CS5050 Advanced Algorithms

Fall 2023

Assignment 2: Algorithm Analysis

Due Date: 11:59:59 p.m., *Monday, October 2, 2023*

Total Points: 110

Note: The assignment is expected to be much more difficult and time-consuming than Assignment 1, so please start early. Again, to answer each algorithm design question, please follow the guideline at the end of Assignment 1.

1. **(20 points)** Suppose there are two **sorted arrays** $A[1 \dots n]$ and $B[1 \dots n]$, each having n elements (note that they may NOT be integers) sorted in ascending order. Given a number x, we want to find an element $A[i]$ from A and an element $B[j]$ from B such that $A[i] + B[j] = x$, or report that no such two elements exist.

   Design an $O(n)$ time algorithm for the problem.

2. **(20 points)** Your are given k **sorted lists** $L_1, L_2, \dots, L_k$, with $1 \leq k \leq n$, such that the total number of the elements in all k lists is n. Note that different lists may have different numbers of elements. We assume that the elements in each sorted list L_i, for any $1 \leq i \leq k$, are already sorted in ascending order.

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

   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., $n \log \log n = o(n \log n)$).

   The following gives an example. There are five sorted lists (i.e., $k = 5$). Your algorithm needs to sort the numbers in all these lists.

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

   $L_2 : 34, 89$

   $L_3 : 17, 26, 87$

   $L_4 : 28$

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

3. **(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 $\{1, 2, \dots, n\}$ 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)**

4. **(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 $n \leq 1$.

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

5. **(20 points)** You are consulting for a small computation-intensive investment company, and they have the following type of problem that they want to solve. A typical instance of the problem is the following. They are doing a simulation in which they look at n consecutive days of a given stock, at some point 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. They want to know: When should they have bought and when should they have sold in order to have made as much money as possible? (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 time $O(n^2)$: 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 $O(n \log n)$ time. Your algorithm should use the divide-and-conquer technique.

   Note: The divide-and-conquer technique can actually solve the problem in $O(n)$ time. But such an algorithm is not required for this assignment. You may think about it if you would like to challenge yourself (but no bonus point this time).