CS5050 Advanced Algorithms

Fall Semester, 2019

Assignment 2: Divide and Conquer

Due Date: 8:30 a.m., Friday, Sept. 27, 2019 (**at the beginning of CS5050 class**)

Note: The assignment is expected to be much more difficult and time-consuming than Assignment 1, so please start early. Again, for each of the algorithm design problems in all assignments of this semester, please follow the rules at the end of Assignment 1.

1. **(20 points)** Suppose there are two **sorted arrays** $A[1 \ldots n]$ and $B[1 \ldots n]$, each having n elements. 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. You may assume that both arrays A and B are sorted in ascending order.

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

2. **(25 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.

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

   (b) 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. **(30 points)** Let $A[1 \ldots n]$ be an array of n elements and $B[1 \ldots m]$ another array of m elements, with $m \leq n$. Neither A nor B is sorted. The problem is to compute the number of elements of A that are smaller than $B[i]$ for each element $B[i]$ with $1 \leq i \leq m$. For simplicity, we assume that no two elements of A are equal and no two elements of B are equal.

   For example, let A be $\{30, 20, 100, 60, 90, 10, 40, 50, 80, 70\}$ of ten elements. Let B be $\{60, 35, 73\}$ of three elements. Then, your answer should be the following: for 60, return 5 (because there 5 numbers in A smaller than 60); for 35, return 3; for 73, return 7.

   (a) Design an $O(n \log n)$ time algorithm for the problem. **(5 points)**

   (b) Design an $O(nm)$ time algorithm for the problem. Note that this is better than the $O(n \log n)$ time algorithm when $m < \log n$. **(5 points)**

   (c) Improve your algorithm to $O(n \log m)$ time. Because $m \leq n$, this is better than both the $O(n \log n)$ time and the $O(nm)$ time algorithms. **(20 points)**

   Hint: Use divide and conquer. Since $m \leq n$, you cannot sort the array A because that would take $O(n \log n)$ time, which is not $O(n \log m)$ as m may be much smaller than n.

4. **(15 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) $T(n) = 2 \cdot T(\frac{n}{2}) + n \log n$.

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

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 of n days, 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 day 2, sell on day 5" (buying on day 2 and selling on day 5 means they would have made \$6 per share, the maximum possible for that period of five days).

   Here is a simple algorithm that takes $O(n^2)$ time: try all possible pairs of buy/sell days and see which makes 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: 110