CS 325 - Winter 2020
Homework #2

Problem 1.
Give the asymptotic upper bounds for $T(n)$ in each of the following recurrences. Show your work and explain how you solve each case.

Problem 1.a. (2 points)

- $T(n) = b \cdot T(n-1) + 1$ where b is a fixed positive integer greater than 1.

Problem 1.b. (2 points)

- $T(n) = 3 \cdot T(n/9) + n \cdot \log n$

Problem 2. (6 points)
Solve Exercise 4.1-5 from the 3rd Ed. of the textbook.

Problem 3.
Consider the recurrence $T(n) = 3 \cdot T(n/2) + n$.

Problem 3.a. (3 points)

- Use the recursion tree method to guess an asymptotic upper bound for $T(n)$. Show your work.

Problem 3.b. (3 points)

- Prove the correctness of your guess by induction. Assume that values of n are powers of 2.

Problem 4.
Consider the following pseudocode for a sorting algorithm, for $0 < \alpha < 1$ and $n > 1$.

```
badSort(A[0 ··· n − 1])
    if (n = 2) and (A[0] > A[1])
        swap A[0] and A[1]
    else if (n > 2)
        m = ⌈α · n⌉
        badSort(A[0 ··· m − 1])
        badSort(A[n − m ··· n − 1])
        badSort(A[0 ··· m − 1])
```

Problem 4.a. (3 points)

- Show that the divide and conquer approach of badSort fails to sort the input array if $\alpha \leq 1/2$.

Problem 4.b. (2 points)

- Does badSort work correctly if $\alpha = 3/4$? If not, why? Explain how you fix it.

Problem 4.c. (2 points)

- State a recurrence (in terms of n and α) for the number of comparisons performed by badSort.

Problem 4.d. (2 points)

- Let $\alpha = 2/3$, and solve the recurrence to determine the asymptotic time complexity of badSort.

Problem 5.

Problem 5.a. (3 points)

- **Implementation**: Implement badSort from Problem 4 to sort an array of integers. The value of α should be an input parameter to your program. Implement the algorithm in C/C++. 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. The output will be written to a file called "bad.out".

Problem 5.b. (3 points)

- **Modify code**: Modify the code to collect running time data. Call the new timing program badSortTime. 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 n = 5000, 10000, 15000, 20,000, ... for two values of $\alpha = 2/3$ and $\alpha = 3/4$. You may need to modify the values of n if an algorithm runs too fast or too slow to collect the running time data. Provide a table with the timing data.

Problem 5.c. (2 points)

- **Plot data and fit a curve**: Plot the running time data you collected for each value of $\alpha \in \{2/3, 3/4\}$ 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. How does your experimental running time compare to the theoretical running time of the algorithm?

Problem 5.d. (2 points)

- **Comparison**: Looking at the plots in the previous step for $\alpha \in \{2/3, 3/4\}$, which α provides better performance?

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.