CSE 417 25au Homework 3: Divide and Conquer

Released: Friday, October 10, 2025 @ 11:30am

First due by: Friday, October 17, 2025 @ 11:59pm

Last resubmissions by: Wednesday, October 29, 2025 @ 11:59pm

Instructions

For Problems 5, 5X.2, and 6, you have four options for submission:

- **Film a video in which you explain your solution.** See the Homework Guide for more details.
- **Use LaTeX to type your solutions.** A template is provided in the "Tasks" page of the course website, if you like.
- **Use Google Docs or Microsoft Word to type your solutions.** If doing so, please use the Equation Editor to ensure that any equations are legible and easy to read.
- **Handwrite your solutions on paper or digitally.** Please write neatly and if on paper, scan in black/white mode, not grayscale.

We prefer either video or LaTeX, but accept any of the 4 options.

A few more reminders:

- **Submit all problems on Canvas.** Each problem should have its own submission. *Do not* submit one large file containing answers to several problems. A reminder that **Problem 5X.1 requires submitting both code and a mini-report**.
- **Suggested word counts are rough guidelines.** We won't actually count, but if your writing is verbose to the point of obscuring your main argument, we may ask you resubmit more concisely.
- **Review the collaboration policy in the syllabus**. Collaboration is encouraged but strict rules apply, and remember to cite your collaborators.
- If you don't finish in time, we encourage you to be honest and just upload what you have so far. Resubmission won't cost you anything, and we can give you timely feedback on your partial progress by submitting on time.

Happy problem solving!

Problem 5: Contact tracing

The purpose of this problem is to review and extend the algorithm that you saw for "closest pair of points".

During a pandemic, a group of people are gathering for a social event. Each person is standing at a point in the 2D plane, given to you as an array $A[1 \dots n]$ of points $p = (x, y)$. For simplicity, you may assume that no two people share the same x-coordinate or y-coordinate.

For public health purposes, the event organizers have enforced a rule that every $2\delta \times 2\delta$ square (including the boundary) must have no more than c people, for some appropriate constants δ and c. It is known that everyone is following this rule. For contact tracing purposes, the organizers would now like to find all pairs of people currently standing at distance $\leq \delta$ apart.

Recall that the distance between (x_1, y_1) and (x_2, y_2) is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. (You do not need to write this formula explicitly in your pseudocode, just saying the word "distance" or similar phrases is fine.)

1. Write pseudocode for a divide and conquer algorithm that solves this problem efficiently. As usual, you should not need to use hash maps or hash sets. (Hint: Think about the closest pair of points problem from class.) (suggested 12–20 lines)
2. For this problem, we will prove the correctness just for the main recursive case. Assume that the recursive call(s) in your algorithm produce the right answer, in other words, their result(s) contain all pairs at distance $\leq \delta$ apart, where both people in the pair belong to the input *of the recursive call*. Explain why the output of the main recursive case contains all pairs at distance $\leq \delta$ apart where both people belong to the overall input. Depending on your particular pseudocode, drawing a picture may be helpful. (suggested 75–200 words)
3. Use the Master Theorem to explain why:
   - Your algorithm runs in time $O(n \log n)$.
   - There are at most $O(n \log n)$ pairs of people standing $\leq \delta$ apart.

   If you code contains unbounded loops or early breaks, be sure to explain how many iterations it will actually run. (If not, that's fine too.) Depending on your particular pseudocode, drawing a picture for this may be helpful. (suggested 50–150 words)

(Video submissions are suggested to take 4–8 minutes total.)

Problem 5X: Contact tracing (Extensions)

This is an extension problem that builds on the ideas of Problem 5. **Pick one** of the following to complete:

1. Divide and conquer algorithms typically have faster asymptotic running times compared to simple brute force solutions, but in reality, the speed may only be faster for large inputs. For example, even though $10n$ is $O(n)$ and n^2 is $O(n^2)$, an algorithm that takes n^2 seconds is still faster than $10n$ seconds for all $n \leq 9$.

   In this extension, you will implement your solution to Problem 5 in Java. A simpler brute-force algorithm to solve the same problem is provided for you. Additionally, we are providing code to you that will benchmark the two solutions across randomly generated inputs (for fixed δ and c), and produces CSV-formatted text that says how long each algorithm takes depending on the input size.

   Run the benchmarking function for an appropriate range of input sizes. (You may need to experiment to find an appropriate range.) Graph the produced CSV file by importing it into Google Sheets, Excel, matplotlib, or your favorite chart-making software, and ensure that it is a proper chart (with labeled axes, units, appropriate scales, etc.). Place the chart in a document and additionally answer: On which input sizes is the brute force algorithm faster, and on which input sizes is the divide and conquer algorithm faster? For reproducibility, also write down what computer you have and its CPU model, or the age of the computer if you don't know. (Example: 2022 MacBook Pro with Apple M2 chip.)

