CS302 - Problem Set 3

1. Consider the following two examples of simple algorithms that can do harm through the weighting of their inputs:

   (a) "Data becomes a tool of profound social change or a weapon of political warfare depending on whose hands it is in. In the United States, racism has always been numerical. The decision to make every Black life count as three-fifths of a person was embedded in the electoral college, an algorithm that continues to be the basis our current democracy." Yeshimabeit Milner, Founder of Data for Black Lives, from An Open Letter to Facebook from the Data for Black Lives Movement

   (b) Read pages 2-5, 9-10 of the pdf of Chapter 3 of Weapons of Math Destruction by Cathy O'Neil. (I encourage you to read it the rest of the chapter if you are interested, but the key issues related to weighting inputs are in the pages listed.)

   Reflect on the following questions (and have a few thoughts ready to share on Monday 3/15):

   - What surprised you about these examples?
   - Which groups are harmed or benefit from the weighting of each of these algorithms, and how does that harm/benefit reinforce systemic inequities?
   - Milner says that data can "become a tool of profound social change." For each of these algorithms, how could weights have instead be chosen to work against inequitable power structures? Explain.
   - Can you think of another real-life "simple" algorithm (like an averaging algorithm, or a voting algorithm) that is problematic because of its choice of weights?

2. (a) In our first attempt at calculating the runtime of Closest Points, we got the following recurrence relation for the runtime:

$$T(n) = O(1) \text{ for } n \leq 3, \qquad T(n) = 2T(n/2) + O(n \log n). \tag{1}$$

   We can't use the tree method formula that you came up with in the last problem set to evaluate this recurrence relation because there is no constant d such that $n^d = n \log n$. However, if you go through the same series of steps that you went through in PSet 1, #1, with $O(n^d)$ replaced with $O(n \log n)$, and setting $a = 2$ and $b = 2$, you can evaluate this recurrence relation. Please do this. You will likely need to use properties of logs and the arithmetic series formula:

$$\sum_{i=1}^{n} i = n(n+1)/2. \tag{2}$$

(b) After using a presorting trick, we were able to improve the runtime of our closest points algorithm, and we achieved a runtime that follows the recurrence relation:

$$T(n) = O(1) \text{ for } n \leq 3, \qquad T(n) = 2T(n/2) + O(n). \tag{3}$$

For this recurrence relation, we can use the tree method formula from Problem Set 1, #1. Use that formula to evaluate the runtime. (You do not need to do the full derivation again; you can just use your final formula.)

(c) Please comment on how much we gain by the preprocessing trick. Is the added complexity of the algorithm worth the improvement in runtime?

3. I've updated the 3D algorithm from the last problem set to have an input that is presorted (see below).

(a) What is the runtime of this 3D closest points algorithm?

(b) Comment on the runtimes of the 2D and 3D closest points algorithm. If you had to guess, what do you think the runtime is in d-dimensional space? (Challenge - create an algorithm for the d-dimensional closest points problem.)

Algorithm 1: `DivideFrontBack`(X, Y, Z)

Input : X, Y, and Z: Lists of n points sorted by x-, y-, and z-coordinate respectively
Output: The distance of the closest pair of points

1 If $|X| \leq 3$, brute force search;
2 Split points into front and back halves by z-coordinate around the midline z_{mid}, to get X_F, X_B, Y_F, Y_B, Z_F, and Z_B;
3 $\delta^* = \min\{$`DivideFrontBack`(X_F, Y_F, Z_F),`DivideFrontBack`$(X_B, Y_B, Z_B)\}$;
4 Create X_{δ^*}, Y_{δ^*} and Z_{δ^*}, which are sorted arrays of points whose z-coordinates are within δ^* of z_{mid}.;
5 Return `DivideLeftRight`$(X_{\delta^*}, Y_{\delta^*}, Z_{\delta^*}, \delta^*, z_{mid})$;

Algorithm 2: `DivideLeftRight`$(X, Y, Z, \delta^*, z_{mid})$

Input : X, Y, and Z: Lists of n points sorted by x-, y-, and z-coordinate respectively, where all n points have z-ccordinates within δ^* of z_{mid}.
Output: The distance of the closest pair of points

1 If $|X| \leq 3$, brute force search;
2 Split points into left and right halves by x-coordinate around the midline x_{mid}, to get X_L, X_R, Y_L, Y_R, Z_L, and Z_R;
3 $\delta =$ $\min\{\delta^*,$`DivideLeftRight`$(X_L, Y_L, Z_L, \delta^*, z_{mid})$,`DivideLeftRight`$(X_R, Y_R, Z_R, \delta^*, z_{mid})\}$;
4 Let Y_δ be the set of points sorted by y-coordinate whose x coordinate is within δ of x_{mid} or 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 $\min\{\delta, \delta'\}$;

4. Suppose you have n events, where the ith event has a start time s_i and end time f_i, for $i \in \{1, \ldots, n\}$. Unfortunately, you only have one auditorium, and you can't schedule conflicting

events (events where a start time of one is between the start time and end time of another.) We would like to maximize the number of events that are held.

(a) Do you have any ethical concerns about implementing an algorithm to solve this problem?

(b) For each of the following greedy algorithms, create an example of a series of events (with start and end times) where the algorithm does not perform optimally.

i. At each iteration, pick the remaining event with the earliest start time.

ii. At each iteration, pick the remaining event that has the shortest time ($f_i - s_i$ is smallest.)

5. You have a shipping container that can hold W worth of weight (please ignore volume). You have n types of items that you can ship. You have w_i pounds of item i, and item i has a value of v_i dollars per pound. However, you can take a fraction of any item, so if you wanted, you could pack $p \times w_i$ pounds of item i, which would be worth $p \times w_i \times v_i$ dollars, where $0 < p \leq 1$. You would like to pack the shipping container so that the total value is as large as possible.

Consider a greedy algorithm that evaluates a function f for each item. Then the algorithm puts as much of the largest f-valued item into the container as possible. If there is still room remaining, the algorithm goes to the next largest f-valued item, and so on.

(a) Do you have any ethical concerns about implementing an algorithm to solve this problem?

(b) What function f should you use to rank items? (The obvious one is the one to go with!)

(c) Let's rename the items according to their f score so that item 1 has the highest f-score, item 2 the 2nd highest, and so on. (You may assume all f-values are unique.) To make the proof as simple as possible, you can suppose that with the optimal strategy we are always able to fit exactly all of the first m items into the shipping container, but not more. Prove using a proof by contradiction/exchange argument that any other strategy that doesn't put all of items $1, \ldots, m$ into the container is not optimal. (It is probably somewhat obvious that this greedy strategy is optimal - the point of this problem is to practice this proof strategy.)

(d) What is the runtime of this greedy algorithm?

6. Let $T(n)$ be the number of bit strings of length n where there are two consecutive 1s. Create a recurrence relation for $T(n)$.

This goal of this problem is to remind you how to solve problems recursively - a skill from 200 that will be crucial for our next paradigm, Dynamic Programming. I know it has been a while for some of you, so I've made a video about how to solve these types of problems. You don't have to watch, but it is available if you need it:

- Panopto
- Google Drive