1. (20 points)

Algorithm Mystery(A: Array [i..j] of integer)   i & j are array starting and ending indexes

if i=j then return A[i]

else

    k=i+floor((j-i)/2)

    temp1= Mystery(A[i..k])

    temp2= Mystery(A[(k+1)..j]

    if temp1<temp2 then return temp1 else return temp2

(a) (1 points) What does the recursive algorithm above compute?

Returns the minimum number from the array.

(b) (6 points) Develop and state the two recurrence relations exactly (i.e., determine all constants) of this algorithm by following the steps outlined in L7-Chapter4.ppt. Determine the values of constant costs of steps using directions provided in L5-Complexity.ppt. Show details of your work if you want to get partial credit.

Step	Cost of execution
1. if $i = j$	3
2. return A[i]	4
3. else $k = i + \lfloor j-i/2 \rfloor$	8
4. temp1 = Mystery(A[i..k])	$T(N/2)$
5. temp2 = Mystery(A[k+1...j])	$T(N/2)$

Step	Cost of Execution
6. if temp1 < temp2	3
7. return temp1	1.
8. else return temp2	1.

Recurrence Relations :

$T(n) = 07 = \Theta(1)$ when $i = j$

$T(n) = 2T\left(\frac{n}{2}\right) + 16 = 2T\left(\frac{n}{2}\right) + O(1)$ when $i \neq j$

(c) (12 points) Use the Recursion Tree Method to determine the precise mathematical expression T(n) for this algorithm. First, simplify the recurrences from part (b) by substituting the constant "c" for all constant terms. Drawing the recursion tree may help but you do not have to show the tree in your answer; instead, fill the table below. Use the examples worked out in class for guidance. Show details of your work if you want to get partial credit.

You will need the following result: $\sum_{i=0}^{k} x^i = \frac{x^{(k+1)}-1}{x-1}$

Level	Level number	Total # of recursive executions at this level	Input size to each recursive execution	Work done by each recursive execution, excluding the recursive calls	Total work done by the algorithm at this level
Root	0	2^0	$n/2^0$	16	16×2^0
One level below root	1	2^1	$n/2^1$	16	16×2^1
Two levels below root	2	2^2	$n/2^2$	16	16×2^2
The level					

just above the base case level	$\log_2(n-1)$	$2^{\log_2(n-1)}$	$\frac{n}{2^{\log_2(n-1)}}$	16	$16 \times 2^{\log_2(n-1)}$
Base case level	$\log_2 n$	$2^{\log_2 n}$	$\frac{n}{2^{\log_2 n}}$	~~6~~ 7	$6 \times 2^{\log_2 n}$

$$T(n) = \sum_{i=0}^{\log_2(n-1)} (16 \times 2^i) + \cancel{10}\, 7 \times 2^{\log_2 n}$$

$$= 16 \left(\frac{2^{\log_2(n-1)+1} - 1}{2-1} \right) + 7n$$

$$= 16 \left(2^{[\log_2(n-1) + \log_2 2]} - 1 \right) + 7n$$

$$= 16 \left[2^{(\log_2 2(n-1))} - 1 \right] + 7n \quad = 16 \left[2(n-1)^{\log_2 2} - 1 \right] + 7n$$

$$= 23n - 32$$

(d) (1 points) Based on T(n) that you derived, state the order of complexity of this algorithm:

$O(n)$.

