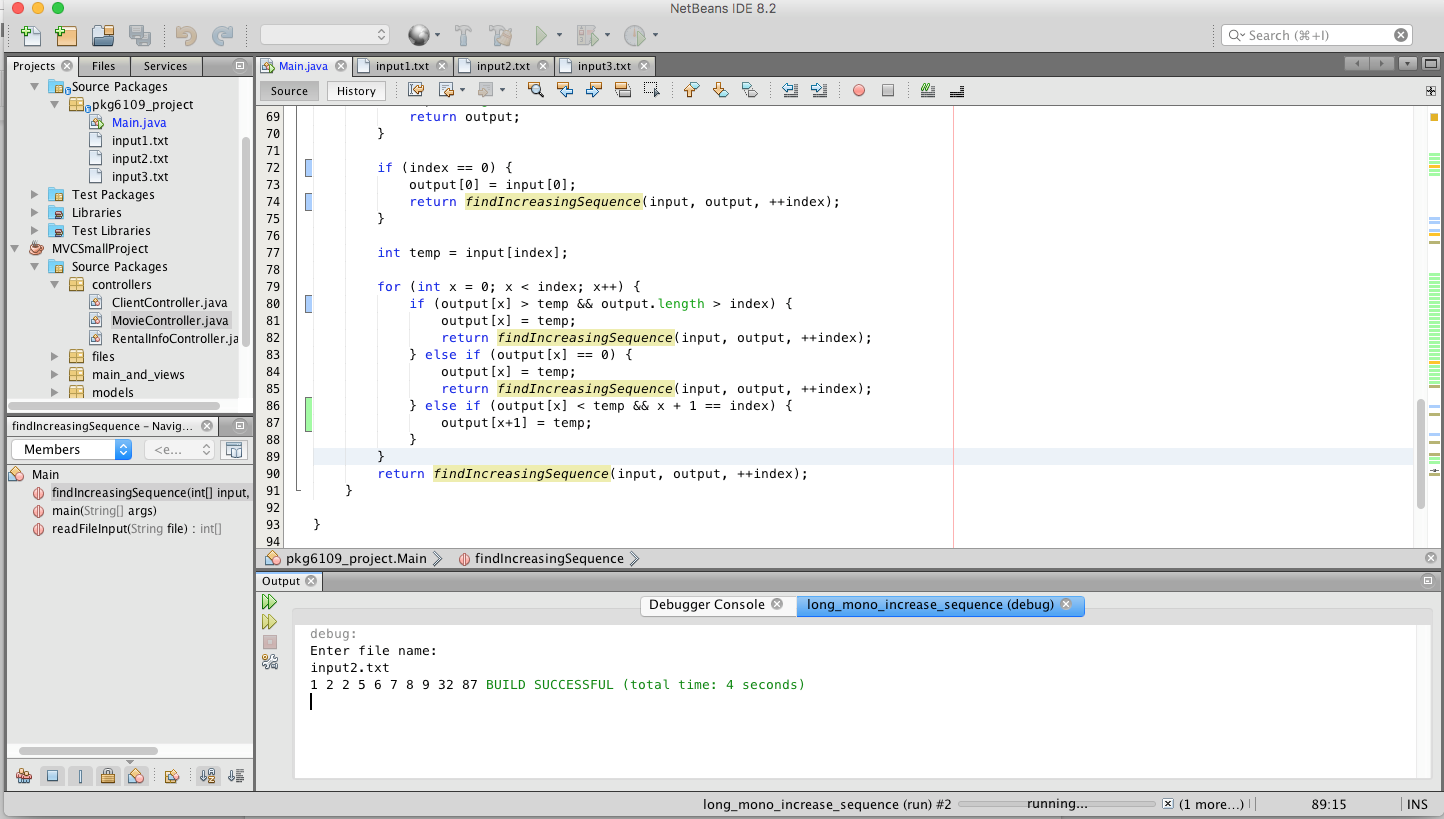
You must complete the following steps and submit your work for each step:

