Recurrences, Master Theorem, tree and table method, induction.

1. Given the recurrences
   a. T(N) = 3*T(N/5) + N + lgN
   b. T(N) = 4*T(N/2) + $\sqrt{N}$
   c. T(N) = 6*T(N/5) + N^3
   d. T(N) = 6*T(N/5) + 7

   Find their Θ time complexity with the tree method. You must show the tree and fill out the table like we did in class.
   Find their Θ time complexity with the Master Theorem method.
   For a. since N+lgN = Θ(N), solve for T(N) = 3*T(N/5) + N
   When applying the master theorem, make sure you give a value for ε and check all the conditions (especially for case 3).
2. Use the substitution method (induction) to show that T(N) = 2T(N/2) + N^3 is O(N^3). Let **T(0)=4.**
   Need to find c and N_0 s.t. T(N) ≤cN^3 for all N≥N_0.
   Base cases:
   T(0) = 4 fails: (we need 4≤c*0 = 0)
   T(1) = 2T(1/2)+1^3 = 2*4+1 = 9 need: 9≤c1^3.=> holds for all c≥9.
   T(2), T(3), T(4)… use T(1) or higher in their recurrence. Try to prove them using the recursive case.
   Recursive case:
   T(N) = 2T(N/2) + N^3 ≤ 2*c*$(N/2)^3$ + N^3 =N^3[1+(c/4)]
   Need T(N) ≤ cN^3 => need: N^3[1+(c/4)] ≤ cN^3 => N^3[c-1-(c/4)]≥0
   Since N^3 ≥0 for all N≥0 => need [c-1-(c/4)]≥0=>(¾)c-1≥0 => c≥1/(3/4) => c≥4/3.
   Keep the larger of the c (from base case and recursive case)=> c = max{9, 4/3} => c = 9 (or any value larger than 9). N_0 = 1 (first value of N for which the inequality T(N) ≤cN^3 holds.
3. CLRS 3rd edition (textbook)
   a. Reminder: The book calls ‘substitution method’ what we called ‘induction method’.
   b. Page 87: 4.3-1 – Consider every one of the three methods. Can you apply it? If yes, solve with that method, if no, explain why.
   c. Page 87, 4.3-7
   d. page 92, 4.4-1, 4.4-2, 4.4-3 (NOT with the tree on the given recurrence. Instead, use a similar but easier recursion, and guess it with the Master theorem or the tree and prove it with induction).
   e. page 96, 4.5-1

41. (6 points) A recursive algorithm for processing arrays works as follows: it first does some processing which takes N^2 and allows it to split the array in 3 equal parts. Next the algorithm applies itself again to each one of those smaller arrays.

If the array has 0, 1, or 2 elements the algorithm executes 5 instructions and finishes. Give the recurrence formula (including the base case) for this algorithm.

T(0) = T(1) = T(2) = 5 (Also ok to use c instead of 5)

$T(N) = 3T(N/3) + cN^2$

P5 . (Exam 1, Fall 15, 002)

a) (5 points) Is anything wrong with the following recurrence definition?

g(0) = N   **Yes. g(0) cannot be N. Incorrect even from a pure mathematical point of view.**

g(N) = g(N-1) + c

P6. (Exam 1, Fall 15, 002)

```
int foo(int * array, int N)
{
  if (N == 0) return 0;
  int result = 0;
  int b, c;
  for (b = 0; b < N/4; b++)
    for (c = N; c > 1 ; c = c/2)
      result = result + array[b] * array[c];
  return result + foo(array, N-1);
}
```

Give the recurrence formula (including the base case).

T(0) = d

T(N) = T(N-1) + d(N/4)lgN