Question 2.5 from DPV

(a) T (n)=2T (n/3) + 1.
Θ bound: Θ(1).

(b) T (n)=5T (n/4) + n.
Θ bound: $\Theta(n^{\log_4 5})$.

(c) T (n)=7T (n/7) + n.
Θ bound: Θ(n log n).

(d) T (n)=9T (n/3) + n^2 .
Θ bound: $\Theta(n^2 \log n)$.

(e) T (n)=8T (n/2) + n^3 .
Θ bound: $\Theta(n^3 \log n)$.

(f) T (n) = 49T (n/25) + $n^{3/2}$ log n.
Θ bound: $\Theta(n^{3/2}\log n)$.

(g) T (n)= T (n − 1) + 2.
Θ bound: Θ(n).

(h) T (n)= T (n − 1) + n^c .
Θ bound: $\Theta(n^{c+1})$.
Lower bound: consider the second half of the series

$$1^c + 2^c + ... + n^c \geq (n/2)^c + (n/2)^c + ... + (n/2)^c = (n/2)^{c+1}.$$

$$\text{Upper bound: } 1^c + 2^c + ... + n^c \leq n^c + n^c + ... + n^c = n^{c+1}.$$

(i) T (n)= T (n − 1) + c^n, c > 1.
Θ bound: $\Theta(c^n)$. Use Geometric Series Lemma.

(j) T (n)=2T (n − 1) + 1.
Θ bound: $\Theta(2^n)$.

(k) T(n)= T(n) + 1.
Θ bound: Θ(log logn).
At level i of the recursion tree, the problem size is $n^{(1/2)^\wedge i}$. Assume at the lowest level, the problem size equals a, where a is a number very close to 1. Then the height of the recursion tree would be $\log_2 \log n$ ($n^{(1/2)^\wedge i} = a$, $2^2 = \log_a n$). At each level, the time complexity is O(1), so the total complexity is Θ(log log n).

Question: stooge sort

1. Explain why the following algorithm sorts its input.

$$\text{STOOGESORT}(A[0\ldots n-1])$$
$$\quad \text{if } n = 2 \text{ and } A[0] > A[1]$$
$$\qquad \text{swap } A[0] \text{ and } A[1]$$
$$\quad \text{else if } n > 2$$
$$\qquad k = \lceil 2n/3 \rceil$$
$$\qquad \text{STOOGESORT}(A[0\ldots k-1])$$
$$\qquad \text{STOOGESORT}(A[n-k\ldots n-1])$$
$$\qquad \text{STOOGESORT}(A[0\ldots k-1])$$

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.

solution:

1. The base case either has one element or two elements, which are correctly sorted. The three recursive calls overlap by >n/3 elements (by the rounding-up choice). Call these elements the overlap elements, the first n/3 elements the prefix elements and the last n/3 elements the suffix elements.

   After the first recursive call, the prefix elements are smaller than the overlap elements. After the second recursive call the overlap elements are smaller than the suffix elements; the suffix elements are sorted. So the prefix elements are smaller than the suffix elements. The final recursive call sorts the prefix and overlap elements.

   (*4 pts*)

2. No. A counterexample should be given. For example, consider the input list [0 3 1 2]. n = 4 and $\lfloor 2n/3 \rfloor = 2$ so $A[0 \ldots k-1] = [0\ 3]$ which does not change in the recursive call and $A[n-k \ldots n-1] = [1\ 2]$ does not change in the recursive call. The list does not get sorted.
   (*2 pts*)

3. $T(n)=3T(2n/3) + O(1)$ (*2 pts*)

4. $T(n)= O(n^{\log_{3/2} 3})$ (*2 pts*)

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,...,i]
        post-left = post[1,...,i]
        left[pre[1]] = TREE-IFY (pre-left, post-left)

        pre-right = pre[i+1,...,length(pre) – 1]
        post-right = post[i+1,...,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?

solution:

1 EASY

2 (indexes start at 1)

```
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,...,i]
    post-left = post[1,...,i]
    left[pre[1]] = TREE-IFY (pre-left, post-left)

    pre-right = pre[i+1,...,length(pre) − 1]
    post-right = post[i+1,...,length(pre) − 1]
    right[pre[1]] = TREE-IFY (pre-right, post-right)
```

3. Let $T(n)$ be the time required to build an n node tree. There are two recursive calls with n and nr nodes each such that $n + nr = n - 1$. The non-recursive time is $O(n)$ as dominated by the linear search.

   So, $T(n)= T(n)+ T(nr)+ O(n)$.

   Using the recursion tree method, we see that at each level, the total number of nodes in each level decreases by at least one in each recursive call (because $n + nr = n - 1$). In the worst case, the depth of the recursion is $O(n)$ for a total running time proportional to $1 + 2 + 3 + 4 + \cdots + n$,which is $O(n^2)$.