Problem 1: *(8 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) $T(n) = 3T(n-1) + 1$

b) $T(n) = T(n-2) + 2$

c) $T(n) = 9T\left(\frac{n}{3}\right) + 6n^2$

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

Problem 2: *(7 points)* The ternary search algorithm is a modification of the binary search algorithm that splits the input not into two sets of almost-equal sizes, but into three sets of sizes approximately one-third.

a) Verbally describe and write pseudo-code for the ternary search algorithm.
b) Give the recurrence for the ternary search algorithm
c) Solve the recurrence to determine the asymptotic running time of the algorithm. How does the running time of the ternary search algorithm compare to that of the binary search algorithm.

Problem 3: *(5 points)* Consider the following pseudocode for a sorting algorithm.

```
StoogeSort(A[0 ... n - 1])
          if n = 2 and A[0] > A[1]
                    swap A[0] and A[1]
          else if n > 2
                    m = ceiling(2n/3)
                    StoogeSort(A[0 ... m - 1])
                    StoogeSort(A[n - m ... n - 1])
                    Stoogesort(A[0 ... m - 1])
```

a) State a recurrence for the number of comparisons executed by STOOGESORT.
b) Solve the recurrence to determine the asymptotic running time.

Problem 4: *(15 points)*

a) Implement STOOGESORT 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.out".

b) <u>Modify code</u> - 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) <u>Collect running times</u> - 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) <u>Plot data and fit a curve</u> - 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) <u>Comparison</u> - 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) <u>Combine</u> - 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.