CAS CS 330. Problem Set 7

Problem 1

Algorithm: I am going to use Dynamic Programming algorithm to solve this weighted interval problem. Every time, we are going to look at one interval, and we have two choice each time: add this interval to the best solution or not. we can use recursion to go through all the intervals and decide which are in the best solution. In order to speed up the algorithm, we will create a Memory List M that stores all the calculations we have done. We will create a list P[j] that stores the first index of the interval before j interval that does not overlap with jth interval.

```
optimal_solution(j){
    if j = 0 then
        return 0
    else if M[j] is not empty then
        return M[j]
    else
        define M[j] = max(vj+ optimal_solution(p(j)),
optimal_solution(j-1))
```

Analyze: Assume that we already have a list P[j]. Taking value out from list M and stores value into list M only cost theta(1), and we did this j times, therefore, the algorithm for computing the optimal solution is in Theta(n). When we return the solution, we only access the list M once every time, so the running time for finding the solution is in Theta(n). The whole algorithm is in Theta(n).

Problem 2

part a.

This algorithm never use the value h_i, it could be the case that doing h_i is a much better choice than doing h_{i+1} or l_i.

Example: l = [10,1,10,1],

h = [50,20,50,20]

the optimal output would be doing high stress work on week 1 and week 3, and low stress work on week 4 and no work on week 2:

value = 50+0+50+20 = 120

The algorithm will give us this: Doing high stress work on week 2 and week 4, doing nothing on week 1 and week 3

value = 0+20+0+20 = 40

Part b.

```
def Optimal (n):
    if n == 0:
        return 0
    if n == -1:
        return 0
    elif M[n] is not empty:
        return M[n]
    else:
        M[n] = max( h_n + Optimal(n-2) ,  l_n + Optimal(n -1))
        return M[j]


find_solution(n):
    if n= 0:
        do nothing
    else:
        if h_n +M[n-2] >  l_n + M[n - 1]:
            output “high stress work” with find_solution(n-2)
        else:
            output “low stress work” with find_solution(n-1)
```

Analyze:

I am using recursive function to make decision on every week's work type, instead of making two weeks as a small group like the question did. This can avoid the problem like missing evaluating some h_n. Every time, this algorithm only taking out a value from a list or store a value into a list, which is in Theta(1), therefore, the total running time for this algorithm is in Theta(n).

Problem 3

Algorithm：

```
M[ ] is a D list that contains the optimal comball for first n elements
N = total number of elements on a string
input = a list contains the elements in the string

def Optimal (n):
    if n == 0:
        return 0
    else if M[n] is not empty:
        return M[n]
    else:
        list = [0] * N
        for i = 0: n:
            list[i] = quality( input(n) to input(n-i) ) + Optimal( n-i)
        M[n] = the max value in the list
        return M[n]
```

Analyze:

This algorithm is right because it tries out all the possible combinations of words, and return the segmentation that have the highest quality value.

Running Time:

For nth letter in the list, this algorithm needs to check n different combination and return the one with highest quality value. Therefore, for a string with n letters, the running time for the worst case will be (n) *(n-1)*....*2*1, which is n!. So the running time for this algorithm is in Theta(n**2)