CSE 321 HW5 171044052

Q1) Firstly, I splited all my elements smaller sub arrays.Then I compare two words until the end of shortest one.If my arrays has common letter, I store this letters in an array.Lastly found common word.Based on merge sort technique.Then I merged all letters and created a string.

Worst case of this code,O(n.log(n)).Splitting array to subarrays take log(n) time, and comparing words letter by letter takes O(n).These are nested so, It takes O(n.log(n)).

Q2)a) In this question, I splited my arrays until get 2 or 1 element left. And look for right side elements of arrays, and left side elements of arrays.I solve like this because I can not sell before buying the product.Then in innter function, I found my bigger element in rightprices subarray.Then extracted all of leftprices subarray elements from it.I found profit and everytime I found a bigger profit, I updated max Income.I tried the return positions of buytime and selltime buy I could not be succesful.My idea was getting buy value and sell value from subararys and search their indexes in my biggest array.But i could not use tüple values.

Worst case of this code,O(n.log(n)).Splitting array to subarrays take log(n) time, and every profit for subarrays take O(n).These are nested so, It takes O(n.log(n)).

b)In this part, I solved it with 2 nested loops.Inner loop,I compared all incomes of arr[i] elements and arr[j] elements.Then I stored their indices for printing.Then I returned a list which is contains all of requested informations.This algorithm takes O(n^2) time complexity for two nested loops.

c)First algorithm is faster than second algorithm in worst case scenario.

Q3) In this question, Firstly I filled lengthOfArray with 1’s until length of input array.Then in nested for loops, I think them as subarrays.If my i is 4, this means that I have a sub array with 4 element.And I looked how many element in this sub array is smaller than my arr[i] element.If this elements are smaller than arr[i] and all elements of arr[j] is smaller arr[j+1], then this is a increasing sub array.I increased my storearray inputs for every increasing sub list, for example, if my i is 4 and subarray is, 1,2,3,4 then I increase my storeSubArrayCounts[i] for every consecutive element.So my storeSubArrayCounts[i] = 4 and that means my 4th sub array has 4 element consecutive.

Worst case is equal to O(n^2) because I have two nested loops.

Q4)a) Firstly I created storeScores array for storing my score result and paths.Then I assign my first score (score[0][0]) in this list.Becaues players always must to start at this point. At each element in the array, the algorithm compares the highest-scoring

path that ends at the element above it with the highest-scoring path that ends at the element to its left. Then it chooses the path with the higher score and adds the current element to that path. The resulting path and its score are stored in the storeScores list.

b)In greedy approach, every move must be played as biggest side.There is no control for biggest path.Look for only next move and next move must be biggest one in choices.So I go for biggest score for every path.

c)For correctness, brute force and dynamic approach can solve this game correct for every situtation because they found every path. But in greedy, it goes biggest score for every move.It does not control every path.So in greedy, there can be false results.

For time complexity.Brute force solve this problem in two nested loops.Which has O(n^2) time complexity.In my dynamic programing approach, I too solve this problem in O(n^2) for worst case.But in greedy, the complexity is equal to O(row+column).And it’s linear.So greedy is faster then this two algorithms.