CS2800 Design and Analysis of Algorithms
Problem Set #2 January 30, 2023

1. Solve the following recurrence relations.
   (a) $T(n) = T(n/2) + 2T(n/3) + 3T(n/4) + n^2$.
   (b) $T(n) = \sqrt{n}T(\sqrt{n}) + n$.

2. Consider the following algorithm for sorting elements in an array.

Algorithm 1: StoogeSort

input : An array $A[0, 1, \ldots, n-1]$
output: Array A in sorted order

```
1 if n = 1 then return
2 if n = 2 then
3     if A[0] > A[1] then swap(A[0], A[1])
4 else
5     m = ⌈2n/3⌉
6     StoogeSort(A[0, 1, ..., m − 1])
7     StoogeSort(A[n − m, ..., n − 1])
8     StoogeSort(A[0, 1, ..., m − 1])
```

   (a) Prove that the Stooge-sort algorithm correctly sorts an array of n numbers.
   (b) What will happen if we replace $m = \lceil 2n/3 \rceil$ with $m = \lfloor 2n/3 \rfloor$ in the algorithm? Either prove that the algorithm sorts correctly or give a counter-example.
   (c) Write the recurrence for the running time $T(n)$ of the StoogeSort algorithm and solve it.

3. Is it asymptotically faster to square an n digit number than multiplying two n digit numbers? If you think it is true, give an algorithm. If you feel that it is incorrect, justify your answer.

4. The *Hadamard matrices*, denoted by $H_0, H_1, H_2, \ldots$, are defined as follows:
   - $H_0 = [1]$ is a 1×1 matrix.
   - H_k is a $2^k \times 2^k$ matrix defined as follows.

$$H_k = \begin{pmatrix} H_{k-1} & H_{k-1} \\ H_{k-1} & -H_{k-1} \end{pmatrix}$$

   (a) Write down the matrix H_3.
   (b) For $n = 2^k$, let $\mathbf{x}$ be an $n \times 1$ vector. Give an algorithm to obtain the product $H_k\mathbf{x}$. Note that the naive approach gives an $O(n^2)$-time algorithm. Can you improve this using a divide-and-conquer algorithm.

5. An array $A[1, 2, \ldots, n]$ is said to have a majority element if strictly more than half of the elements are the same. Observe that there will exist at most one majority element in an array. The entries of the array need not necessarily be integers, and hence you cannot compare the elements. You are only allowed queries of the form "Is $A[i] = A[j]$"?

   (a) Consider the following divide-and-conquer algorithm: First divide the array into two parts $A[1, 2, \ldots, n/2]$ and $A[n/2 + 1, \ldots, n]$ and find their majority element. How does finding the majority element of these two arrays help us obtain the majority element of A, if it exists. Write a pseudocode, and argue that your algorithm runs in time $O(n \log n)$.

   (b) Here is another divide-and-conquer strategy that gives an $O(n)$-time algorithm: First, pair the elements into groups of two arbitrarily. For each group of two, if both the elements are different then discard them. Otherwise, keep just one copy. Show that the new set of elements has a majority element iff the old set of elements had a majority element. Consequently, obtain an $O(n)$-time algorithm to find the majority element, if it exists.