Due Date: March. 1st, 2019 (F), BEFORE the lecture starts.

This assignment covers textbook Chapter 4 & Chapter1~3.

In problem 1 to 3, solve the following recurrence with three different methods that we learned in class.

$$T(n) = \begin{cases} \Theta(1) & n \le k \\ T\left(\frac{n}{3}\right) + T\left(\frac{n}{3}\right) + 2^{(\log_3 n)} & n > k \end{cases}$$

where $n = 3^j$ for a positive integer j, and k is a small positive integer. That is, find a function g(n) such that $T(n) \in \Theta(g(n))$. The $\Theta(1)$ terminating condition is intended to represent some small constant.

1. Use the **Master Theorem** to solve this recurrence. (10 points)
2. Use the **recursion tree** to solve the recurrence. (15 points)
3. Use **substitution** method to prove the correctness of your answer to problem 1 and 2 above. Be sure to justify both the O and Ω parts of the Θ notation. (15 points)

4. **Resolve Recurrence** (15 points)
An algorithm processes an array of size *n* by operating on its first one-third, its second one-third, its third one-third, and then operating on its first one-third again, recursively. It then combines the solutions in $2^{\lg n}$ time.

Derive a recurrence for the running time of above algorithm. You may assume that $n=3^k$ for some positive integer k. Use an appropriate method (just pick one method) to solve the recurrence by finding a tight upper and lower bound solution for the recurrence. You must show the procedure of calculation.

5. **Recurrence** (15 points)
For each of the following recurrences, give an expression for the runtime T(n) if the recurrence can be solved with the Master Theorem. Otherwise, explain why the Master Theorem does not apply. Justify your answer.

(1) $T(n) = 4T\left(\frac{n}{4}\right) + n$
(2) $T(n) = 3T\left(\frac{n}{2}\right) + \sqrt{10}n^2$
(3) $T(n) = T(n-1) + 10n$
(4) $T(n) = 3T\left(\frac{2n}{3}\right) + n$
(5) $T(n) = 2^n T\left(\frac{n}{3}\right) + n^2$

6. **Design an Algorithm** (30 points)
Given a sorted array A that stores *n* distinct integers. Design ***an efficient* divide-and-conquer** algorithm to find out whether there is an index i for which A[i] == i.

(1) (10 points) Pseudocode *(please use the textbook conventions)*

(2) (10 points) Analysis: Provide a tight upper bound on the worst-case asymptotic running time of your pseudocode (provide the recurrence and resolve it). Justify your answer (i.e., list the cost for executing each line of code and how you calculate the total running time)

(3) (10 points) Prove your answer using the **substitution method**

Algorithms -- COMP.4040 Honor Statement
(Courtesy of Prof. Tom Costello and Karen Daniels with modifications)

Must be attached to each submission, otherwise, your homework will not be graded.

Academic achievement is ordinarily evaluated on the basis of work that a student produces independently. Infringement of this Code of Honor entails penalties ranging from reprimand to suspension, dismissal or expulsion from the University.

Your name on any exercise is regarded as assurance and certification that what you are submitting for that exercise is the result of your own thoughts and study. Where collaboration is authorized, you should state very clearly which parts of any assignment were performed with collaboration and name your collaborators.

In writing examinations and quizzes, you are expected and required to respond entirely on the basis of your own memory and capacity, without any assistance whatsoever except such as what is specifically authorized by the instructor.

I certify that the work submitted with this assignment is mine and was generated in a manner consistent with this document, the course academic policy on the course website on Blackboard, and the UMass Lowell academic code.

Date: 02/25/2019

Name (please print): DANGNHI NGO

Signature: [illegible]

100/100

$$T(n) = \begin{cases} \Theta(1) & n \leqslant k \\ T\left(\frac{n}{3}\right) + T\left(\frac{n}{3}\right) + 2^{(\log_3 n)} & n > k \end{cases}$$

10 1/ Use the Master Theorem

$$T(n) = T\left(\tfrac{n}{3}\right) + T\left(\tfrac{n}{3}\right) + 2^{(\log_3 n)} \qquad (n > k)$$

$$= 2T\left(\tfrac{n}{3}\right) + n^{(\log_3 2)}$$

$a = 2$, $b = 3$ then $\log_b a = \log_3 2$

Compare $n^{(\log_3 2)}$ vs. $n^{(\log_3 2)}$ $\quad$ $(f(n) = n^{(\log_3 2)})$

$f(n)$ is polynomially equal to $n^{\log_b a}$ (Case 2)

