Homework Assignment 2 (Theory)

Vidhi Sharma, 2019286
Dolly Sidar, 2019304

1.
Subproblems -
With C[i] as the type of candy i th animal is holding and let
Circus peanuts are type 1, Heath bars are type 2, and Cioccolateria Gardini chocolate trues are type 3. So, Let for all i= 0,1,2,3,....n and j=0,1,2,3
opt(i,j) denotes the maximum score I can earn if I start the swap game at animal i by having j type of candy in my hand. Note that i = 0 denotes that there are no animals and j = 0 effectively means there is no candy.

Recurrence -
1. Base cases: For all i >n, opt[i][j] = 0.

2. For all 1 ≤ i ≤ n, 1 ≤ j ≤ 3, if C[i]=j then

opt(i, j) = opt(i+1, j) + 1

3. For all 1 ≤ i ≤ n, 1 ≤ j ≤ 3, if C[i] != j

opt(i, j) = max{ opt(i+1,j) , opt(i+1, C[i]) - 1 }

Final Solution -
We want to compute opt(1, 1).

Correctness of Recurrence -
We first observe that the optimal solution opt(i, j) can have two cases with respect to the type of candy in my hand i.e. j at the start animal i - either that j candy is equal to C[i] or it is not. We prove the correctness of the recurrence in both these cases.
1. Case I (j candy in my hand is equal to C[i] in opt(i,j)): We claim that then opt(i +1, j) = opt(i, j) -1 is the optimal solution for animals i+1 ,i+2, · · n and candy in my hand at i+1 as j. Suppose this is not true, for the sake of contradiction. Then,

opt(i + 1, j) is a different optimal solution for the sub-problem with animals i+1 ,i+2, · · n and candy in my hand at i+1 as j, such that opt(i + 1, j) > opt(i, j) − 1. But this means that we can create another feasible solution for opt(i, j) in which we are not swapping i and j. But clearly since C[i] = j in this case, by swapping i and j we can increase the score by 1. So clearly the solution in which they are swapped has a higher score and therefore is an optimal solution. This contradicts the optimality of the other feasible solution assumed.

2. Case II (j candy in my hand is not equal to C[i] in opt(i,j)): We claim that then opt(i +1, C[i]) = opt(i, j) +1 is the optimal solution for animals i+1 ,i+2, · · n and candy in my hand at i+1 as C[i]. Suppose this is not true, for the sake of contradiction. Then, opt(i + 1, C[i]) is a different optimal solution for the sub-problem with animals i+1 ,i+2, · · n and candy in my hand at i+1 as C[i], such that opt(i + 1, C[i]) > opt(i, j) + 1. But this means that we can create another feasible solution for opt(i, j) in which we are not swapping i and j. But clearly since the candy in my hand at i+1th animal is C[i] and candy at i th element was j, j and C[i] should obviously be swapped. So clearly the solution in which they are swapped has a higher score and therefore is an optimal solution. This contradicts the optimality of the other feasible solution assumed.

Pseudocode -

```
procedure opt((n, 3))
opt : 2-D Array of size (n + 2) × (3 + 1)
for j=1 to 3 do
        opt[n+1][j] = 0;
end for
for i=1 to n do
        for j=1 to 3 do
                if (C[i] == j) then
                        opt[i][j] = opt[i+1][j] + 1
                else
                        opt[i][j] = max{ opt[i+1][j] , opt[i+1][C[i]]- 1 }
                end if
        end for
end for
Return opt[1][1]
End procedure
```

Runtime -
The running time is clearly dominated by the nested for-loops. Hence it in O(n*3).

2.
Subproblems -
i = the number of bakeries set up
j = the j th house.
Let for all i= 0,1,2,3,....k and j=0,1,2….n opt(i,j) denotes the minimum sum of distances between 0 to j houses and their nearest bakeries with i bakeries set up between houses 0 to j(inclusive). Note that i = 0 denotes that no bakeries are set up and j = 0 effectively means there are no houses.

