190104017 Yavuz Selim Kızılcık

1-) The algorithm use divide and conquere strategy to find closest pairs.

closest_pair function sorts the points based on x_coordinates. calls the recursive function closest_pair_recursive.

Closest pair recursion function finds closest pair of points in given set of points it employs divide and conquer strategy.

If the number of points are small it uses brute force function. to find closest pair. otherwise it divides the points into left and right halves and finds the closest in each pairs. It strips the points within a certain distance from the middle is identified using the points_within_strip function It checks the closest pairs within the strip using the closest_in_strip function.

The algorithm divides the points into smaller sets, and solves the problem recursively. And combines the results to find the overall minimum distance, the closest_pair_recursive divides the pars into two halves takes log time.

Sorting according to x values takes O(NLogN) time. N; number of points.

closest in strip use nested loop but only for limited number of points takes O(N)time. Algorithm domanated by sorting step takes O(NlogN) time.

2-) The provide code implements an algorithm to find the optimal set of sensors for securing a perimetre. The sensors are represented as 2D plane, and the goal is to activate a subset of sensors to form a secure perimeter. The algorithm uses divide and conquere approach to solve the problem efficiently.

Secure Parametre fonction:
Sorts the censors based on x coordinates.

divide and Conquer function

recursive divide and conquere function that solves the problem for a given set of sorted sensors,

If there are fewer than three sensors. It returns the entire set of sensors as the base case.

Otherwise it divides the set into left and right halves. and recursively finds the optimal solutions. for each halves

The solutions combines using combine-solution function.

combine_solutions-function:

merges and sorts the left and right halves based on y-coordinates. It initialized the final solution with the first sensor and iterates through the remaining sensors. adding only those that contribute to the secure permetre.

Sensors are added to the final solution if their y-coordinates is greater than the y coordinate of the last sensor in the final solution. ensuring non decreasing order.

time complexity of the algorithm dominated by sorting operation. which secure-permetre combine-solution takes O(NlogN) time.

3:) the algorithm use dynamic programing for aligning two DNA sequence. The goal is to find minimum cost alignment by considering insertion, deletion, and substitution operations.

Initialization: the lengths of the two DNA sequence are stored in variable. m and n

A 2D matrix dp-matrix is initialized to store the cost of alignment.

Matrix Filling:

Matrix filling using a nested loop that iterates over the lengths of both sequences.

The cost of alignment is calculated based on insertion, deletion. and substitution operations.

The minimum cost are choosen among the operations that stored in the matrix.

Trace Back:
After filling the matrix, the algorithm trace back to find the sequence of operations with the minimum cost.
starting from the bottom-right corner of the matrix, the algorithm moves diagnally. Based on the values in the matrix.
Reverse Op:
The list of operations is reversed to obtain the correct order of operations.

Time complexity:
The nested loop has time complexity:
O(n.m) m and n are sequence length.
the track back O(n+m)
Overall time complexity O(m.n).

5-) Maximize the activation of antennas the goal is to select a subset of antennas that do not intersect in coverage and have the maximum possible activation, The algorithm sorts the antenas based on their coverage. and iterates through them, activating non intersecting antennas.
Sorting:
Antennas are sorted on their coverage endpoint in ascending order.
Activate Antennas Set:
An empty set ('activated_antennas') is initialized to keep track of the activated antennas.
Iteration through Sorted Antennas:
the algorithm iterates through the sorted antennas.
For each antenna, it checks whether it intersects with any activated antenna by comparing its starting position with the endpoint of already activated antennas,
If the antenna does not intersect with any activated antenna, it is added to the set of activated antennas,
Time Complexity:
Sorting time complexity takes O(NlogN). N number of antennas
through sorted antennas takes O(N) time.

Overall time Complexity takes O(NlogN) time