CSCI 570 -HW4

Isaac Delgado
4189182267

1. Highway (PCH). There are n possible locations along the highway, and the distance from the start to location k is $d_k \geq 0$, where $k=1,2,\ldots,n$. You may assume that $d_i < d_k$ for $i<k$. There are important constraints: each location k you can open only one charging station with the expected profit p_k., You must open at least one charging station along the whole highway. Any two stations should be at least M miles apart.

I. Define subproblems to be solved.

Let OPT[r] be the maximum profit where r is the possible locations on PCH.

II. Write recurrence reaction for subproblems.

If $d_r - d_j \geq M$ where $j = r-1$

$OPT[r] = MAX(OPT[r-1] + p_r, p_r)$

If $d_r - d_j \leq M$ where $j = r-1$

$OPT[r] = MAX(OPT[r-1], p_r)$

Base Case:

OPT[r] = pr if r = 1

OPT[r] = 0 if r = 0

III. Pseudo-code

```
PCH_Gas(int location[n]){
    for(r = 2; r ≤ n; r++){
        for(j = 1; j≤ r - 1; j++){
            If dr - dj ≥ M
                OPT[r] = MAX(OPT[r-1] + pr, pr)
            else if dr - dj ≥ M
                OPT[r] = MAX(OPT[r-1], pr)
            else if r = 1
                OPT[r] = pr
            else if r = 0
                OPT[r] = 0
        }
    }
    return OPT[n]
}
```

IV. Run-Time Complexity

O(n^2)

Isaac Delgado
4189182267

2. Given n balloons, indexed from 0 to n - 1. Each balloon is painted with a number on it represented by array nums. You are asked to burst all the balloons. If the you burst balloon i you will get $nums$[left] · $nums$[i] · $nums$[right] coins. Here left and right are adjacent indices of i. After the burst, the left and right then becomes adjacent. You may assume $nums$[-1] = $nums$[n] = 1 and they are not real therefore you cannot burst them. Design a dynamic programming algorithm to find the maximum coins you can collect by bursting the balloons wisely. Analyze the running time of your algorithm.

I. Define subproblems to be solved

Let OPT[j, k] be the maximum coins collected in the range j to k.

II. Write the recurrence relation for subproblem

OPT[j, k] = MAX(OPT[j, i-1] + OPT[i+1, k] + (num[j-1]*num[i]*num[k+1]), OPT[j, k])

Base Case:

If j = k then OPT[j, k] = num[j]

III. Pseudo-code

```
Ballon(int num[n]){
    num[-1] = num[n + 1] = 1
    for(int k =1; k < n-1; k++){
        for(int j = k; j ≥ 1, j—){
            for(int x = k; x = k; x++){
                If j = k
                    OPT[j, k] = num[j]
                else
                        OPT[j, k] = MAX(OPT[j, i-1] + OPT[i+1, k]
                    + (num[j-1]*num[i]*num[k+1]), OPT[j, k])
            }
        }
    }
    return OPT[j, k]
}
```

IV. Run-Time Complexity

Since we are testing every single element in our num array with every possible first element and last element then our complexity is as follows;

n starting points * n ending points * n last ballon to burst = O(n^3)