**Question 2.5 from DPV:** Solve the following recurrence relations and give a Θ bound for each of them.

1. \( T(n) = 2T(n/3) + 1 \)
2. \( T(n) = 5T(n/4) + n \)
3. \( T(n) = 7T(n/7) + n \)
4. \( T(n) = 9T(n/3) + n^2 \)
5. \( T(n) = 8T(n/2) + n^3 \)
6. \( T(n) = 49T(n/25) + n^{3/2} \log n \)
7. \( T(n) = T(n-1) + 2 \)
8. \( T(n) = T(n-1) + n^c \), where \( c \geq 1 \) is a constant
9. \( T(n) = T(n-1) + c^n \), where \( c > 1 \) is some constant
10. \( T(n) = 2T(n-1) + 1 \)
11. \( T(n) = T(\sqrt{n}) + 1 \)

**question: stooge sort**

1. Explain why the following algorithm sorts its input.
   ```
   STOOGESORT(A[0...n-1])
   if n = 2 and A[0] > A[1]
       swap A[0] and A[1]
   else if n > 2
       k = \lceil 2n/3 \rceil
       STOOGESORT(A[0...k-1])
       STOOGESORT(A[n-k...n-1])
       STOOGESORT(A[0...k-1])
   ```

2. Would STOOGESORT still sort correctly if we replaced \( k = \lceil 2n/3 \rceil \) with \( m = \lfloor 2n/3 \rfloor \)? *(Hint: what happens when \( n = 4 \)?)*

3. State a recurrence for the number of comparisons executed by STOOGESORT.
4. Solve the recurrence. Simplify your answer.

**question: DFS numbers to DFS tree**

1. In her characteristic absentminded-ness, Professor Vergessen lost her complete 27-node binary tree. Thankfully, she still has an ordering of the nodes (each labelled with a letter of the German alphabet) by preorder and postorder:
   ```
   pre-order I Q J H L E M V O T S B R G Y Z K C A ß F P N U D W X
   post-order H E M L J V Q S G Y R Z B T C P U D N F W ß X A K O I
   ```
   Draw Professor Vergessen’s tree (with the root at the top and such that DFS visits left children before right children). Recall that a complete binary tree is one in which each node has either no children or two children.
2. Argue that the following recursive algorithm will reconstruct a binary-tree given its pre-order and post-order node sequences. You may assume that DFS was started at the unique node of degree 2 (the root).

Tree-ify(pre, post)

if pre and post are empty return nil
otherwise
    \(i\) be such that post\([i]\) = pre\([2]\) (by linear search)

    pre-left = pre\([1, \ldots, i]\)
    post-left = post\([1, \ldots, i]\)
    left[pre\([1]\)] = Tree-ify(pre-left, post-left)

    pre-right = pre\([i + 1, \ldots, \text{length}(pre) - 1]\)
    post-right = post\([i + 1, \ldots, \text{length}(pre) - 1]\)
    right[pre\([1]\)] = Tree-ify(pre-right, post-right)

Hints:

- The recursive procedure should take two arrays (giving the pre- and post-order of a tree) as input and return the parent of the tree represented by these arrays.
- The tree is represented by designating a left and right child for each non-leaf. If a node is a leaf, then the children may be designated as ‘nil’. You may assume that left and right children are indicated/stored globally.
- Given pre- and post-order sequences, how can you determine the root and the root’s left and right children?
- Given the root and the left and right children, how can you break the pre- and post-order sequences in to pre- and post-order sequences for the left and right subtrees?

3. What is the asymptotic running time of your algorithm?