Therefore, $\boxed{T(n) = \Theta(n^{\log_3 2} \cdot \lg n)}$

20 2/ Use the Recursion Tree

$$T(n) = T\left(\tfrac{n}{3}\right) + T\left(\tfrac{n}{3}\right) + 2^{(\log_3 n)} \qquad (n > k)$$

$$= 2T\left(\tfrac{n}{3}\right) + n^{(\log_3 2)}$$

Given $n = 3^i$, $\quad T\left(\frac{n}{3}\right) = 2.T\left(\frac{n}{9}\right) + \left(\frac{n}{3}\right)^{(\log_3 2)}$

$$= 2T\left(\tfrac{n}{9}\right) + \frac{n^{(\log_3 2)}}{2}$$

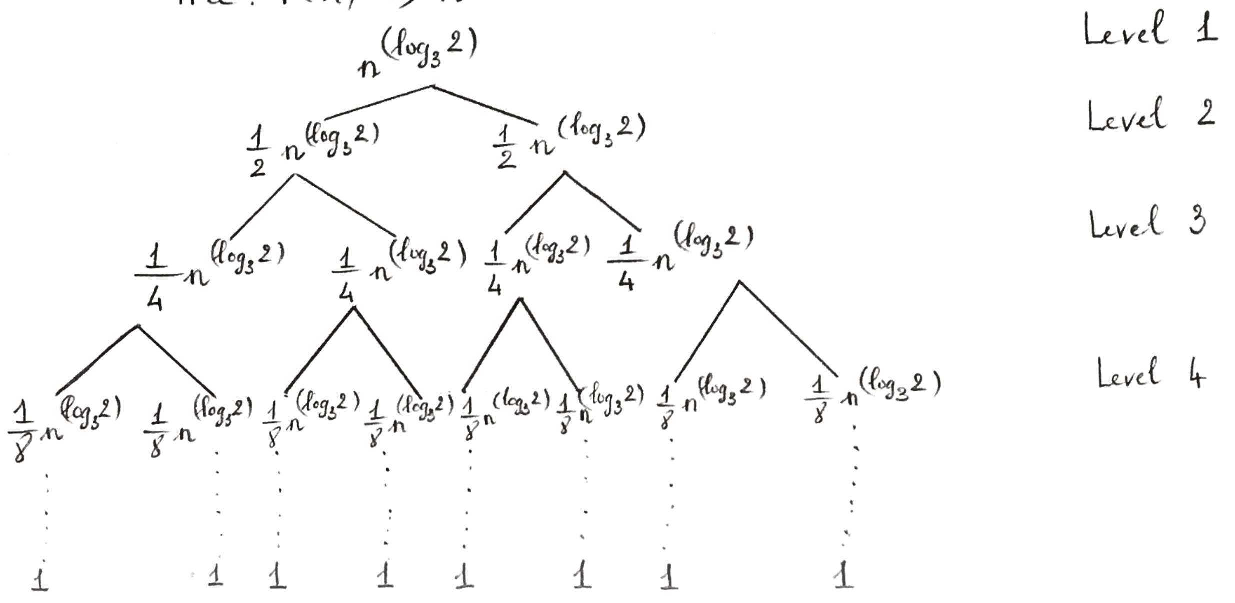
$$T\left(\tfrac{n}{9}\right) = 2T\left(\tfrac{n}{27}\right) + \left(\tfrac{n}{9}\right)^{(\log_3 2)}$$

$$= 2T\left(\tfrac{n}{27}\right) + \frac{n^{(\log_3 2)}}{4}$$

$$T\left(\tfrac{n}{27}\right) = 2T\left(\tfrac{n}{81}\right) + \left(\tfrac{n}{27}\right)^{(\log_3 2)}$$

$$= 2T\left(\tfrac{n}{81}\right) + \frac{n^{(\log_3 2)}}{8}$$

Tree : $T(n) \Rightarrow n^{(\log_3 2)}$



$n^{(\log_3 2)}$
$\frac{1}{2} n^{(\log_3 2)}$
$\frac{1}{2} n^{(\log_3 2)}$
$\frac{1}{4} n^{(\log_3 2)}$
$\frac{1}{4} n^{(\log_3 2)}$
$\frac{1}{4} n^{(\log_3 2)}$
$\frac{1}{4} n^{(\log_3 2)}$
$\frac{1}{8} n^{(\log_3 2)}$
$\frac{1}{8} n^{(\log_3 2)}$
$\frac{1}{8} n^{(\log_3 2)}$
$\frac{1}{8} n^{(\log_3 2)}$
$\frac{1}{8} n^{(\log_3 2)}$
$\frac{1}{8} n^{(\log_3 2)}$
$\frac{1}{8} n^{(\log_3 2)}$
$\frac{1}{8} n^{(\log_3 2)}$
1 1 1 1 1 1 1 1
Level 1
Level 2
Level 3
Level 4

