Net ID: wpicc001
Name: William Patrick Picca

Sources Used:
Communicated with fellow student: Amneh Alsuqi
Sites:
https://www.geeksforgeeks.org/longest-increasing-subsequence-dp-3/

Problem A: Longest Decreasing Subsequence
Submission ID: 134928895

For this programming assignment I start off by taking in the input for the size of a set of integers and then the set of integers stored into an array. I then call the longest decreasing function where I will perform the calculation. Within the function I create a temp to store the LDF value of every number of our inputted array of integers. I then create a nested for loop, with the outer loops iterating up to n times starting at 1, with the inner loop iterating up to the current iteration of the outer loop. Within this nested loop I compare if the current 'i' element is less than the current 'j' element, and if the LDS value of the current 'i' element is less than the LDS value of the current 'j' element. In summation this will calculate the longest decreasing subsequence for every integer value of our array, at which point we will use a for loop to find our highest value of our array of LDS, and return it for our final answer.

Problem B: Weighted Edit Distance
Submission ID: 135018674

For this programming assignment we had to find the edit distance of 2 passes in sets of integers, but instead of counting the changes made, we had to count the differences in integer values. To start off I take in the input for the size of a set of integers and then the set of integers stored into vectors. I then call my edit distance function where I pass in both vectors sizes and then the vectors themselves. Within the edit distance function I set the size of my local 2D vector of integers equal to 1 more than both the size of my vectors. I then initialize the edges of the 2D vector where whenever its 'j' coordinate is 0, I set every 'i' element of this dp vector equal to the previous 'i' element plus the previous passed in aVector element. This will sum up the aVector elements. We then repeat the process with the 'i' coordinate being 0, where I set every 'j' element of the DP vector equal to the previous 'j' element plus the previous bVector element. With this done I have prepared my DP table.

I then take the bottom up approach to calculate the DP table through a nested for loop. Within this for loop we start by comparing if the previous passed in aVector's element is equal to the

previous passed in bVector element. If they are equal then we simply set the current DP cell value equal to the previous ‘i’ and ‘j’ DP cell value. If they are not equal; however, then we must take the minimum of 3 possible operations. The first operation is if we perform an insertion, that is adding a new element to match the value. This would take the form of the previous ‘i’ element of the current DP value summed with the previous aVector element. The second operation, deletion, is deleting a new element to match the value. This would take the form of the previous ‘j’ element of the current DP value summed with the previous bVector element. Our final operation is edit, where the cost is the difference between our previous aVector and bVector elements. This takes the form of the previous ‘i’ and ‘j’ element of our DP, summed with the difference of our previous aVector and bVector elements. When this operation is done it will go back to the top of the nested for loop and repeat this entire operation until the entire table is done. At the end our final dp[i][j] will be our lowest weighted edit distance and it will be returned as our final answer!

Problem C: Longest Unimodal subSequence
Submission ID: 134934327

For this programming assignment I start by taking in the input for the size of a set of integers and then the set of integers stored into an array. I then call the function that will be the main function to calculate our LUS. At the start of the function I create an integer count to store our LUS. I then have a for loop that will iterate through all elements of our array of passed in integers. For any given integer, I will consider that number as the middle point, effectively breaking the array of integers into 2 separate arrays. With all integers that come before that number and all integers with that number and those that come after. I will then call a separate function to calculate the longest increasing subsequence for the array of left numbers. This LIS function works nearly exactly the same as the problem A LDS function, except that when comparing our ‘i’ and ‘j’ numbers within the nested forloop, we check if our ‘i’ number is greater than our ‘j’ number. Following the rest of the function, it will return the highest LIS function value. Then I call the LDS function, passing in the right array of numbers, to obtain it’s LDS value. We then combine the LDS and LIS numbers and compare this sum value with our existing functions LUS value. If it is more than our existing LUS value, we replace it. The for loop then iterates and our middle number of our passed in array has its index incremented by 1. We repeat the entire process and when done, our final LUS value returned is the value of our Longest Unimodal Subsequence!