CSE 321
HW5

1) Closest Drones

- This function sorts the drones points based on their x coordinates. I use selectionSort for sorting. This part has $O(n^2)$ time complexity.

closestDronesUtil

- This function is a recursive function divides the set of drones into two halves and recursively finds the closest pair in each half. Here is the recurrence relation:

$$T(n) = 2T(n/2) + f_{strip}(n).$$

stripClosest

- This function iterates through the drones in the strip and finds the minimum distance between drones in the strip. The nested loop takes $O(n^2)$ time complexity.

Overall

$$T(n) = 2T(n/2) + O(n^2)$$

$$a=2 \quad b=2 \quad d=2$$

$$b^d > a \rightarrow 2^2 > 2$$

$$T(n) \in \underline{O(n^2)}$$

2) min Sensors

- This function sorts the sensors points based on their x coordinates. I use selectionSort for sorting This part has $O(n^2)$ time complexity.

minSensorsUtil

- This function is a recursive function that divides the set of sensors into two halves and recursively finds the minimum number of sensors needed to cover each half.

Here is the recurrence relation:

$T(n) = 2T(n/2) + O(n)$

Master Theorem $\rightarrow$ $O(n \log n)$

Overall

This code has $O(n^2)$ time compexity.

3) minCostDNA

n → length of the first sequence
m → length of the second sequence

- This function initializes cost matrix 'dp' at first and takes $O(n*m)$. Then filling first row and column takes $O(n+m)$.
- The nested loop runs for '$(m+1)*(n+1)$' iterations and for each iteration a constant amount of work is done. This parts time complexity is $O(n*m)$.
- And trace back loop runs in $O(n+m)$.

Overall

This code has $O(n*m)$ time compexity.

4) dpMaxDiscount

- This dynamic programming function iterates over each store and for each store it creates a copy of the current subset and appends the current store. 'calc' function is called to calculate the discount for the current subset.
- In question 'calc' function mentioned to be $O(1)$.
- The loop has $O(n)$ time complexity.

Overall

This code has $O(n)$ time compexity.

5)
```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

```
Class Antenna:
    def __init__(self, leftPoint, rightPoint):
        self.leftPoint = leftPoint
        self.rightPoint = rightPoint
```

- I create two class for this program

maxAntennas

- Sorting Part

- In this part I use selection sort for sorting antennas based on their X coordinates. This part has $O(n^2)$ time complexity.

- Iterating over Antennas

- After sorting code iterates through the sorted antennas to find the set of antennas that can be activated without interfering with each other. For each antenna, it checks whether it can be activated based on the condition. This operation has $O(n)$ time complexity.

Overall

This code has $O(n^2)$ time compexity.

Note!! I use selection sort in every question for simplicity. Other sorting methods can have more optimal time complexities.