CS 333 Project 2

Kaan Alp S018037

I used 3 methods and only one class for the structure of the code. One of the methods is the logic behind the dynamic programming implementation. transformArrayInt method is for turning the outlets into an integer array by giving it integers with contrast from the lamps locations.

transformArrayString method is for turning the subsequence with the intended solution(index) back into the 2-digit hexadecimal code for printing to the console.

My dynamic programming logic is based on finding the longest subsequence of increasing numbers. I first transform the outlets array into numbers. These numbers are based on the 2-digit hexadecimal codes position in the lamps array. Then with integers in hand I try to find the longest increasing array. I first created a list of integer arrays which will store the increasing subsequences one by one. Then I populate these arrays. After populating these arrays, I check the size of these lists and try to find the longest one which I will find by storing the longest one into an integer named index. After finding the index I turn it back into the 2-digit hexadecimal code and print the solution onto the console.

List<List<Integer>> subSequences;

for (i= 0; i < n; i++)

subsequnces.add (ArrayList<>); // Populating the lists with respect to n**.**

subSequences.get (0). add (outlets (0)) // Add the 0^{th} element first

for (i = 1; i < n; i++) { // start from the second element

for (j = 0; j < i; j++)

if(subSequnces(j). size > subSequences(i). size && outlets[j] < outlets[i])

// check if the size of new sequence is bigger and check if it is increasing.

subSequences.set (i, subSequences.get(j)); // if yes then create new subsequence by using the old one and add i.

}

subSequences.get(i). add(outlets[i]); // adds itself into the list.

The time for the dynamic programing time complexity for this algorithm is O(n^2)

For the main method the time complexity is O(2n) as to populate both the lamps and the outlets takes O(n) time.

For the transformArrayInt the time complexity is O(n^2) as there is a nested for loop to check both these arrays and populate the integer array properly.

For the checkConnection method the time complexity is O(n^2). First it takes O(n) time to populate the array and then checking and adding the arrays and variables into the subsequences takes O(n^2) time. It takes O(n) time to check and find the index.

So overall it takes 2 * n^2 + 5n time so the time complexity becomes O(n^2).