<u>CIS 522: Assignment 5 – Dynamic Programming</u>

01951462

Anubhav Shankar

<u>Part B</u>

Q2.) Chapter 6: Exercise 2 (Consulting Jobs)

Ans.) Let L be all the revenue from low-stress jobs and H be all the revenue from high-stress jobs.

a.) According to the question stem, we are given the following pseudo-code of the algorithm to analyze:-

```
For iterations i = 1 to n
  If h_{i+1} > ℓ_i + ℓ_{i+1} then
    Output "Choose no job in week i"
    Output "Choose a high-stress job in week i+1"
    Continue with iteration i+2
  Else
    Output "Choose a low-stress job in week i"
    Continue with iteration i+1
  Endif
End
```

In brief, the above algorithm considers that if a high-stress job is available, it is punted down the line for the successive week and then chooses the next available job to continue. However, if the above condition does not hold, the algorithm determines a low-stress job to begin with and then business as usual depending on the next job.

Let us now take an example to understand visually how the above algorithm operates. We can then see if another optimal solution can be obtained.

Consider,

No. of weeks	Week 1	Week 2	Week 3
L	4	4	4
H	1	8	20

According to the algorithm, it chooses to do nothing in Week 1 and then takes on a high-stress job in the successive week, followed by a business-as-usual approach going forward. So, as per the algorithm, the schedule should be as follows -> 0(Week 1) + 8 (Week 2) + 4 (Week 3) = 12.

This method seems to be a bit shortsighted as there doesn't seem to be an adequate lag period between two jobs which is especially detrimental if the successive jobs happen to be high stress.

Consequently, a more efficient allocation would be to choose a low-stress job in Week 1, no job in Week 2, and then finally a high-stress job in Week 3. So, the consequent schedule will be as follows -> 4 (Week 1) + 0 (Week 2) + 20 (Week 3) = 24.

The optimal algorithm doubles the output in comparison to the original one.

b.) **Pseudocode** ->

optmaxrevenue = max(l_i, h_i) //Denotes the maximum achievable revenue for weeks 1 to i

Int l_i = 0 //Initialize a pointer for low stress jobs
Int h_i = 0 //Initialize a pointer for high stress jobs

If(l_i in week i then select low and high-stress jobs till week i – 1){
//Checks conditions for low-stress jobs
// In such a condition the total revenue will be l_i as follows

$l_i = l_i + \max(l_i(i-1), h_i(i-1))$

}

If(h_i in week i then we cannot select any low-stress jobs til week i-1 but we can choose them for week i – 2){

//Checks conditions for high-stress jobs
// In such a condition the total revenue will be h_i as follows

$h_i = h_i + \max(l_i(i-2), h_i(i-2))$

}

Return optmaxrevenue = max(l_i + optmaxrevenue(i – 1), h_i + optmaxrevenue(i – 2))

c.) The time complexity of the above algorithm will be O(n).

d.) **Program Output** ->

```
<terminated> Job_HW5 [Java Application] C:\Users\anubh\.p2\pool\plug
Week 1 = 0
Week 2 = 50
Week 3 = 0
Week 4 = 0
Week 5 = 40
Maximum achievable revenue: 90
```

Q3.) Chapter 6: Exercise 11 – Carrier Problem

Ans.)

a.) Let, OPT(i) denote the minimum costs spanning weeks 1 to i
   ‘s’ be the supplied items
   ‘r’ be the constant rate
   ‘c_a’ be the costs associated with company A
   ‘c_b’ be the costs associated with company B

b.) **<u>Pseudocode</u>** ->

```
//initializes the cost associated with companies A & B respectively
Int ca[ ]
Int cb[ ]
Int cba[ ] //Cost associated with taking the combination of companies A & B together
Int r // constant rate
Int s //number of items supplied
Int weeks //number of weeks

Mincost = min(ca, cb, cba) //Denotes the minimum costs

For(i = 0; i<= weeks – 1; i++){

        If Company A then
                TA = r * s + OPT(i – 1)
        If Company B then
                TB = 4 cb + OPT(i – 4)

        If combination of companies A & B then
                OPT(i) = min(TA, TB)
```

c.) The time complexity of the above is a constant of OPT(i) ∀ i > 0 and using this we can obtain a schedule by tracing the array of OPT values.

<u>Part A</u>

Q1.) Chapter 6: Solved Exercise 1

Ans.)

a.) This is an optimization problem as we must maximize the revenue obtained by the placement of our billboards subject to the restriction that no similar billboards be placed within five (5) miles of each other.

Let, n -> the next billboard
maxRevenue[ ] -> maximum revenue possible from the allocated sites. This will be an array
restrict -> mile restriction
d -> distance
b[ ] -> an array to check our billboard locations
revenue[ ] -> a predefined array of revenues
placement[ ] -> an array of billboards placed in different miles

b.) **Pseudocode** ->

```
n = 0 //initialize the billboard counter
restrict = 5 //Set the 5-mile restriction
d = 0 //initialize distance

int i = 1 //pointer to keep track of the billboards

If (billboard is present in that mile i.e. n < b.length)
        If(b[n] != i) //check if there is no billboard in that mile
                Then maxRevenue[i] = maxRevenue[i-1]
        Else
                If(i >= restrict)
                        Check maxRevenue[i] = max(revenue[i] + placement[i],
maxRevenue[i-1])
                Else
                        No billboard previously: Place billboard
        n = n + 1

Else
        maxRevenue[i] = maxRevenue[i-1] //Take the maximum revenue from
all the placed billboards

return maxRevenue[d] //maximum revenue for the total highway

revenue[] = [1,3,4,5,5,2,1,5,4,4,3]
```

placement[ ] = [0,0,0,1,0,3,3,7,5,5,6]

Walkthrough:

1 + 0,0 = 1
3 + 0,0 = 3
4 + 0,3 = 4
5 + 1,4 = 6
5 + 0,6 = 11
11 + 3,11 = 22
22 + 7,22 = 44
44 + 5,44 = 88
88 + 5,88 = 166
166 + 6,166 = 322

maxRevenue[] = [1,3,4,6,11,22,44,88,166,322]

c.) The time complexity of the algorithm is O(n)