CS 325 - Homework Assignment 2

Problem 1: *(6 points)* Give the asymptotic bounds for T(n) in each of the following recurrences. Make your bounds as tight as possible and justify your answers. *Assume the base cases T(0)=1 and/or T(1) = 1.*

a) $T(n) = 4T\left(\frac{n}{2}\right) + n$

b) $T(n) = 2T\left(\frac{n}{4}\right) + n^2$

Problem 2: *(6 points)* An array A[1 . . . n] is said to have a majority element if more than half of its entries are the same. The majority element of A is any element occurring in more than n/2 positions (so if n = 6 or n = 7, the majority element will occur in at least 4 positions). Given an array, the task is to design an algorithm to determine whether the array has a majority element, and, if so, to find that element. The elements of the array are not necessarily from an ordered domain like the integers, so there can be no comparisons of the form "is A[i] > A[j]?". (Think of the array elements as GIF files, say.) Therefore you cannot sort the array. However you can answer questions of the form: "does A[i] = A[j]?" in constant time.

a) Give a detailed verbal description for a **divide and conquer** algorithm to find a majority element in A (or determine that no majority element exists).
b) Give pseudocode for the algorithm described in part a)
c) Give a recurrence for the algorithm. Solve the recurrence to determine the asymptotic running time.

Note: A O(n) algorithm exists but your algorithm only needs to be O(nlgn).

Problem 3: *(3 points)* Stooge sort is a "bad" recursive sorting algorithm. Given an array A, the algorithm can be defined as follows:

Step 1: If the value at the leftmost position of the array is larger than the value at the rightmost position then swap values.

Step 2: If there are 3 or more elements in the array, then:

- Recursively call Stooge sort with the initial 2/3 of the array.
- Recursively call Stooge sort with the last 2/3 of the array.
- Recursively call Stooge sort with the initial 2/3 of the array again.

Give a recurrence for the number of comparisons executed by the stooge sort algorithm. Solve the recurrence to determine the theoretical running time of the stooge sort algorithm.

Problem 4: *(15 points)*

a) Implement the stooge sort algorithm from Problem 3 to sort an array/vector of integers. Implement the algorithm in the same language you used for the sorting algorithms in HW 1. Your program should be able to read inputs from a file called "data.txt" where the first value of each line is the number of integers that need to be sorted, followed by the integers (like in HW 1). The output will be written to a file called "stooge.txt".

b) Modify code - Now that you have verified that your code runs correctly using the data.txt input file, you can modify the code to collect running time data. Instead of reading arrays from the file data.txt and sorting, you will now generate arrays of size n containing random integer values from 0 to 10,000 to sort. Use the system clock to record the running times of each algorithm for ten different values of n for example: n = 5000, 10000, 15000, 20,000, …, 50,000. You may need to modify the values of n if the algorithm runs too fast or too slow to collect the running time data (do not collect times over a minute). Output the array size n and time to the terminal. Name the program stoogeTime.
Submit a copy of all your code files and a README file that explains how to compile and run your code in a ZIP file to TEACH. We will only test execution with an input file named data.txt.

c) Collect running times - Collect your timing data on the engineering server. You will need at least eight values of t (time) greater than 0. If there is variability in the times between runs of the same algorithm you may want to take the average time of several runs for each value of n. Create a table of running times for the algorithm.

d) Plot data and fit a curve - Plot the running time data you collected on an individual graph with n on the x-axis and time on the y-axis. You may use Excel, Matlab, R or any other software. What type of curve best fits the data set? Give the equation of the curve that best "fits" the data and draw that curve on the graph.

e) Comparison - Compare your experimental running times to the theoretical running time of the algorithm? Remember, the experimental running times were the "average case" since the input arrays contained random integers.

f) Combine - Plot the data from Stooge sort with the data from merge sort and insertion sort. If the scales are different you may want to use a log-log plot.