CS5050 Advanced Algorithms

Fall Semester, 2015

Assignment 2: Divide and Conquer

Due Date: 8:30 a.m., Wednesday, Sept. 30, 2015 (**at the beginning of CS5050 class**)

Note: 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. Given an array $A[1,\ldots,n]$ of n elements, we consider the problem of sorting the elements in A in ascending order. In class, we discussed the merge-sort algorithm, which uses the divide-and-conquer technique. Now consider the following version of **three-way merge-sort** that also uses the divide-and-conquer technique.

   (1) In the "divide" step, we divide A into three equal-sized subarrays $A[1,\ldots,\frac{n}{3}]$, $A[\frac{n}{3}+1,\ldots,\frac{2n}{3}]$, and $A[\frac{2n}{3}+1,\ldots,n]$.

   (2) In the "conquer" step, we recursively sort each of the three subarrays. If each of the three subarrays has no more than three elements, then we sort it simply by comparisons.

   (3) In the "combine" step, we merge the three sorted subarrays into a single sorted list, which is the sorted list for the elements in the original array A.

   You are asked to do the following.

   (a) Give a linear time algorithm to merge the three sorted subarrays into a single sorted list. **(5 points)**

   (b) Model the running time of the three-way merge-sort algorithm by a recurrence. **(5 points)**

   (c) Solve your recurrence to obtain the running time of the three-way merge-sort. **(10 points)**

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)**

Note: 0 points will be given for this problem if your algorithm is **not** based on the divide-and-conquer strategy.

3. Solve the following recurrences (you may use any of the approaches we discussed 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. 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 $n = 3$, $p(1) = 9$, $p(2) = 1$, $p(3) = 5$. Then you should return "buy on 2, sell on 3" (buying on day 2 and selling on day 3 means they would have made \$4 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. **(20 points)**

   Note: 0 points will be given for this problem if your algorithm is **not** based on the divide-and-conquer technique.

Total Points: 90