CS5050 Advanced Algorithms

Fall Semester, 2017

Assignment 2: Divide and Conquer

Due Date: 8:30 a.m., Friday, Sept. 29, 2017 (**at the beginning of CS5050 class**)

Note: As for the last question of Assignment 1, for each of the algorithm design problem in all assignments of this class, you are required to clearly describe the main idea of your algorithm. Although it is not required, you may give the pseudo-code if you feel it is helpful for you to describe your algorithm. You also need to briefly explain why your algorithm is correct if the correctness is not that obvious. Finally, please analyze the running time of your algorithm.

1. **(20 points)** Your are given k sorted lists $L_1, L_2, \ldots, L_k$, 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).

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

   $L_1 : 3, 12, 19, 45, 34$

   $L_2 : 34, 89$

   $L_3 : 17, 26, 87$

   $L_4 : 28$

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

   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, when $k = O(\log n)$, then $n \log k = O(n \log \log n)$, which is strictly smaller than $n \log n$ (i.e. it is $o(n \log n)$).

2. 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.

   (a) List all inversions of the array $\langle 14, 12, 17, 11, 19 \rangle$. **(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. 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, $T(n)$ is constant for $n \leq 2$. **(20 points)**

   (a) $T(n) = 2 \cdot T(\frac{n}{2}) + n^4$.

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

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

   (d) $T(n) = T(\frac{2}{3} \cdot n) + n$.

4. **(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 over and over. 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.

Total Points: 90