Level	T(n)	Cost	Factor	
1	$T(n)$	$n^{(\log_3 2)}$	1	$n^{(\log_3 2)}$
2	$T(\frac{n}{3})$	$\frac{1}{2} n^{(\log_3 2)}$	2	$n^{(\log_3 2)}$
3	$T(\frac{n}{9})$	$\frac{1}{4} n^{(\log_3 2)}$	4	$n^{(\log_3 2)}$
⋮	⋮	⋮	⋮	⋮
i	$T\left(\frac{n}{3^{i-1}}\right)$	$\frac{1}{2^{i+1}} n^{(\log_3 2)}$	2^{i-1}	$n^{(\log_3 2)}$

$$\frac{n}{2^{i-1}} = 1 \Rightarrow 2^{i-1} = n$$
$$\Rightarrow i-1 = \lg n$$
$$\Rightarrow i = \lg n + 1$$

$$T(n) = (\# \text{ of level} \times n^{(\log_3 2)})$$
$$= (\lg n + 1) \times n^{(\log_3 2)}$$
$$= \lg n . n^{(\log_3 2)} + n^{(\log_3 2)}$$

$$\boxed{T(n) = \theta(\lg n . n^{(\log_3 2)})}$$

20 3/ Use Substitution Method

$$T(n) = T(\tfrac{n}{3}) + T(\tfrac{n}{3}) + 2^{(\log_3 n)} \qquad (n > k)$$
$$= 2T(\tfrac{n}{3}) + n^{(\log_3 2)}$$

* Upper bound: $T(n) \leqslant 2T(n/3) + c.n^{(\log_3 2)}$ (c : positive constant)

Guess: $T(n) = O(n^{(\log_3 2)} . \lg n)$

$$\leqslant d.(n^{(\log_3 2)} . \lg n) \qquad (d : \text{positive constant})$$

$$T(\tfrac{n}{3}) \leqslant d.(\tfrac{n}{3})^{(\log_3 2)} . \lg(\tfrac{n}{3})$$
$$T(\tfrac{n}{3}) \leqslant d.\frac{n^{(\log_3 2)}}{2} . \lg(\tfrac{n}{3})$$

Substitution: $T(n) \leqslant \cancel{2}(d.\frac{n^{(\log_3 2)}}{\cancel{2}} . \lg(\tfrac{n}{3})) + c.n^{(\log_3 2)}$

$$= dn^{(\log_3 2)} . (\lg n - \lg 3) + c.n^{(\log_3 2)}$$
$$= dn^{(\log_3 2)} \lg n - dn^{(\log_3 2)} \lg 3 + c.n^{(\log_3 2)}$$
$$\leqslant dn^{(\log_3 2)} \lg n \quad (\text{if } (-dn^{(\log_3 2)} \lg 3 + c.n^{(\log_3 2)}) \leqslant 0)$$
$$\Rightarrow d\cancel{n^{(\log_3 2)}} \lg 3 \geqslant c.\cancel{n^{(\log_3 2)}}$$
$$\Rightarrow d \geqslant c/\lg 3$$

Therefore, $T(n) = O(n^{(\log_3 2)} . \lg n)$ (1)

* Lower bound : $T(n) \geq 2T(\frac{n}{3}) + c.n^{(\log_3 2)}$ (c: positive constant)

Guess : $T(n) = \Omega(n^{(\log_3 2)}.\lg n)$

$\geq d(n^{\log_3 2}.\lg n)$ (d: positive constant)

$T(\frac{n}{3}) \geq d(\frac{n}{3})^{(\log_3 2)}.\lg(\frac{n}{3})$

$T(\frac{n}{3}) \geq d\,\frac{n^{(\log_3 2)}}{2}.\lg(\frac{n}{3})$

Substitution : $T(n) \geq 2(d\,\frac{n^{(\log_3 2)}}{2}.\lg(\frac{n}{3})) + c.n^{(\log_3 2)}$

$= d.n^{(\log_3 2)}.(\lg n - \lg 3) + c.n^{(\log_3 2)}$

$= d.n^{(\log_3 2)}\lg n - dn^{(\log_3 2)}\lg 3 + c.n^{(\log_3 2)}$

$\geq dn^{(\log_3 2)}\lg n$ (if $(-dn^{(\log_3 2)}\lg 3 + c.n^{(\log_3 2)} \geq 0)$)

