Recurrences, Master Theorem, tree and table method, write the recurrence for recursive functions.

Conventions: When giving answers (online):

- Do NOT put any spaces in your answers
- For level TC (time complexity) give the answer in the form: number_of_nodes*1_node_TC without any spaces. E.g. the tree for the recurrence **T(n) = 4T(n/2) + cn** has at level TC at level t: **4^t*c(n/2^t)** where 4^t is the number of nodes per level and c(n/2^t) is the TC of one node at level t.

P1. 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. <u>You must show the tree and fill out the table like we did in class.</u>
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 indicate the case as done and in class and check all the conditions (especially for case 3).

P2. Solve the recurrence T(n) = 2T(n-3)+c, where T(n) = c for all n≤3.

The tree method is needed (Cannot apply Master Theorem). At level t we will have

- **problem size: n-3t => last level k = n/3**
- **2^t nodes (at this level)**
- **TC of 1 node: c**
- **=> level TC will be: 2^t*c.**

T(n) is the sum over levels from 0 to k =>

T(n) = 2^0*c+2^1*c+2^2*c+….+$2^{n/3}$*c = c(2^0+2^1+2^3+…$2^{n/3}$) = c($2^{(n/3)+1}$-1)/(2-1) = Θ($2^{n/3}$).

P3. Can you solve the recurrence: $T(n) = T\left(\left\lfloor \frac{N}{2} \right\rfloor\right) + T\left(\left\lceil \frac{N}{2} \right\rceil\right) + cN$ with base cases $T(N) = c \; for \; N \leq 13$?
Here the symbols ⌊ ⌋, ⌈ ⌉ indicate rounded down and rounded up.

Yes. The Master Theorem can be used for rounded down and rounded up (in the recursive terms) and when for small problem sizes (up to a constant, here 13) a different subroutine is applied (here the subroutine has constant time complexity, c).

Therefore the given recurrence has the same answer as T(N) = 2T(N/2)+cN and T(1) = c which we have already solved.

P4. (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

P7. Short answer.

a) (5pts) Can you apply the master theorem for the recurrence **T(n) = 4T(n-2) + cn**? Justify. (E2,Fall 18)
No. The it has subtraction in the smaller problem size n-2

b) (6 pts) Consider the tree for the recurrence **T(n) = 4T(n/2) + cn**. Fill in the answers regarding the tree for this recurrence:
Any internal node has __**4**___ children.
The **problem size** for a node on level 3 is ___ **$n/2^3$**___ (where the root is at level 0).
The **TC of a single** node on level 3 is ____**$c(n/2^3)$**____ (where the root is at level 0).

The **last level** is (if the answer involves a log, indicate the base for it) : k = ____ **$\log_2 n$** ____

How many nodes will the tree have at level t ? _____**4^t**_____

What is the **Level TC** at some level t? _____**$4^t * c(n/2^t)$** ___

c) (5 pts) Give a recurrence formula that will result in a tree that has the last level: k = N/4 **T(n) = T(n-4) + c**

d) *This problem if not covered in Fall 2020.* (4 pts) Mark (with X) the correct statement about the Tree (and table) method for solving recurrences as done in class:

__**X**__ it computes an estimate of the time complexity (but it does not completely prove it) (Because we did not use induction to prove any of the "formulas" used and also for the

_____ it computes and mathematically proves the correct answer.

e) (5 pts) Consider the recurrence: T(N) = T(N-7)+c. Assume the first applicable value for N is 0 (i.e. assume it is never applied to negative values).

How many base case(s) does this problem have? (2pts) : __**7**_____

List the values of N for the base case (3pts) : ___**0,1,2,3,4,5,6**_____

f) T/F: 3^{log_2(N)} = N **False**

P8. (10pts) Can you use the **Master theorem** to solve the recurrence: **T(n) = 4T(n/2) + n** ? If yes, solve it with this method (make sure you indicate the case. If no, show/say why you cannot use it.

T(n) = 4T(n/2) + n

a=4, b=2, $\log_b a = \log_2 4 = 2$.

p=1

$1<2$, => $p < \log_b a$ => M1 case2 => $T(N) = \Theta(N^{\log\_b(a)})$ => $T(N) = \Theta(N^2)$

P9. (8 pts) Write a recursive function that has the recurrence formula (for time complexity): T(N) = 2*T(N/3) + cN and base cases: T(N) = c (for all N≤2).

```
void foo(int N){
     int k;
     if (N<=2) return;
     for(k = 0; k<N; k++) {
          printf("B");
     }
     foo(N/3);
     foo(N/3);
}
```

P10. (30 pts) Use the tree method to compute the **Θ** time complexity for T(N) = 4T(N/4) + cN with T(1) = c .

Fill in the table below and finish the computations outside of it:

Level	Argument/ Problem size	Cost of one node	Nodes per level	Cost of whole level
0	N	cN	1	cN
1	N/4	cN/4	4	4*cN/4=cN
2	$N/4^2$	$cN/4^2$	4^2	$4^2*cN/4^2 = cN$
i	$N/4^i$	$cN/4^i$	4^i	$4^i*cN/4^i = cN$
k= $\log_4 N$ *Leaf level. Write k as a function of N.*	$1=N/4^k$ => $k=\log_4 N$	c (= $cN/4^k$)	4^k	$4^k*cN/4^k = cN$

Total tree cost calculation: ……$\sum_{i=0}^{log_4 N} cN = cNlog_4 N$ ……

T(N) = Θ(……$N\log_4 N$……)

Draw the tree.-Show **levels 0,1 and 2.** (Show just a few nodes at level 2) Show the problem size T(…) as a label next to the node and inside the node show the local cost (cost of one node) as done in class.

EXTRA: topic induction method (not covered and not required for test or quiz)

PExtra1. 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. $T(N) \leq cN^3$ for all $N \geq N_0$.
Base cases:
$T(0) = 4$ fails: (we need $4 \leq c*0 = 0$)
$T(1) = 2T(1/2)+1^3 = 2*4+1 = 9$ need: $9 \leq c1^3$.=> holds for all $c \geq 9$.
T(2), T(3), T(4)… use T(1) or higher in their recurrence. Try to prove them using the recursive case.
Recursive case:
$T(N) = 2T(N/2) + N^3 \leq 2*c*(N/2)^3 + N^3 = N^3[1+(c/4)]$
Need $T(N) \leq cN^3$ => need: $N^3[1+(c/4)] \leq cN^3$ => $N^3[c-1-(c/4)] \geq 0$
Since $N^3 \geq 0$ for all $N \geq 0$ => need $[c-1-(c/4)] \geq 0$=>(¾)c-1≥0 => $c \geq 1/(3/4)$ => $c \geq 4/3$.
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 $T(N) \leq cN^3$ holds.

PExtra2. 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 – the recurrences here only need the Master Theorem in order to be solved.