2. (10 points) T(n)=7T(n/8)+cn; T(1)=c. Determine the polynomial T(n) for the recursive algorithm characterized by these two recurrence relations, using the Recursion Tree Method. Drawing the recursion tree may help but you do not have to show the tree in your answer; instead, fill the table below. You will need to use the following results, where and b are constants and x<1:

$$a^{\log_b n} = n^{\log_b a}$$

$$\sum_{i=0}^{i=\infty} x^i = 1/(1-x) \ \ when\ x < 1$$

level	Level number	Total # of recursive executions at this level	Input size to each recursive execution	Work done by each recursive execution, excluding the	Total work at this level

				recursive calls	
Root	0	7^0	$n/8^0$	$cn/8^0$	$cn \times \left(\frac{7}{8}\right)^0$
1 level below	1	7^1	$n/8^1$	$cn/8^1$	$cn \times \left(\frac{7}{8}\right)^1$
2 levels below	2	7^2	$n/8^2$	$cn/8^2$	$cn \times \left(\frac{7}{8}\right)^2$
The level just above the base case level	$\log_8(n-1)$	$7^{\log_8(n-1)}$	$n/8^{\log_8(n-1)}$	$cn/8^{\log_8(n-1)}$	$cn \times \left(\frac{7}{8}\right)^{\log_8(n-1)}$
Base case level	$\log_8 n$	$7^{\log_8 n}$	$n/8^{\log_8 n}$	c	$c \times \left(\frac{7}{8}\right)^{\log_8 n}$

T(n) = $cn \sum_{i=0}^{\log_8(n-1)} \left(\frac{7}{8}\right)^i + c\left(\frac{7}{8}\right)^{\log_8 n}$

$< cn \sum_{i=0}^{\infty} \left(\frac{7}{8}\right)^i + c\left(7 \times 8^{-1}\right)^{\log_8 n}$

$= \frac{1}{1-(7/8)} cn + c\left[7 \times (-n)\right]^{\log_8 8}$

$= 8cn - 7c$

$\therefore\ T(n) = O(n)$

3. (11 points) Use the substitution method to prove the guess that $T(n) = O(n)$ is indeed correct when T(n) is defined by the following recurrence relations: T(n)=3T(n/3)+5; T(1)=5. At the end of your proof state the value of constant c that is needed to make the proof work.

Statement of what you have to prove:

$T(n) \le cn - 5$ where $n \ge n_0$.

Base Case proof: $T(1) = 5 \leq c\times1 - 5$

$5 \leq c - 5$

This is true when $c \geq 10$ and $n_0 = 1$

Inductive Hypotheses:

Assume. $T(n/3) \leq \frac{cn}{3} - 5$ when $n \geq n_0$.

Inductive Step:

We need to show that $T(n) \leq cn - 5$ when $n > n_0$

$T(n) = 3T(\frac{n}{3}) + 5 \leq 3\left[\frac{cn}{3} - 5\right] + 5$ — from inductive hypothesis

