CS302 - Problem Set 2

1. Pick (at least) one of the following topics and read the articles relating to the topic. Then answer the questions below.

   - Improved air traffic control:
     - Impact of Improvements of Air Traffice Control
     - Air Travel and Global Warming
   - Improved Robot Control:
     - Impact of Growing Robotics in China
     - Infographic on Manufacturing Employment/Robot Use
     - Study on Modern Workforce

   Did the readings change your thoughts about the ethical implications of improved air traffic control or improved robot control that we started thinking about in class? What stood out to you in the articles about the ethical implication of this technology? Are there additional aspects to these issues that the articles did not consider or glossed over, perhaps because of bias in the authors' perspectives? How comfortable would you be (from an ethical perspective) implementing the relevant algorithm?

2. For the closest points problem, we argued we only needed to look at points with x-coordinates that were within $\pm\delta$ of the midline, where δ is the smallest separation found in either the left half or the right half of points. In this problem, we consider how that analysis would change if we were to calculate the distance between points differently. Decide what range of x-coordinates away from the midline you should consider if we instead used

   (a) The Minkowski distance: $d(p_i, p_j) = (|x_i - x_j|^q + |y_i - y_j|^q)^{1/q}$, for $q \in \mathbb{R}^+$. This distance is what you get on curved spacetime like distances close to a black hole. (Technically all spacetime is curved, but we really only notice it when you get close to a massive object.)

   (b) The skewed distance: $d(p_i, p_j) = \sqrt{2(x_i - x_j)^2 + (y_i - y_j)^2}$. This distance would make sense in a scenario where it is much harder to travel in the x-direction than in the y-direction. For example, suppose to travel in the y-direction, you can get on highways, but in the x-direction, you have to take local roads...like in Vermont! (Just try to get to Maine from here!)

3. Please write a formal proof for the correctness of the closest points algorithm using strong induction. You should combine all of the pieces we discussed in class into a proof that is easy to read and understand. You may use figures (pictures) in your proof, but you should clearly explain what is happening in the figure using English. The goal of this problem is to clearly and concisely explain complex mathematical/algorithmic ideas in English. I would recommend typing your proof so that it is easy to make edits. You should not turn in the first version you write - make sure you reread and make changes for clarity and correctness.

For reference, my proof is about a page typed. In your proof, please reference to the following algorithm:

`ClosestPair`(P) (where P is an array containing x-coordinates and y-coordinates of n points, where no two points have the same x- or y-coordinate.)

Step 1: If $|P| \leq 3$ use brute force search and return closest distance.

Step 2: Sort by x-coordinate into sets L and R

Step 3: $\delta = \min\{$`ClosestPair`$(L),$ `ClosestPair`$(R)\}$

Step 4: Create Y_δ, an array of points within δ of midline between L and R, sorted by y-coordinate.

Step 5: Loop through elements of Y_δ, and calculate distance from each point to next 7 points, keeping track of δ', the smallest distance found.

Step 6: Return $\min\{\delta, \delta'\}$

4. In 3 dimensions, the distance between two points $p_i = (x_i, y_i, z_i)$ and $p_j = (x_j, y_j, z_j)$ is $D(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$. In this problem, we'll adapt our 2D Closest Points algorithm to 3D points, and see if we can get the same runtime as 2D.

   (a) I can describe the general idea of our 2D Closest Points algorithm as follows: "For a small number of points, do a brute force search. Otherwise, divide the points into a left and right half, and recursively solve to find the closest distance in each half. Now we just need to check points that cross from one half to the other across the mid-*line*. However, we only need to worry about a region close the this midline, and so we end up being concerned with a *line*-like strip. So we use an approach similar to our algorithm for Closest Points in 1D (a line) to deal with this strip." Please give a similar description for a divide and conquer algorithm for Closest Points in 3D. Please make an attempt at this part before moving on to the next step.

   (b) On the final page of the problem set is pseudocode to solve the 3D Closest Points problem. What number should replace the "??" in Algorithm 2, line 5? (It should be a constant, like "7.") Please explain your reasoning. For this problem, choose the number that you can most easily explain. Do not worry about finding the smallest number possible.

   (c) **Challenge:** What is the smallest possible number you could choose in part (b)? Please justify.

Algorithm 1: `DivideFrontBack`(P)

Input : Set of 3D points P.

Output: The distance of the closest pair of points

1 If $|P| \leq 3$, brute force search;
2 Split points into front (F) and back (B) halves by z-coordinate around the midline z_{mid};
3 $\delta^* = \min\{$`DivideFrontBack`(F),`DivideFrontBack`$(B)\}$;
4 Create P_{δ^*}, an array of points whose z-coordinates are within δ^* of z_{mid}.;
5 Return `DivideLeftRight`$(P_{\delta^*}, \delta^*, z_{mid})$;

Algorithm 2: `DivideLeftRight`(P, δ^*, z_{mid})

Input : Set 3D points P, values δ^* and z_{mid}, such that all points in P have z-coordinate within δ^* of z_{mid}

Output: The distance of the closest pair of points in P, or δ^*, whichever is smaller

1 If $|P| \leq 3$, brute force search;
2 Split points into left (L) and right (R) halves by x-coordinate around the midline x_{mid};
3 $\delta = \min\{\delta^*,$`DivideLeftRight`(L, δ^*, z_{mid}),`DivideLeftRight`$(R, \delta^*, z_{mid})\}$;
4 Let Y_δ be the set of points sorted by y-coordinate whose x coordinate is within δ of x_{mid} and whose z coordinate is within δ of z_{mid};
5 Loop through the elements of Y_δ, checking the distance between each point and the next ?? points, and let δ' be the smallest distance found;
6 Return $\min\{\delta, \delta'\}$;