$\Rightarrow \quad c\cancel{n^{(\log_3 2)}} \geq d\cancel{n^{(\log_3 2)}}\lg 3$

$\Rightarrow \quad c \geq d.\lg 3$

$\Rightarrow \quad d \leq c/\lg 3$

Therefore, $T(n) = \Omega(n^{(\log_3 2)}.\lg n)$ (2)

From (1) and (2),

$\boxed{T(n) = \Theta(n^{(\log_3 2)}.\lg n)}$

4/ Resolve Recurrence

10 An algorithm processes an array of size n by operating on its first one-third, its second one-third, its third one-third, and then operating on its first one-third again, recursively. It then combines the solution is $2^{\lg n}$ time

This means that:

$$T(n) = 3T\left(\frac{n}{3}\right) + T\left(\frac{n}{3}\right) + 2^{\lg n}$$

$$T(n) = 4T\left(\frac{n}{3}\right) + n$$

$a = 4,\ b = 3 \Rightarrow \log_b a = \log_3 4$

Compare $n^{(\log_b a)} = n^{(\log_3 4)}$ vs. $f(n) = n$

$\Rightarrow$ $f(n)$ is polynomially smaller than $n^{(\log_b a)}$ $(n < n^{(\log_3 4)})$

$\Rightarrow$ Case 1 in Master Method

$T(n) = \theta(n^{(\log_3 4)})$: Recurrence for the running time of algorithm

* Use Substitution Method to solve the recurrence

– Upper bound : $T(n) \leqslant 4T\left(\frac{n}{3}\right) + cn$ (c: positive constant)

Guess : $T(n) \leqslant d(n^{(\log_3 4)}) - d'n$ (d, d' : positive constant)

$$T\left(\frac{n}{3}\right) \leqslant d\left(\frac{n}{3}\right)^{(\log_3 4)} - d'\left(\frac{n}{3}\right)$$

Substitution: $T(n) \leqslant 4\left(d.\left(\frac{n}{3}\right)^{(\log_3 4)} - d'\left(\frac{n}{3}\right)\right) + cn$

$$= 4\left(d\frac{n^{(\log_3 4)}}{4} - d'\left(\frac{n}{3}\right)\right) + cn$$

$$= d.n^{(\log_3 4)} - \frac{4}{3}d'n + cn$$

$$\leqslant d.n^{(\log_3 4)} \quad \left(\text{if } \left(-\frac{4}{3}d'n + cn\right) \leqslant 0\right)$$

$$\Rightarrow cn \leqslant \frac{4}{3}d'n$$

$$\Rightarrow d' \geqslant \frac{3}{4}c$$

Therefore, $T(n) = O(n^{(\log_3 4)})$ (1)

- Lower bound: $T(n) \geqslant 4T(\frac{n}{3}) + cn$ (c: positive constant)

Guess: $T(n) \geqslant d(n^{(\log_3 4)})$ (d: positive constant)

$T(\frac{n}{3}) \geqslant d(\frac{n}{3})^{(\log_3 4)}$

Substitution: $T(n) \geqslant 4d(\frac{n}{3})^{(\log_3 4)} + cn$

$= \cancel{4} . d . \frac{n^{(\log_3 4)}}{\cancel{4}} + cn$

$= d . n^{(\log_3 4)} + cn$

$\geqslant d.n^{(\log_3 4)}$ (if $cn \geqslant 0$)

$\Rightarrow$ always possible

Therefore: $T(n) = \Omega(n^{(\log_3 4)})$ (2)

From (1) and (2):

$$\boxed{T(n) = \theta(n^{(\log_3 4)})}$$

5/ Recurrence

(1) $T(n) = 4T(\frac{n}{4}) + n$

10

$a = 4, b = 4 \Rightarrow \log_b a = \log_4 4 = 1$

$n^{(\log_b a)} = n^1 = n$

$f(n) = n$

Compare $n^{(\log_b a)} = n$ vs. $f(n) = n$

$\Rightarrow f(n)$ is polynomially equal to $n^{(\log_b a)}$

$\Rightarrow$ Case 2 in Master Method

$T(n) = \Theta(n.\lg n)$ ✓

(2) $T(n) = 3T(\frac{n}{2}) + \sqrt{10}.n^2$

$a = 3, b = 2 \Rightarrow \log_b a = \log_2 3 \approx 1.58$

Compare $n^{(\log_b a)} = n^{\log_2 3}$ vs. $f(n) = \sqrt{10}\,n^2$

$\Rightarrow f(n)$ is polynomially greater than $n^{(\log_b a)}$ $(\sqrt{10}n^2 > n^{\log_2 3})$

