NEVRA GÜRSES
161044071

CSE 321 HOMEWORK 5

1.QUESTION:

In my algorithm , optNY[i] denotes the minimum cost for month 1, …., i and in month i , person in NY,optSF[i] denotes the minimum cost for month 1, ….., i and in month i, person in SF. Then so, there occur below relation:

optNY[i] = NY[i]+min (optNY[i-1], M+optSF[i-1])

optSF[i] =SF[i]+min (optSF[i-1], M+optNY [i-1])

In my algorithm I use this relation.

At begin,I assign =>**optNY[0]=NY[0]**, and **optSF[0]=SF[0].** Then I start loop for contol all moths.In loop I use above relation. Last elements of optNY and optSF gives min cost of each array.So consequently,for finding optimal cost, last element of which one is smaller ,I return it.

Worst-Case Running Time:

My algorithm has n iterations.In each iteration takes constant time.Thus ,the worst case running time is **O(n).**

2.QUESTION:

In my algorithm,I take start times of sessions and finish times of sessions as array for my selectSession function.In my greedy choice, I always select the first session and I start to select the remain sessions.For remain sessions,I create a loop for sessions.If start time of current session in loop is more than or equal to the finish time of previously selected session,I add it optimalList.And consequently,I return optimalList array that holds indexes of selected sessions.

Worst-Case Running Time:

My algorithm has n iterations.In each iteration there is a if statement that makes constant time operations ,so each iteration also takes constant time.Thus ,the worst case running time is **O(n).**

3.QUESTION:

For this question I write dynamic programming algorithm that finds total number of subsets with the total sum of elements equal to zero.Recurrence relation for dynamic programming algorithm is:

subsetArr[i][j + size] = (zeroSumSubset(i + 1, j + arr[i], arr) + zeroSumSubset(i + 1, j, arr))

If there is a subset with the total sum of elements equal to zero, this function returns integer bigger to 1 other case it returns 1.

Worst-Case Running Time:

Worst case running time of my algorithm is **O(n*j)**, where n is the number of elements in the array and j is the sum of all the elements.

NEVRA GÜRSES
161044071

4.QUESTION:

For solving string alignment problem,firstly I create a 2d array to keep values in it.I initialize this 2d array .After that I create a recurrence relation for solving problem.The crucial step in dynamic programming algorithms,deriving the recurrence relation to solve problem.My recurrence relation is:

alignmentArr[i][j] = max(alignmentArr[i - 1][j - 1] + missMatch ,alignmentArr[i - 1][j]

+ alignmentArr[i][j - 1] + gap)

I create inner 2 loop for control alignments.When (i-1)th element of first string is equal to (i-1)th element of second string I assign to alignmentArr[i][j] = alignmentArr[i - 1][j - 1] +match , if it is not equal,I apply above recurrence relation for alignmentArray.

After that,I create two index i and j and I assign element number of first string for i and element number of second string for j.Total length is adding size of 2 strings.I create 2 array that are size total length of 2 string.And I started loop from total size.I control 4 cases.First is str1[i - 1] == str2[j - 1], second is alignmentArr[i - 1][j - 1] + missMatch == alignmentArr[i][j] ,third is alignmentArr[i - 1][j] + gap == alignmentArr[i][j] and last is alignmentArr[i][j - 1] + gap == alignmentArr[i][j].And so I filled 2 array.This 2 array is alignment of 2 strings.For calculating cost I create calculateCost function and as parameter I give this aligned 2 string and so I calculate max number of cost.

Worst-Case Running Time:

At the beginner part of code there is a inner while loop to fill inner of alignmentArr,that is n*m time.Another parts of code at most n+m time.So in worst case running time is **O(n*m).**

5.QUESTION:

A greedy algorithm is a simple, intuitive algorithm that is used in optimization problems. The algorithm makes the optimal choice at each step as it attempts to find the overall optimal way to solve the entire problem.

My greedy algorithm for solving this problem is finding minimum element in each step and then adding total sum.To do this:

Firstly I find min two number in array and add them and I deleted them.After I create loop and I started to add currently mininum element to total result and also I increase operation number according to total.Then I delete that element in array for another steps.

So,in each step I find minumum element in array ,I add it total sum number and then delete it.So in this way I find the sum of the array with the minimum number of operations. For showing this:

NEVRA GÜRSES
161044071

If my array is =>30,3,10,9,2,8

2+3=5

5+8=13

13+9=22

22+10=32

32+30=62

So minimum operation number is: 5+13+22+32+62=134

Worst-Case Running Time:

There is only a while loop for my greedy algorithm that is length (n-2).Other operations is constant time.So worst case running time of my algorithm is **O(n).**