$\Rightarrow T(n) \leq cn - 15 + 5$

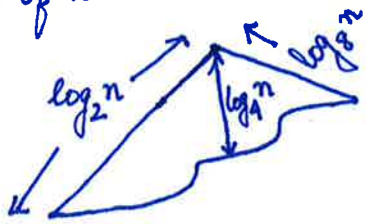
$\Rightarrow T(n) \leq (cn - 5) - 5$

$\therefore T(n) \leq (cn - 5)$

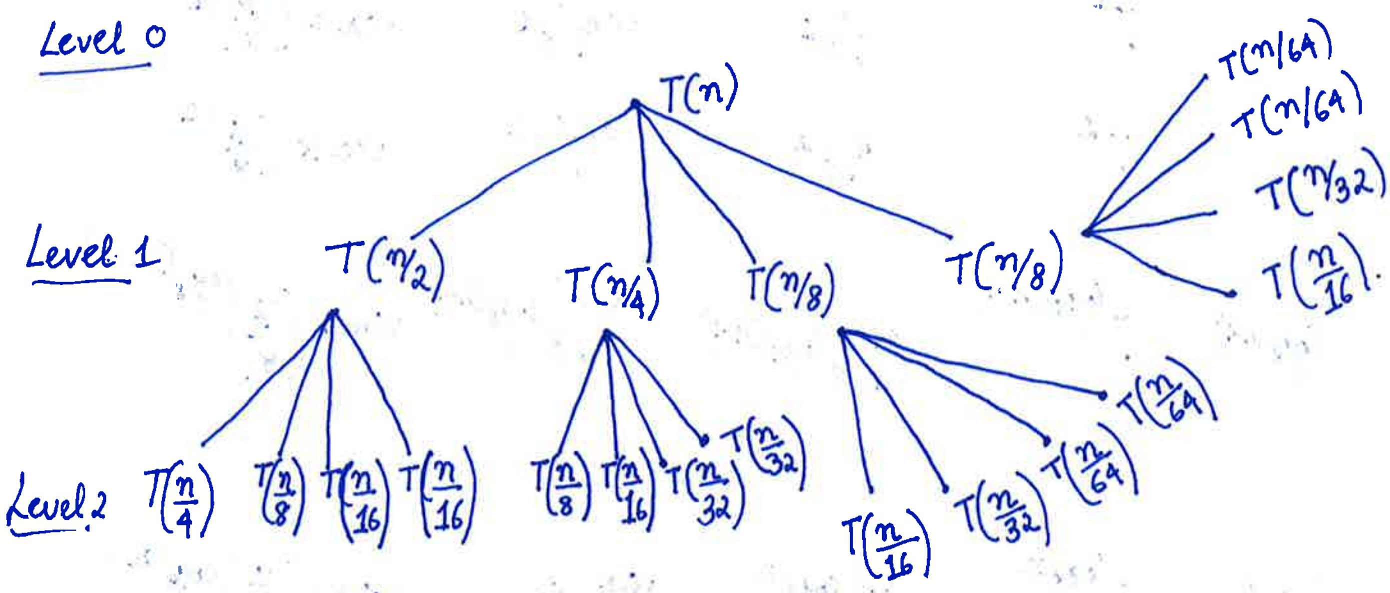
Value of c:

4. (16 points) Guess a plausible solution for the complexity of the recursive algorithm characterized by the recurrence relations T(n)=T(n/2)+T(n/4)+T(n/8)+T(n/8)+n; T(1)=c using the Substitution Method. (1) Draw the recursion tree to three levels (levels 0, 1 and 2) showing (a) all recursive executions at each level, (b) the input size to each recursive execution, (c) work done by each recursive execution other than recursive calls, and (d) the total work done at each level. (2) Pictorially show the shape of the overall tree. (3) Estimate the depth of the tree at its shallowest part. (4) Estimate the depth of the tree at its deepest part. (5) Based on these estimates, come up with a reasonable guess as to the Big-Oh complexity order of this recursive algorithm. Your answer must explicitly show every numbered part described above in order to get credit.

