CS 325 HW 2 – 30pts

Note: Do not submit hand-written assignments.

Problem 1: *(6 pts)* Give the asymptotic bounds for T(n) in each of the following recurrences. Make your bounds as tight as possible and justify your answers.

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

b) $T(n) = 4T\left(\frac{n}{2}\right) + n^3$

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

Problem 2: *(4 pts)* How many times as a function of n (in Θ form), does the following PHP function echo "Print"? Write a recurrence and solve it.

```
function Algo1( $n )  {
        if ($n > 1) {
            Algo1($n/3);
            Algo1($n/3);
            Algo1($n/3);
            Algo1($n/3);
            for ($i = 1; $i <= $n; $i += 1) {
                echo " Print  ".$i." <br> ";
            }
            echo " <br>";
         } else {
            return 1;
         }
}
```

Problem 3: *(5 pts)* 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) Write pseudo-code a recursive ternary search algorithm

b) Let T(n) denote the running time of ternary on an array of size n. Write a recurrence relation for T(n).

c) Solve the recurrence relation to obtain the asymptotic running time.

Problem 4: *(5 pts)* The Mergesort3 algorithm is a variation of Mergesort that instead of splitting the list into two halves, splits the list into three thirds. Mergesort3 recursively sorts each third and then merges the thirds together into a sorted list by calling a function named Merge3.

a) Write pseudo-code for Mergesort3 and Merge3
b) Let T(n) denote the running time of Mergesort3 on an array of size n. Write a recurrence relation for T(n).
c) Solve the recurrence relation to obtain the asymptotic running time.

CS 325 HW 2 – 30pts

Problem 5: *(10 pts)*

a) Implement the Mergesort3 algorithm to sort an array/vector of integers. You must implement the algorithm in same language you used in homework 1. Name the program "Mergesort3". 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.

Example values for data.txt:

4 19 2 5 11

8 1 2 3 4 5 6 1 2

The output will be written to files called "merge3.txt".

For the above example the output would be:

2 5 11 19

1 1 2 2 3 4 5 6

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 an 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 these new program merge3Time.

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. Create a table of running times for each algorithm.

d) Plot data and fit a curve - For each algorithm plot the running time data you collected on a 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 each data set? Give the equation of the curves that best "fits" the data and draw that curves on the graphs.

d) Compare - Plot the data from Mergesort3 and Mergesort (from HW 1) together on a combined graph. Which algorithm runs faster? How does your experimental running times compare to the theoretical running times of the algorithms?

Submit to TEACH (in a ZIP file) copy of all your code files and a README file that explains how to compile and run your code on the Flip servers. We will test execution with an input file named data.txt.