CSE 321 Homework 5

① We can use divide-and-conquer approach for finding closest distance of two drones. The main idea is to divide the set of drones into two halves, recursively find the minimum distance in each half, and then merge the results by considering the minimum distance accross the two halves and within a "strip" around the middle line.

- if the number of drones are 3 or less, uses brute-force
- sort the drones based on their x-coordinates and split them two halves
- recursively find the minimum distance in each half
- find the minimum distance between points on the left and right halves.
- create strip
- sort the strip based on y-coordinates and check for the minimum distance within in strip.
- return the global minimum distance considering the result from the halves and the strip.

Time Complexity Analysis: Sorting step $O(n \log n)$, Recursion $O(n)$, Strip Check $O(k \log k)$ where k is the number of points in the strip. Dominat term is $O(n \log n)$ so the time complexity of the algorithm is $\boxed{O(n \log n)}$

---

② We can use divide-and-conquer approach for idea of recursively partitioning the sensor locations and making decisions at each step to minimize the number of sensors needed to secure the perimeter. The algorithm can be designed to efficiently determine the minimum number of sensors required while ensuring coverage of critical exploration areas.

- sort the sensors based on their x-coordinates
- divide the sensors into two halves and apply recursively.
- merge the right and left
- when the set of sensors contains only one or two sensors, calculate and return the minimum number of sensors needed to cover the critical exploration areas.
- for each merge step, ensure that the critical exploration areas are covered by selecting sensors strategically.

Time Complexity Analysis: Sorting sensors takes $O(n \log n)$, dBC step $O(n \log n)$. Therefore, the overall time complexity of the proposed algorithm is $\boxed{O(n \log n)}$.

---

③ We use dynamic programming algorithm to store the operations at each step. This can be achieved by keeping track of the previous operation that led to the current state. The operations include insertion, deletion and substitution.

Time Complexity Analysis: The time complexity of the dynamic programming part is $O(m*n)$ where m and n are the lengths of the two DNA sequences. The backtracking step adds a linear factor, so the overall time complexity is still $O(m*n)$. Therefore, overall time complexity is $\boxed{O(m*n)}$

(4) By using dynamic programming, can calculate the maximum achievable discount for a given sequence of stores visited, Main idea is to build a dynamic programming table where each entry represents the maximum achievable discount for a subsequence of the stores. The recurrence relation will be based on whether we include or exclude the current store in our subsequence.

Time Complexity Analysis: The time complexity of the algorithm is $O(n)$ where n is the number of stores. Algorithm iterates through the stores once, and at each step, it performs constant time operations.

(5) Main logic is to always choose the antenna that covers the rightmost point and does not intersect with the currently activated antennas. This ensures that we cover the maximum area while avoiding interference -

- Sort the antennas based on their rightmost coverage point.
- Start with the first antenna in the sorted list and activate it. This antenna covers the rightmost point.
- Move to the next antenna. If it doesn't intersect with the currently activated antennas, activate it. Otherwise skip the next antenna.
- Continue this process until all antennas are considered.

Time Complexity Analysis: Sorting the antennas initially takes $O(n \log n)$ time. Greedy algorithm iterates through the sorted list once, checking for intersections and activating antennas. The overall time complexity is dominated by sorting step, resulting in $O(n \log n)$.