Recurrence -
Let's consider
mid[i][j] = the sum of distances between i to j houses(inclusive) and the bakery set up in the middle house. (This is because if the middle house is set up as a bakery then the sum of distances from it will be minimum).

1. Base cases: For all i = 1, and 1 ≤ j ≤ n
   opt[1][j] = mid[1][j]

2. For all 2 ≤ i ≤ k, 1≤ j ≤ n,
   1≤ u ≤ j
   opt[i][j]= min (opt[i][j], opt[i-1][u-1] + mid[u][j])

Final Solution -
We want to compute opt(k, n).

Correctness of Recurrence-
We claim that For all 2 ≤ i ≤ k, 1≤ j ≤ n, 1≤ u ≤ j
opt[i][j]= min (opt[i][j], opt[i-1][u-1] + mid[u][j])
is the optimal solution for i post office set up between 0 to jth house. Suppose this is not true, for the sake of contradiction. Then, opt(i, j) has a different optimal solution for the sub-problem. But this means that we can create another feasible solution for opt(i, j) in which the mid[u][j] is different for this solution. Opt[i-1][u-1] has already set up i -1 bakeries till u-1 house. The i th bakery is to be set up

somewhere between u to j house, we chose this house to be the middle house. But if mid[u][j] is different, then we can't choose the middle house between u to j houses as a bakery for a minimal solution. But the sum of differences between the x coordinates of each house (u to j) and a selected coordinate (between u to j) can be the minimum only if the selected coordinate lies in the middle. This means mid[u][j] is minimum and therefore opt[i][j] is the optimal solution. This contradicts the optimality of the other feasible solution assumed.

Pseudocode-

```
for i=0 to k do
        for j=0 to n do
            opt[i][j] = inf
        end for
end for
for i=1 to n do
         opt[1][j] = sum[1][j]
end for
for i=2 to k do
        for j=1 to n do
                for u=1 to j do
                         opt[i][j]= min ( opt[i][j], opt[i-1][u-1] + mid[u][j])
                end for
        end for
end for
Return opt[k][n]
```

Runtime :

The running time is clearly dominated by the nested for-loops. Hence it in O(n^2*k).

3.

Subproblems-

Let , for i ,j = 1..n we have opt(i,j) is the optimal sequence for which the score is maximum.

Recurrence-

1. Base case : for all i, j = 1….n , for i== j
   return array A[i]

2. For all i ≤ k <j, if ((opt (i, k))^2 + (opt(k+1,j)^2) > score)
   opt(i,j) = opt (i, k) + opt(k+1,j)

Final Solution -

We want to compute opt(n, n).

Correctness of Recurrence-

We claim that for all i ≤ k <j

if ((opt (i, k))^2 + (opt(k+1,j))^2 > score)

then , opt(i,j) = opt (i, k) + opt(k+1,j)

is the optimal sequence to maximize the energy gained in every single operation for combining two drops and getting a new drop from two original drops. Suppose this is not true, for the sake of contradiction. Then, opt(i, j) has a different optimal sequence for the sub-problem, such that there exists a more efficient sequence of operation to achieve the maximum amount of energy for time traveling that is P*. if P be the cost for the above contradiction statement then P* must be greater than P, but after performing all the 2(n-1)C(n-1)/n sequence for any n drops it is found that P* can’t be greater than P and which means this contradicts the optimality of the other feasible solution assumed.

Pseudocode-

```
initalized the 2D array opt with -1
int ans = 0;
travellling( A[1...n] , i , j, score)
{
        if (i == j)
                return A[i] ;
        if(opt[i][j] != -1)
                return opt[i][j] ;
        for(int k =i ; k<j; k++)
```

```
                {
                        int c = travelling(A , i ,k,score);
                        int d = travelling(A , k+1 ,j, score)
                       if( score+ c^2 + d^2 > ans)
                        {
                          opt[i][j] = c+d
                          score = score+ c^2 + d^2
                          ans = score
                        }
                }
                return opt[i][j] ;
        }

travelling( A, 1, n, 0);
print ans;
```

Runtime :

The running time is clearly dominated by the nested for-loops. Hence it in O(n*3).