CSE 321 Homework 5

1) drone_distance (drone1, drone2) → It calculates the distance between two drones using Euclidean distance formula → $O(1)$

minimum_distance_brute_force (drones) → It exhaustively iterates all the drone pairs and returns the distance with the minimum distance. It uses two for loop → $O(n^2)$

closest_in_strip (strip, min_dist) → This function exhaustively calculates the minimum distance of all the drone pairs by considering there are maximum 6 drones to compare. Finally compares the minimum with the input minimum and returns the minimum. → $O(n^2)$

closest_pair_recursive (sorted_drones_x, sorted_drones_y) → This function divides the drones to half according to the middle drone in terms of x. Creates a strip on that middle drone. Finds the minimum distance for each half and gets the minimum of that two distances. According to that minimum restricts the strip by x-coordinate + minimum and x-coordinate - minimum in terms of x. And finds the drones in that area and stores them in the strip list. calls the closest_in_strip function and calculates the minimum again and returns it. → $O(\log n) + O(n^2)$

closest_drone_pair (drones) → prepares the recursive function and calls it. Sorts the drones according to x and y. → $O(n\log n)$ → sorting

$+ \rightarrow O(\log n) \cdot 2$

$+ \rightarrow O(n^2)$

$T(n) \in O(n\log n)$ because sorting

---

2) cross (o, a, b) → a, b are sensors and o is the origin. this function finds the cross product of oa and ob. This function helps to determine the side of the point like left or right side → $O(1)$

build_hull (sensors, leftmost, rightmost):
This recursive function constructs the convex hull into segments.
It identifies the farthest point from the line combined from leftmost and rightmost.
Then it divides the remaining points into two groups.
The function recursively divides the groups and finds the farthest points.
It ends up building the best perimeter. → $O(n\log n)$

find_best_perimeter (sensors) → Prepares the recursive function build_hull and combines the drones. → $O(n\log n)$

3) align-dna-sequences (seq1, seq2):

This function solves the problem by creating a dynamic table named matrix. It creates table matrix which is 2D $len(seq1)+1 \times len(seq2)+1$. It fills the table's first row and column like this:

0	1	2	3	...	n
1	0	0	0	-	0
2	0	0	0		0
3	0	0	0		0
4	0	0	0		0
.					
n	0	0	0		0

as base case.

Then iterates every row and colum to build the table. if two sequence charaters are not same. It checks which option is optimal. For example i,j

table[i-1][j] + 1 → for addition

table[i][j-1] + 1 → for remove

table[i-1][j-1] + 3 → for substition.

and fills the table accordingly. returns the result which is the last element of the table.

Time complexity: $O(n \cdot m)$

beause it iterates every element of the table which size is $m \cdot n$

---

4) The algorithm generates all the sequence calculations and fills the dynamic table accordingly. Generates all sequences in 2^n time and fills the table in 2^n time. After all finds the max discount in the dynamic table and returns it in 2^n time.

$T(n) \in O(2^n \cdot 3) = O(2^n)$. As a comment this problem is not suitable for dynamic programming.

---

5) max-antenna (antennas):

This algorith takes a list of all antennas along the street. And sorts them according to their ending-place. Every antenna has a starting-place and ending-place.

The greedy algorithm iterates all that sorted antennas and check if the starting-place of the current antena is bigger than the previous antenna then add that antenna to the solution.

in the end of the searching return the solution list.

Time complexity: initial sorting → $O(n\log n)$

searching → $O(n)$

$T(n) \in O(n\log n)$