CS5800: Algorithms Spring 2018
Assignment 3.2

Saptaparna Das

February 11, 2018

(a) Yes this problem can be solved using dynamic programming because it asks us to compute optimal solution out of many solutions. Moreover, in every step we have a choice of picking or leaving that element as a part of sequence, which indicates dynamic programming. It also exhibits optimal substructure property i.e. each local optimal solution is part of global optiomal solution. If we try to solve it by greedy approach, we might not get correct result always, since in greedy style we choose whatever is best at that point of time. For example, if the sequence is like 10,1,6,5 then in greedy style we get subsequence 10,1 but in dynamic programming approach we get the longest sequence 10,6,5.

(b) **Technical problem:** The problem asks us to compute longest decreasing subsequence in an array, where the elements of the output might not be contiguous.

(c) **Pseudo code:**

```
//This method computes the longest losing streak for each participant
// and returns winner
getWinner(arr[0,1,..,m-1][0,1,...n-1])
       //Stores longest losing streak
        max=0

        for(i=0 to m-1)
         result=getLDS(arr[i])
         if(result>max)
             max=result
       // return winner
        return max

//This method computes longest decearesing subsequence
// and prints actual sequence
getLDS(arr[0,1..n-1])
       if(arr.length=0)
         return 0
       dp[arr.length]
      //initialize dp with value 1, because at every index the longest
      //decreaing subsequence is atleast 1
       for(i=0 to n-1)
         dp[i]=1

     //Stores index of longest sequence
      maxIndex=0
     //Stores longest sequence
      result=0
     //Check longest at every index
      for(i=1 to n-1)
         for(j=0 to i)
              if(arr[i]<arr[j]){
                        dp[i] = Max(dp[i],dp[j]+1)
                 if(dp[i]>dp[maxIndex])
                    maxIndex=i
                    result=dp[maxIndex]
      //print actual sequence
       track= arr[maxIndex]
       int next = result-1
```

```
        for(x=maxIndex-1 to 0)
          if(dp[i]==next)
               track=arr[i]+path
               next--
         print track

return dp[maxIndex]
```

The Time complexity here is $O(mn^2)$. Since, it takes $O(n^2)$ time to get longest losing streak in a single array of n elements and there are m such arrays.