CS302 - Problem Set 2

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

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

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, Problem 1, with $O(n^d)$ replaced with $O(n \log n)$, (and using $a = 2$ 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 preprocessing trick, the runtime was described by the following recurrence relation:

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

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

(c) Please comment on how much we gain by the preprocessing trick. Is the added complexity worth it?

2. 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 refer 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: $\delta = \min\{\texttt{ClosestPair}(L), \texttt{ClosestPair}(R)\}$

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 $\min\{\delta, \delta'\}$

3. 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 Closest Points algorithm to 3D points.

   (a) I can describe the general idea of the 2D Closest Points algorithm as follows: "For a small number of points, brute force search. Otherwise, divide the points into a left and right half, and recursively solve to find the closest distance in each half. Then check a vertical, line-like strip for close points that cross the midline using an approach that is similar to Closest Points on a line." 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.

   (d) What is the runtime of my algorithm?

   (e) Comment on the runtimes of 2D vs 3D closest points.

4. Suppose you have a line graph G on 6 vertices ($(v_1, v_2, v_3, v_4, v_5, v_6)$, connected in that order) with the following vertex weights:

$$w(v_1) = 3 \qquad w(v_2) = 7 \qquad w(v_3) = 10 \qquad w(v_4) = 5 \qquad w(v_5) = 4 \qquad w(v_6) = 5$$

   (a) Let G_i be the line graph on the first i vertices with the same weights as above. Let $S(G_i)$ be the max-weight independent set on G_i. What are $S(G_1), S(G_2), \ldots, S(G_6)$? (Note: your answers should be *sets.*)

   (b) For $i = 3, 4, 5, 6$, verify the recurrence relationship we discussed in class: that if $v_i \in S(G_i)$, then $S(G_i) = S(G_{i-2}) + v_i$ while if $v_i \notin S(G_i)$, then $S(G_i) = S(G_{i-1})$.

5. Approximately how long did you spend on this assignment (round to the nearest hour)?

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 $\delta^* = \min\{\texttt{DivideFrontBack}(X_F, Y_F, Z_F), \texttt{DivideFrontBack}(X_B, Y_B, Z_B)\}$;
4 Create $X_{\delta^*}$, $Y_{\delta^*}$ and $Z_{\delta^*}$, which are sorted arrays of points whose $z$-coordinates are within $\delta^*$
  of $z_{mid}$.;
5 Return $\texttt{DivideLeftRight}(X_{\delta^*}, Y_{\delta^*}, Z_{\delta^*}, \delta^*, z_{mid})$;
```

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
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 $\delta =$
  $\min\{\delta^*, \texttt{DivideLeftRight}(X_L, Y_L, Z_L, \delta^*, z_{mid}), \texttt{DivideLeftRight}(X_R, Y_R, Z_R, \delta^*, z_{mid})\}$;
4 Let $Y_\delta$ be the set of points sorted by $y$-coordinate whose $x$ coordinate is within $\delta$ of $x_{mid}$ or
  whose $z$ coordinate is within $\delta$ of $z_{mid}$;
5 Loop through the elements of $Y_\delta$, checking the distance between each point and the next ??
  points, and let $\delta'$ be the smallest distance found;
6 Return $\min\{\delta, \delta'\}$;
```