$\Rightarrow$ Case 3 in Master Method ✓

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

(3) $T(n) = T(n-1) + 10n$

$a = 1, b = 1$ It is not in the form $T(n) = aT(\frac{n}{b}) + f(n)$ ✓

$\Rightarrow$ Master Method cannot be applied, because b should be greater than 1

(4) $T(n) = 3T(\frac{2n}{3}) + n$

$a = 3, b = 3/2 \Rightarrow \log_b a = \log_{(3/2)} 3$

Compare $n^{(\log_b a)} = n^{(\log_{(3/2)} 3)}$ vs. $f(n) = n$ ✓

$\Rightarrow f(n)$ is polynomially smaller than $n^{(\log_b a)}$ $(n < n^{(\log_{(3/2)} 3)})$

$\Rightarrow$ Case 1 in Master Method

$T(n) = \Theta(n^{(\log_{(3/2)} 3)})$

(5) $T(n) = 2^n T(\frac{n}{3}) + n^2$

$a = 2^n, b = 3$

$\Rightarrow$ Master Method cannot be applied, because a should be a constant ✓

6/ Design an Algorithm

30 Design an efficient divide - and - conquer algorithm to find out whether there is an index i for which $A[i] == i$

(1) Pseudocode

```
    Algorithm - function ( $A[i...n]$, offset):
1.  if ( $[\lceil n/2 \rceil]$ equals to (offset + $\lceil n/2 \rceil$)), then return true
2.  if $|A| \leqslant 1$, then return false
3.  if ( $A[\lceil n/2 \rceil] <$ (offset + $\lceil n/2 \rceil$)),
4.      return Algorithm - function ( $A[(c\lceil n/2 \rceil + 1).....n]$, offset + $\lceil n/2 \rceil$)
5.  else
6.      return Algorithm - function ( $A[1........(\lceil n/2 \rceil - 1)]$, offset )
```

(2) Analysis: Provide a tight upper bound on the worst - case asymptotic running time of your pseudocode

Lower bound: $\Theta(1)$

Upper bound: worst - case

Total cost: $T(n) = C_1 . 1 + C_2 . 1 + T(\frac{n}{2})$

$$= T(\tfrac{n}{2}) + (C_1 + C_2)$$

$$T(n) = T(\tfrac{n}{2}) + O(1)$$

Master Theorem:

$a = 1, \ b = 2 \Rightarrow \log_b a = \log_2 1 = 0$

Compare $n^{(\log_b a)} = n^0 = 1$ vs. $f(n) = c.n^0 = O(1)$

$\Rightarrow$ $f(n)$ is polynomially equal to $n^{(\log_b a)}$

$\Rightarrow$ Case 2 in Master Method

$$\boxed{T(n) = \Theta(\lg n)}$$

(3) Substitution Method

$$T(n) = T(\frac{n}{2}) + O(1)$$

* Upper bound: $T(n) \leqslant T(\frac{n}{2}) + c.n^0$ (c: positive constant)

Guess: $T(n) = O(\lg n)$

$\leqslant d(\lg n)$ (d: positive constant)

$T(\frac{n}{2}) \leqslant d(\lg(\frac{n}{2}))$

Substitution: $T(n) \leqslant d.\lg(\frac{n}{2}) + c$

$= d(\lg n - \lg 2) + c$

$= d\lg n - d\lg 2 + c$

$= d\lg n - d + c$

$\leqslant d\lg n$ (if $(-d+c) \leqslant 0$)

$\Rightarrow c \leqslant d$

Therefore, $T(n) = O(\lg n)$ (1)

* Lower bound: $T(n) \geqslant T(\frac{n}{2}) + c.n^0$ (c: positive constant)

Guess: $T(n) = \Omega(\lg n)$

$\geqslant d.\lg n$ (d: positive constant)

$T(\frac{n}{2}) \geqslant d.\lg(\frac{n}{2})$

Substitution: $T(n) \geqslant d.\lg(\frac{n}{2}) + c$

$= d(\lg n - \lg 2) + c$

$= d\lg n - d\lg 2 + c$

$= d\lg n - d + c$

$\geqslant d\lg n$ (if $(-d+c) \geqslant 0$)

$\Rightarrow c \geqslant d$

Therefore, $T(n) = \Omega(\lg n)$ (2)

From (1) and (2),

$$\boxed{T(n) = \Theta(\lg n)}$$