CS 325 - Homework Assignment 2
Due Sunday at 11:59pm

Problem 1: 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) $T(n) = T(n-2) + 2$

b) $T(n) = 3T(n-1) + 1$

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

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

Problem 2: Consider the following algorithm for sorting.

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

a) Explain why the STOOGESORT algorithm sorts its input. (This is not a formal proof).

b) Would STOOGESORT still sort correctly if we replaced k = ceiling(2n/3) with m = floor(2n/3)? If yes prove if no give a counterexample. (Hint: what happens when n = 4?)

c) State a recurrence for the number of comparisons executed by STOOGESORT.

d) Solve the recurrence.

Problem 3: The quaternary search algorithm is a modification of the binary search algorithm that splits the input not into two sets of almost-equal sizes, but into four sets of sizes approximately one-fourth. Write pseudo-code for the quaternary search algorithm, give the recurrence for the quaternary search algorithm and determine the asymptotic complexity of the algorithm. Compare the worst-case running time of the quaternary search algorithm to that of the binary search algorithm.

Problem 4: Design and analyze a **divide and conquer** algorithm that determines the minimum and maximum value in an unsorted list (array). Write pseudo-code for the min_and_max algorithm, give the recurrence and determine the asymptotic running time of the algorithm. Compare the running time of the recursive min_and_max algorithm to that of an iterative algorithm for finding the minimum and maximum values of an array.

Problem 5: 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. Give a detailed verbal description and pseudocode for a **divide and conquer** algorithm to find a majority element in A (or determine that no majority element exists). Give a recurrence for the algorithm and determine the asymptotic running time.

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