Instructions.

1. Solution may not be submitted by students in pairs.

2. You may submit a pdf of the homework, either printed or hand-written and scanned, as long as it is easily readable.

3. If your solution is illegible not clearly written, it might not be graded.

4. Unless otherwise stated, you should prove the correctness of your answer. A correct answer without justification may be worth less.

5. If you have discussed any problems with other students, mention their names clearly on the homework. These discussions are not forbidden and are actually encouraged. However, you must write your whole solution yourself.

6. In this homework, please solve the questions yourself (without consulting other students). In the previous homeworks, I was trying to encourage brainstorming and to promote fair team work. I hope that you have found it useful. But of course then, under this model of collaboration, your homework grade reflects what you know, but is also influence by what your team-mates know.

   So for hw5, do not discuss possible solutions with other students, (nor publicly on Piazza.)

7. Unless otherwise specified, all questions have the same weight.

8. You may refer to data structures or their properties studied in class without having to repeat details, and may reference theorems we have studied without proof. If your answer requires only modifications to one of the algorithms, it is enough to mention the required modifications, and the effect (if any) on the running time and on other operations that the algorithm performs.

9. In general, a complete solution should contain the following parts:

   (a) A high level description of the data structures (if needed). E.g. We use a binary balanced search tree. Each node contains, a key and pointers to its children. We augment the tree so each node also contains a field...

   (b) A clear description of the main ideas of the algorithm. You may include pseudocode in your solution, but this may not be necessary if your description is clear.

   (c) Proof of correctness (e.g. show that your algorithm always terminates with the desired output).

   (d) A claim about the running time, and a proof showing this claim.
1. Let $P = \{\alpha_1 \ldots \alpha_n\}$ be the $x$ coordinates of a set of points, all on the horizontal line $y = 0$. Let $Q = \{\beta_1 \ldots \beta_n\}$ be the $x$ coordinates of a set of points, all on the horizontal line $y = 1$.

![Diagram](image)

Figure 1: An example of two sets of points. Each is labeled with a character from \{A,C,T,G\}.

Each point is also associated with a **label**. This label is one of characters \{A,C,T,G\}. Suggest an algorithm, as efficient as possible that finds a set $S$ of segments, satisfying all the following conditions:

(a) Each segment of $S$ connects a point of $P$ to a point on $Q$. A point might me connected only if it matches the same label (for example, 'A' to 'A').

(b) No two segments of $S$ cross each other

(c) The length of each segment is at most some given threshold $L$ (e.g. 17)

(d) The **cardinality** of $S$ is as large as possible.

2. Write the pseudo-code of an algorithm that receives as an input a convex polygon $P$ whose vertices are $\{p_1 \ldots p_n\}$. The output is the list of chords that forms a triangulations of $P$, such that the sum of their lengths is as large as possible. The running time is $O(n^3)$.

Don’t forget the initialization.

3. Let $P$ be a convex polygon whose vertices are $p_1 \ldots p_n$. Assume $n$ is an even number. We mark the vertices $\{p_1, p_2 \ldots p_{n/2}\}$ as **blue** vertices, and the others are **red** vertices. See Figure 2 for an example.

Suggest an algorithm for finding a triangulation of $T$ such that the sum square of areas all triangles is as small as possible, and every triangle contains at least one red vertex and at least one blue vertex.

Solve this question in $O(n^2)$ time. A partial credit would be given for an $O(n^3)$ time algorithm.

4. Suggest an $O(n^2 \log n)$-time algorithm that given two polygonal paths $P = \{p_1 \ldots p_n\}$ and $Q = \{q_1 \ldots q_n\}$, the algorithm finds the smallest value $r^*$ such that $Fr(P, Q, r^*) = \text{YES}$, but $Fr(P, Q, r') = \text{NO}$ for any $r' < r^*$.

Hint start by obtaining an $O(n^4)$-time algorithm. Note that there is some $p_i \in P$ and $q_j \in Q$ such that $r^* = \|p_i - q_j\|$.

5. The question deals with the problem called Weak Frechet Distance. We are given two polygonal paths $P = \{p_1 \ldots p_n\}$ and $Q = \{q_1 \ldots q_n\}$, and a maximum length $L$ of any
possible leash. The person and the dog starts at location $p_1$ and $q_1$ respectively, and they end when they reach locations $p_n, q_n$ respectively. Assume that at a certain time, they are at locations $p_i, q_j$. At each time-stamp,

(a) the person could move to $p_{i-1}$, could stay at $p_i$ or could move to $p_{i+1}$. And at the same time stamp

(b) the dog could move to $q_{j-1}$, could stay at $q_j$ or could move to $q_{j+1}$.

Suggest an algorithm that determines in $O(n^2)$ time whether there is a sequence of jumps for which the distance person-dog (that is, the length of the leash connecting the dog to the person) is $\leq L$. If it exists, explain how to find this sequence within the same time bounds.

For clarification: We measure the length of the leash between jumps, when the person-dog are at rest and are preparing for the next jump.

6. Similar to the previous question, but this time try to minimize the sum of lengths of leashes used in all time stamps. The running time is now $O(n^2 \log n)$. Show all details of your solution.

Hint: Though it is a classical dynamic programming problem you might find it more convenient as problem of finding a shortest path in graph, where weights are assigned to vertices.