[30 %] Question 1. [Proving recurrence relations] For each of the following recurrence relations:

\[
T(n) = \begin{cases} 
1 & \text{if } n = 1 \\
T(2n/3) + 1 & \text{otherwise}
\end{cases} \quad (1)
\]

\[
T(n) = \begin{cases} 
1 & \text{if } n = 1 \\
3T(n/4) + n & \text{otherwise}
\end{cases} \quad (2)
\]

You may show only the upper bound (i.e. big-O) portion of the work, in:

**Part (a)** Solve it, for instance by using recurrence trees. Alternative methods to do this are the “substitution method” and the “iteration method”. Do not give a formal proof, but rather show how to guess the solution.

**Part (b)** Give a proof of the solution found in Part (a), which does not use the Master Theorem. Your proof may assume that \( n \) is a power of some constant base. Note that something similar to the second part of the proof on the merge sort handout would make this proof complete, but you do not have to do it.

**Part (c)** Apply the Master Theorem to obtain the solution of the recurrence relation. Note that his should confirm Parts (a) and (b).

NB: Recall the definition of a geometric series and the formula for the sum of its first \( k \) terms.

[35 %] Question 2. [Recursive list implementation] Implement in Java, with recursive algorithms, the six basic lists operations that were assumed on page 1 of the slides from Lecture 3-3.

[35 %] Question 3. [Arithmetic using a stack] Develop an algorithm and write a corresponding Java class that reads a String from the input and evaluates it as a mathematical expression made of non-negative integers, “\%”, “\*”, “\^”, “\("”, “\)”. Normal priorities of “\)” and of “\^” over “\*”, and over “\%” should be implemented.
Examples:

Input: “1 ∧ (7 * 5)”
Output: 1
Input: “2 * 3 ∧ 5”
Output: 486
Input: “2 ∧ (7 * 5%3)”
Output: 4
Input: “(3 ∧ (35%6)) * (7 * 5)”
Output: 8505
Input: “(3 * 1) ∧ (3 * 4)%21”
Output: 15

Suggestion: to analyze the String use two Stacks; one to store integers and one to store operators. You may use the provided code for the implementation of stacks in Java.

[10 %] Bonus. Modify your implementation in Question 3 so that it handles arbitrary long integers. Suggestion: use the recursive list data type from the slides of Lecture 3-3.