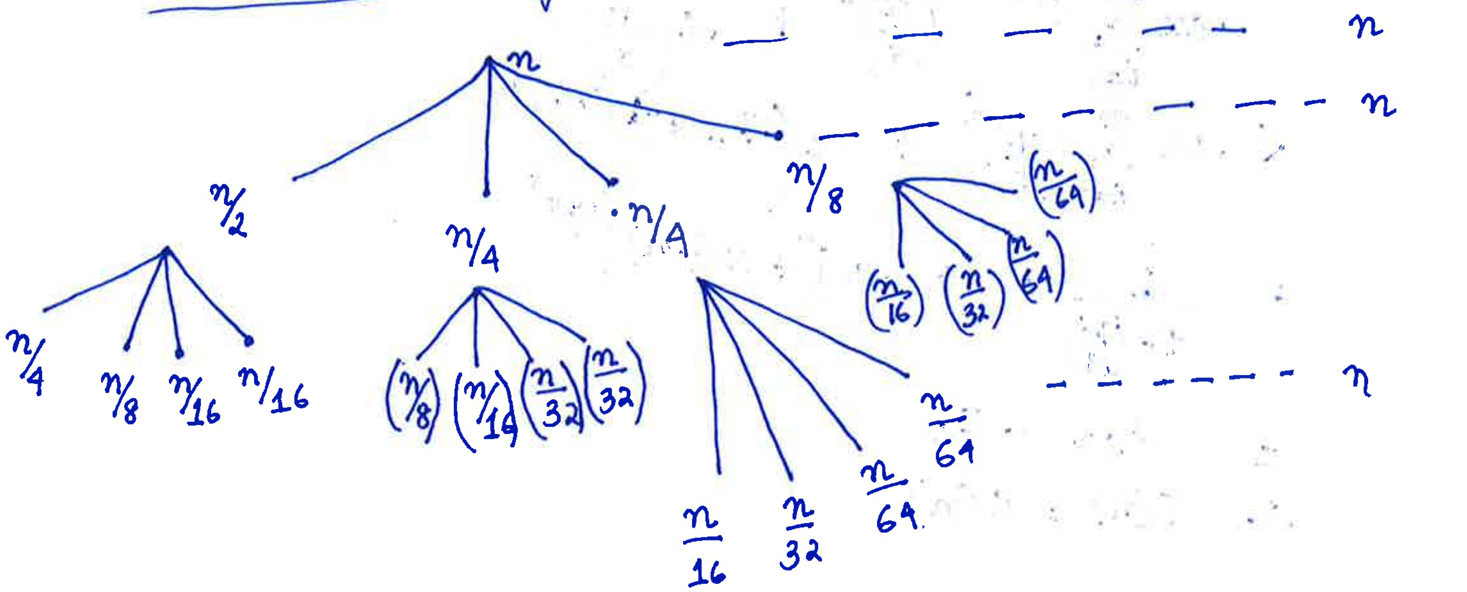
(2) Shape of the Recursion tree



Recursion tree showing the input size



Recursion tree showing the work done



(3) Depth of the tree at shallowest part

$\log_8 n$

(4) Depth of the tree at its deepest part

$\log_2 n$

(5) Guessed solution

$T(n) = O(n \log n).$

5. (10 points) Use the Substitution Method to prove that your guess for the previous problem is indeed correct.

Statement of what you have to prove:

$T(n) \leq cn\log n$

Base Case proof:

$T(1) = c$

For $n = 2$, $T(2) = T(1) + T(0.5) + 2T(0.75) + 2$

$= c + 2$

$T(2) = c+2 \leq 2c \times \log 2$

$\Rightarrow T(2) = c+2 \leq 2c. \Rightarrow T(2) = 2 \leq c.$

This is true if $c \geq 2$

Inductive Hypotheses:

Assume: $T(n/2) \leq \frac{cn}{2}\log\frac{n}{2}$; $T(\frac{n}{4}) \leq \frac{cn}{4}\log\frac{n}{4}$; $T(\frac{n}{8}) \leq \frac{cn}{8}\log\frac{n}{8}$.

Inductive Step:

We need to prove : $T(n) \leq cn\log n$ where $n > n_0$

$T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + 2T(\frac{n}{8}) + n \leq \frac{cn}{2}\log(\frac{n}{2}) + \frac{cn}{4}\log(\frac{n}{4}) + \frac{cn}{4}\log(\frac{n}{8}) + n$

$\leq \frac{cn}{2}\log(n) + \frac{cn}{4}\log(n) + \frac{cn}{4}\log(n) - \left[\frac{cn}{2}\log 2 + \frac{cn}{4}\log 4 + \frac{cn}{4}\log 8\right] + n$

$\leq cn\log n - \left[\frac{cn}{2} + \frac{cn}{2} + \frac{3cn}{4}\right] + n \leq cn\log n - \left(\frac{7cn}{4} - n\right)$

$\therefore T(n) \leq cn\log n$ for $\left[\frac{7cn}{4} - n\right] \geq 0 \Rightarrow \frac{7c}{4} \geq 1 \Rightarrow c \geq 4/7$

6. (9 points) Use the Master Method to solve the following three recurrence relations and state the complexity orders of the corresponding recursive algorithms.

