CS 5150/6150 Graduate (Advanced) Algorithms
Fall 2022
Assignment 2: Divide and Conquer
Total Points: 90

Submission notes

- Due at 11:59 pm on **Thursday, September 22, 2022**.
- Solutions must be typeset (not hand-written or scanned).
- You should strive to write clear, concise solutions to each problem, ideally fitting on a page or less. The easier your work is to read & follow, the easier it is for the TAs to award you points.
- Upload a PDF version of your completed problem set to Gradescope.
- Teaching staff reserve the right to request original source/tex files during the grading process, so please retain these until an assignment has been returned.
- Please remember that for problem sets, *collaboration with other students must be limited to a high-level discussion of solution strategies.* If you do collaborate with other students in this way, you must identify the students and describe the nature of the collaboration at the top of your homework submission. **You are not allowed to create a group solution, and all work that you hand in must be written in your own words. Do not base your solution on any other written solution, regardless of the source.**

Instructions on How to Answer Algorithm Design Questions

Most homework questions in this semester will be algorithm design questions. Please read the following instructions on how to answer them properly (you may lose points if you do not follow the instructions).

1. **Algorithm Description** You are required to clearly describe the main idea of your algorithm.

2. **Pseudocode** Providing pseudocode for your algorithm is optional. However, pseudocode is very helpful for explaining algorithms clearly, and you are *strongly encouraged* to provide pseudocode for your algorithms. Additionally, having pseudocode available often leads to more partial credit for incorrect solutions, as the TAs can more easily identify correct ideas in your solution.

   Note: copying code from an implementation (e.g. python/java/c++) as "pseudocode" does not satisfy this requirement, as grading it is difficult, and key steps may be difficult to identify. In principle, pseudocode should be relatively easy to read while keeping all relevant operations of the algorithms. *You are encouraged to do your best to write good pseudocode, following examples given in class.*

3. **Correctness** You also need to explain/argue why your algorithm is correct, and handles all necessary cases – especially when the correctness of your algorithm is not obvious.

4. **Time Analysis** You are required to analyze the time complexity of your algorithm (i.e., provide a written justification for the claimed asymptotic runtime in terms of the algorithm's operations, not just state the time complexity). You may refer to your pseudocode in this analysis, as it often simplifies the descriptors (e.g. "the `for` loop on line 2").

1. **(20 points)** You are given k **sorted lists** $L_1, L_2, \ldots, L_k$ of numbers, with $1 \leq k \leq n$, such that the total number of the elements in all k lists is n. Note that each list may have a different number of elements. We assume that the elements in each list L_i, for any $1 \leq i \leq k$, are sorted in ascending order.

   Design a divide-and-conquer algorithm to output a sorted list L of all n numbers. Your algorithm should run in $O(n \log k)$ time (instead of $O(n \log n)$ time).

   The following gives an example. There are five sorted lists (i.e., $k = 5$). Your algorithm should output the sorted list L : 2,3,10,12,17,19,21,25,26,28,29,34,36,55,59,61,87,89.

   $L_1 : 3, 12, 19, 25, 36$

   $L_2 : 34, 89$

   $L_3 : 17, 26, 87$

   $L_4 : 28$

   $L_5 : 2, 10, 21, 29, 55, 59, 61$

   Note: An $O(n \log k)$ time algorithm would be better than an $O(n \log n)$ time one when k is sufficiently smaller than n. For example, if $k = O(\log n)$, then $n \log k = O(n \log \log n)$, which is strictly smaller than $n \log n$ (i.e., $n \log \log n = o(n \log n)$).

2. **(30 points)** Let $A[1 \cdots n]$ be an array of n *distinct* numbers (i.e., no two numbers are equal). If $i < j$ and $A[i] > A[j]$, then the pair $(A[i], A[j])$ is called an *inversion* of A.

   You are asked to answer the following questions.

   (a) List all inversions of the array: $\{4, 2, 9, 1, 7\}$. **(5 points)**

   (b) What array with elements from the set $\{1, 2, \ldots, n\}$ has the most inversions? How many inversions does it have? **(5 points)**

   (c) Give a divide-and-conquer algorithm that computes the number of inversions in array A in $O(n \log n)$ time. (**Hint:** Modify merge sort.) **(20 points)**

3. **(20 points)** Solve the following recurrences (you may use any of the methods we studied in class). Make your bounds as small as possible (in the big-O notation). For each recurrence, $T(n) = O(1)$ for $n \leq 1$. You must justify your work (provide the process for how you obtained your answers, not just the bounds).

   (a) $T(n) = 2 \cdot T(\frac{n}{2}) + n^3$

   (b) $T(n) = 4 \cdot T(\frac{n}{2}) + n\sqrt{n}$

   (c) $T(n) = 2 \cdot T(\frac{n}{2}) + n \log n$

   (d) $T(n) = T(\frac{3n}{4}) + n$

4. **(20 points)** You are consulting for a small computation-intensive investment company, and they need to solve the following problem. When running a simulation, the company looks at the price of a given stock on some n consecutive days in the past. Let's number the days $i = 1, 2, \ldots, n$; for each day i, they have a price $p(i)$ per share for the stock on that day. (We'll assume for simplicity that the price was fixed during each day.) Suppose during this time period, they wanted to buy 1000 shares on some day and sell all these shares on some (later) day. The simulation should determine an optimal strategy for making as much money as possible in this scenario. That is, when should they have bought and when should they have sold to maximize profit? If there was no way to make money during the n days, you should report this instead.

   For example, suppose $n = 5$, $p(1) = 9$, $p(2) = 1$, $p(3) = 5$, $p(4) = 4$, $p(5) = 7$. Then you should return "buy on 2, sell on 5" (buying on day 2 and selling on day 5 means they would have made \$6 per share, the maximum possible for that period).

   Clearly, there is a simple algorithm that takes $O(n^2)$ time: try all possible pairs of buy/sell days and see which makes them the most money. Your investment friends were hoping for something a little better. Design an algorithm to solve the problem in $O(n)$ time using the divide-and-conquer technique.