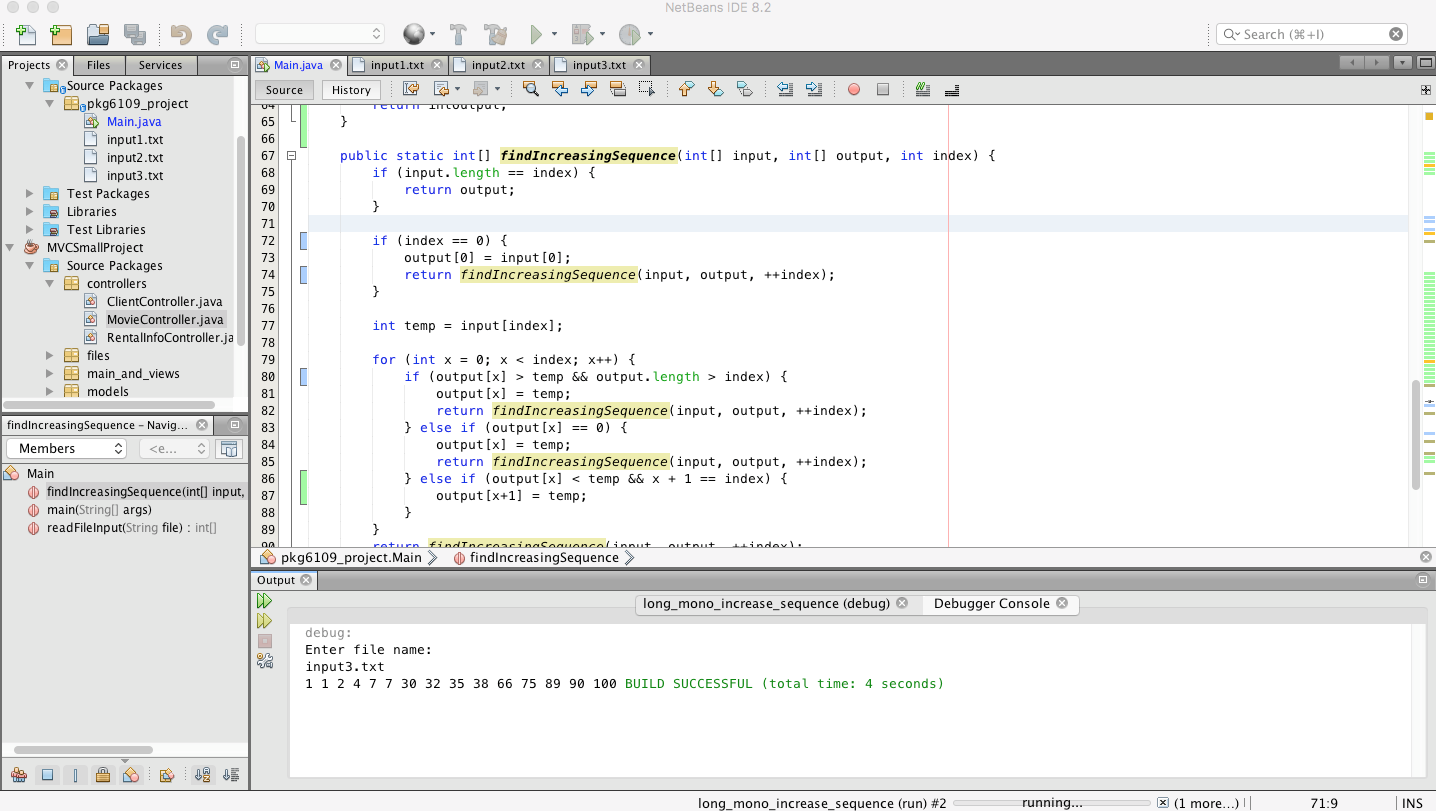
1. Prove that the problem to compute the LMIS exhibits ***optimal substructure.***

* A problem has an optimal substructure if the solution to the problem can be found by using the optimal solutions of its subproblems.
* LMIS utilizes recursion to figure out that for each subproblem whether the new value should be swapped with the tail of the existing element in the output array.
* We can get the a correct increasing integers by comparing at each point the elements and override as needed.
* The final result would be an array with a subset of the main sequence that will have the correct order.

1. Derive a recursive solution to the problem
2. Write the pseudo code to compute the LMIS
   1. Our parameters will be the input array with the main sequence, output array with the subset that will be inputted with new elements and returned at the end of the recursion. The third parameter will be the index.
   2. The base case that will return the output if the index is the same as the input length which means all elements have been exhausted.
   3. For each element in the output array that is before the index of the current index
   4. We will compare to see if the element inside the array is greater than the new value
   5. If it is we swap it with the new element.
   6. The other scenario we will swap with a new element is if the output slot has no element set yet.
   7. Otherwise we put the new element in the tail of the array.
   8. As we do each swap we will increment the index and do a new recursion call.
3. Implement your algorithm to compute the LMIS using Python, or Java (NetBeans IDE 8.0 or above), or C# (Visual Studio 2015)
4. Run your program with the three sample input files: “input1.txt”, “input2.txt”, and “input3.txt” attached on CougarView. If you run your program in a command line, you should be able to type the input file name. If you used a GUI for user interaction, you need to have a File Chooser button to select the input files.
5. Save the screenshots for each input file of the sample run in Word documents’







1. Put everything listed above in a ZIP file including the screenshots for your sample runs.

**Part 2. (20 points extra credits)**

Give an **O(*n lg n*)-**time algorithm to compute the LMIS of a sequence of n numbers. (Hint: Observe that the last element of a candidate subsequence of length ***i*** is at least as large as the last element of a candidate subsequence of length ***i-*1**. Maintain candidate subsequences by linking them through the input sequence.)

You ONLY need to complete the following steps. Put all your work for this part in a Word document.

1. Derive a recursive solution to the problem
2. Write the pseudo code to compute the LMIS
   1. Our parameters will be the input array with the main sequence, output array with the subset that will be inputted with new elements and returned at the end of the recursion. The third parameter will be the index. The difference is instead of keeping one set of arrays for output array we will keep an array within array.
   2. The base case that will return the output if the index is the same as the input length which means all elements have been exhausted.
   3. Instead of looping based on just an index of the output array, we’ll instead loop based on the index of the input array.
   4. For each element in the input array we will use that as the recursion index parameter essentially recursively calling with the next index of the input.
   5. We then do a for loop to check each element of the output inner array to see if the input element is smaller than the what is in the output array.
   6. If it is bigger then we do a while loop to check for each element in the output inner array to see if there are any other elements we can swap in.
   7. If, however, the new input element is bigger than the tail of the output inner array we simply add it to the output array.
3. Prove that your algorithm runs in **O(*n lg n*)-**time
   1. This will be O(n lg n) because the recursion itself will be a loop in the performance time of O(n) because we still have to loop through all the elements in the input array.
   2. However, we will only need to swap if the new input element is bigger and when we do we will only need to loop through the elements that already in the output inner array which will be at a O(lg n) performance level.
   3. Once we accumulate all the subproblems in aggregating the various subsets we will have a subset that has the longest elements based on another for loop.