Programming Assignment 3 Pseudo Code
Submitted by: Tushar Bansal (tb34826)

① find Optimal Response Time (

n ← no. of Houses
k ← no. of Stations
houses Positions ← get House Positions

These are lengths initiated

r = new int[n][k]

Base Case 1:
We have one house & any number of police stations

for i in (0, k-1) {
    r[0][i] = 0 ← assigning all police stations to first house
}

Base Case 2:
We have one police station & any number of houses

for i in (0, n-1) {
    r[i][0] = house Position of (i^th - 1^st) house / 2
}

Putting the police station in exact middle

Recurrence code:

for i in (1, n-1){
  for j in (1, k-1) {

    r[i][j] ← ∞

```
minresponseTime ← ∞

for x in (0, i-1) {

    response ← max ( r[x][j-1], (house at(i) - house at (x+1)]/2)

        if response < minResponse Time {

            minResponse Time ← response

        }

    }
r[i][j] = minResponse Time

    }

}

town. SetResponseTime ( r[n-1][k-1])
 return (town)
)
```

(2) find Optimal Police Station Positions (

n ← no. of Houses
k ← no. of Stations
housePositions ← get House Positions

These are lengths initiated

r = new int[n][k]
c = new int[n][k] (last station at i, j)
l = new int[n][k] (To keep track of prev. station)

acts as a pointer for previous locations

BC1: We have one house & any number of police stations

for i in (0, k-1) {
  r[0][i] = 0 ← assigning all police stations to first house
  c[0][i] = houses at (0) ← all stations at same index as first house
  l[0][i] = 0 } ← no previous location

BC2: We have one police station & any number of houses

for i in (0, n-1) {
  r[i][0] = housePosition of (i^th − 1^st) house / 2 ← station is put in middle of extremes
  c[i][0] = housePosition of (i^th + 1^st) house / 2
  l[i][0] = 0 ← no previous location

Recurrence code:

for i in (1, n-1){
  for j in (1, k-1) {

    r[i][j] ← ∞
    minResponseTime ← ∞

```
for x in (0, i-1) {
    response ← max ( r[x][j-1], [house at (i) − house at (x+1)]/2)
    if response < minResponseTime {
        minResponseTime ← response
        policeStationPlacement ← house at (i)th + (x+1)th house / 2
        prevLocation ← x
    }
    r[i][j] = minResponseTime
    c[i][j] = policeStationPlacement
    l[i][j] = prevLocation
  }
}
optimalPoliceStationPosition ← new ArrayList
currentHouse ← n-1
currentStation ← k-1
while (currentStation >= 0) {
    optimalPoliceStationPosition.add(
        c[currentHouse][currentStation])
    currentHouse = l[currentHouse][currentStation]
    currentStation -= 1
}
```

station placed in middle location before this station is X

The pointer helps know what previous house (x) is not covered by a police station

reverse optimal Police Station Position
town = set Police Station Positions
(optimal Police Station Position)

return (town)
)

(3) Both cases involve bottom-up approach where all the elements of a 2D array are filled by using previously available information.

In both of them, there are three for loops nested in one another with total complexity of $O(n^2K)$. Since within the for loops all lines are used to only use an if-else loop or assign or compare, they are all $O(1)$ operations.

Reason it is $O(n^2k)$ is first loop runs through n iteration * k (second loop) * $i-1$ of third, but since max value of i is n, it goes to do at max operation in range of n, which suggests $O(n * K * n) = O(n^2K)$

The base cases have complexity of $O(K)$ & $O(n)$ respectively which is less than that of the recurrence.

For the second code, there is also a while loop which runs at $O(k)$ but since all operations are again adding or assigning, there is $O(1)$ complexity inside. Reversing the array is also $O(k)$ but it happens outside the while loop

But overall, the complexity for both codes is $O(n^2k)$

where $n \longrightarrow$ no. of houses and $k \longrightarrow$ no. of stations