   Submit both your code and your mini-report PDF together on Gradescope, which is linked in Canvas. Provided starter code is available both on the course website and Canvas. Do not use libraries beyond the standard Java API. Your code will be autograded, but your chart/mini-report will be manually graded.

2. In this problem, we will generalize the closest pair of points problem to three dimensions (what most applications use), which will use the algorithm that you derived in Problem 5 as a subroutine. Again, for simplicity, you may assume that no two people share the same x-, y-, or z-coordinate.

   (a) Suppose you have a set of points and divide them into two groups by a vertical plane $x = m$. Suppose that no two points both left of the plane or both right of the plane are closer than δ apart.

   Consider the all points in the slab $\{(x, y, z) \in \mathbb{R}^3 \mid |x - m| \leq \delta\}$, that is, all points within distance δ from the plane. Find a constant c such that every $2\delta \times 2\delta \times 2\delta$ cube in the slab contains at most c points, and explain why the bound is true for your c. (suggested 40–80 words)

   (b) Now project all the points into the y-z plane, in other words, delete their x-coordinates. Briefly explain why every $2\delta \times 2\delta$ square in the projection contains at most c points. (suggested 10–40 words)

   (c) Write pseudocode that solves the closest pair of points problem in three dimensions. Specifically, the input is a list of points $A[1 \dots n]$ (with at least 2 points) and your job is to output the two points that are closest together in either order. (suggested 10–20 lines)

Tips: Use the previous parts and Problem 5 to help you adapt the algorithm for 2D closest pair of points from class. When calling Problem 5, be sure to say what values of δ and c are being used. As usual, you should not need to use hash maps or hash sets. On input of size n, each recursive step should take $O(n \log n)$ time.

(Hint: Assume for free that after you project a point into the y-z plane, you can recover the original point in x-y-z space in $O(1)$ time. To do this, for example, you can track the points by index, and the indices will not change when you project—this is an implementation detail that you don't need to discuss in pseudocode. You can also assume a version of Problem 5 that returns pairs of indices rather than pairs of points.)

(d) An extended version of the Master Theorem allows us to deal with recurrences of the form $T(n) = aT(n/b) + O(n^k(\log n)^\ell)$, where a, b, k, and ℓ are constants. While we don't require you to know this in general, one of the useful cases is: if $a = b^k$ and $\ell \geq 0$, then $T(n) = O(n^k(\log n)^{\ell+1})$. Note that people in compute science often write $\log^\ell n$ instead of $(\log n)^\ell$ to avoid one set of parentheses, feel free to use either notation.

Use this extended Master Theorem to analyze the running time of your algorithm. (suggested 25–75 words)

(Video solutions are suggested to take 3–6 minutes total.)

Problem 6: Hosting a debate

The purpose of this problem is to develop a divide-and-conquer algorithm from scratch.

An election is coming up and a TV network wants to run a debate. To determine who is eligible to participate in the debate, the TV network decides to run a poll and invite all candidates who receive at least 10% of the responses.

The poll is conducted without predetermined options, so that people can say names as they wish. Because of this, different strings like "Bob Ferguson" and "Robert Ferguson" (the current governor of Washington state) may actually refer to the same person. Thus, techniques such as sorting and hashing will not accurately clump together all votes for a single person. To determine if two names actually refer to the same person, you can ask the user, who you can assume accurately determines this. Do not use any other methods to compare strings. In your pseudocode, if you write "if `name1 = name2`," "count the number of times `name` appears," or similar phrases, we will assume that these statements are implemented by asking the user questions.

To state the problem more specifically, the input is an array $A[1 \dots n]$ of names containing all the responses to the poll. Your job is to output a list of at most 10 people, each of which receive at least 10% of the responses, to invite to the debate. All the different names of a person should count towards the same person. If you determine that multiple names correspond to the same person, use any of their names in the output. An "E" or "S" grade requires an $O(n \log n)$ running time. For your information, an $O(n)$ time algorithm is also possible but not required for this problem.

1. Write the pseudocode for a divide-and-conquer algorithm that solves this problem. Please write in a concise, easy-to-read way. (suggested 5–12 lines)
2. For this problem, we will prove the correctness just for the main recursive case. Assume that the recursive call(s) in your algorithm produce the right answer, in other words, their result(s) contain everyone that appears at least 10% of the time in the input *of the recursive call*. Explain why the output of the main recursive case contains everyone that appears at least 10% of the time in the overall input. (suggested 50–150 words)
3. Assuming your algorithm is correct, write a recurrence for the number of times the user is prompted to compare names as a function of the input size n, as a proxy for the running time. Solve this recurrence via the Master Theorem to analyze the running time. (suggested 40–100 words)

(Video solutions are suggested to take 3–6 minutes total.)