Problem 1. [25 pts] Consider the following more complex variant of the Tower of Hanoi puzzle. The puzzle has a row of $k$ pegs, numbered from 1 to $k$. In a single turn, you are allowed to move the smallest disk on peg $i$ to either peg $i - 1$ or peg $i + 1$, for any index $i$; as usual, you are not allowed to place a bigger disk on a smaller disk. Your mission is to move a stack of $n$ disks from peg 1 to peg $k$.

(a) Describe a recursive algorithm for the case $k = 3$. Exactly how many moves does your algorithm make?

(b) Describe a recursive algorithm for the case $k = n + 1$ that requires at most $O(n^2)$ moves. [Comment: you can try to come up with an algorithm that requires $O(n^3)$ moves first.]

Problem 2. [25 pts]

(a) Prove that the following algorithm actually sorts its input!

(b) Would STOOGESORT still sort correctly if we replaced $m = \lceil 2n/3 \rceil$ with $m = \lfloor 2n/3 \rfloor$? Justify your answer.

*Some of the problems are from the text book. Looking into similar problems from the book, chapters 1 and 2 is recommended.*
(c) State a recurrence (including the base case(s)) for the number of comparisons executed by StoogeSort.

(d) Solve the recurrence, and prove that your solution is correct. [Hint: Ignore the ceiling.]

(e) Prove that the number of swaps executed by StoogeSort is at most $(n/2)$.

Problem 3. [25pts] The input is a two dimensional $n \times n$ array $A$ with the following properties:

- $A$ is composed of distinct integers,
- Each row of $A$ is sorted in ascending order, and
- Each column of $A$ is sorted in ascending order.

Design an algorithm to find the $k$th smallest element in $A$. Analyze the running time of your algorithm as a function of $n$. For full credit the running time of your algorithm must be a polynomial of log $n$.

Problem 4. [25pts] Describe efficient algorithms for the following variant of the text segmentation problem. Assume that you have a subroutine IsWord that takes an array of characters as input and returns True if and only if that string is a “word”. Analyze your algorithms by bounding the number of calls to IsWord.

- Given an array $A[1 \cdot n]$ of characters, compute the number of partitions of $A$ into words. For example, given the string ARTISTOIL, your algorithm should return 2, for the partitions ARTIST $\cdot$ OIL and ART $\cdot$ IS $\cdot$ TOIL.