(a) T(n)=2T(99n/100)+100n

$a = 2$, $b = 100/99$, $f(n) = 100n$

$\log_b a = \log_{100/99}(2) \cong 68.967.$

Case 1 applies here : $f(n) = 100n = O[n^{68.967-\epsilon}]$ for $\epsilon > 0$

$\therefore T(n) = \Theta(n^{68.967}).$

(b) T(n)=16T(n/2)+n^3lgn

$a = 16$, $b = 2$, $f(n) = n^3\log n$

$\log_b a = \log_2 16 = 4$

Case 1 applies here : $f(n) = n^3\log n = O[n^{4-\epsilon}]$ for $\epsilon > 0$

$\therefore T(n) = \Theta(n^4).$

(c) T(n)=16T(n/4)+n^2

$a = 16$, $b = 4$, $f(n) = n^2$

$\log_b a = \log_4 16 = 2$

Case 2 applies : $f(n) = n^2 = \Theta(n^2)$.

$\therefore T(n) = \Theta(n^2)$.

7. (20 points) Use Backward Substitution (10 points) and then Forward Substitution (10 points) to solve the recurrence relations T(n)=2T(n-1)+1;T(0)=1. In each case, do the following: (1) Show at least three expansions so that the emerging pattern is evident. (2) Then write out T(n) fully and simplify using equation (A.5) on Text p.1147. (3) Verify your solution by substituting it in the LHS and RHS of the recurrence relation and demonstrating that LHS=RHS. (4) Finally state the complexity order of T(n). <u>You must show your work for parts (1)-(3) to receive credit.</u>

Using Backward Substitution :

$T(n) = 2T(n-1) + 1$ ——— (1)

$T(n-1) = 2T(n-2) + 1$ ——— (2)

$T(n-2) = 2T(n-3) + 1$ ——— (3)

$\Rightarrow T(n-1) = 2[2T(n-3) + 1] + 1$ ——— substituting (3) in (2)

$= 2^2 \times T(n-3) + 2 + 1$ ——— (4)

$\Rightarrow T(n) = 2[2^2 \times T(n-3) + 2^1 + 1] + 1$ ——— substituting (4) in (1)

$= 2^3 \times T(n-3) + 2^2 + 2^1 + 1$.

So we can continue and T(n) can be represented as :

$T(n) = 2^n \times T(n-n) + 2^{n-1} + \cdots + 2^1 + 2^0$

$= \sum_{i=0}^{n} 2^i = \frac{2^{n+1} - 1}{2-1} = 2^{n+1} - 1 \Rightarrow \boxed{T(n) = 2^{n+1} - 1}$

Using forward Substitution :

$T(0) = 1$; $T(1) = 2T(0) + 1 = 2 + 1$.

$T(2) = 2T(1) + 1 = 2[2+1] + 1 = 2^2 + 2 + 1$.

So if we continue, in this manner, T(n) can be represented as :

$T(n) = 2^n + 2^{n-1} + \cdots 2^1 + 2^0 = \sum_{i=0}^{n} 2^i = \frac{2^{n+1} - 1}{2-1}$

$\Rightarrow \boxed{T(n) = 2^{n+1} - 1}$

~~8. (4 points). Problem 3.1-3 (p.53).~~

Proof.

$T(n) = 2T(n-1)+1.$ —— ①

~~LHS~~ : Solution reached from substitution method :

$T(n) = 2^{n+1} - 1$ —— ②

RHS : $2T(n-1)+1.$

Substituting ② in RHS we get :

$2\left[2^{(n-1)+1} - 1\right] + 1.$

$= 2\left[2^{n} - 1\right] + 1.$

$= 2^{n+1} - 2 + 1.$

$= 2^{n+1} - 1$ (Proved).

(8)

3.1-3 "The running time of algorithm A is at least $O(n^2)$" is meaningless

Sol^n.

This statement says that the $T(n)$ is asymptotically greater than or equal to, all functions that are $O(n^2$
It says nothing about the upper bounds or the lower bounds for the ~~algorithm~~ running time of the algorithm. So this statement is meaningless.