Name: (Print) PHONG VO

For problem 1 & 2, Derive a recurrence for the running time of each algorithm below and then solve the recurrence with one of the three methods we have learned. The solution should be a tight upper and lower bound solution. You must show the work of how to solve the recurrence. You do NOT need to write the algorithm.

(15) 1. **Recurrence** (15 points)

An algorithm solves a problem of size *n* by recursively processing ¾ of *n* elements, and dividing takes Θ(n) time.

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

$a = 1$ $b = 4/3$ $\log_b a = \log_{4/3} 1 = 0$

$$n^{\log_b 2} = n^{\log_{4/3} 1} = n^0 = 1$$

Compare: cn vs. $n^0 (= 1)$

cn is larger than n^0

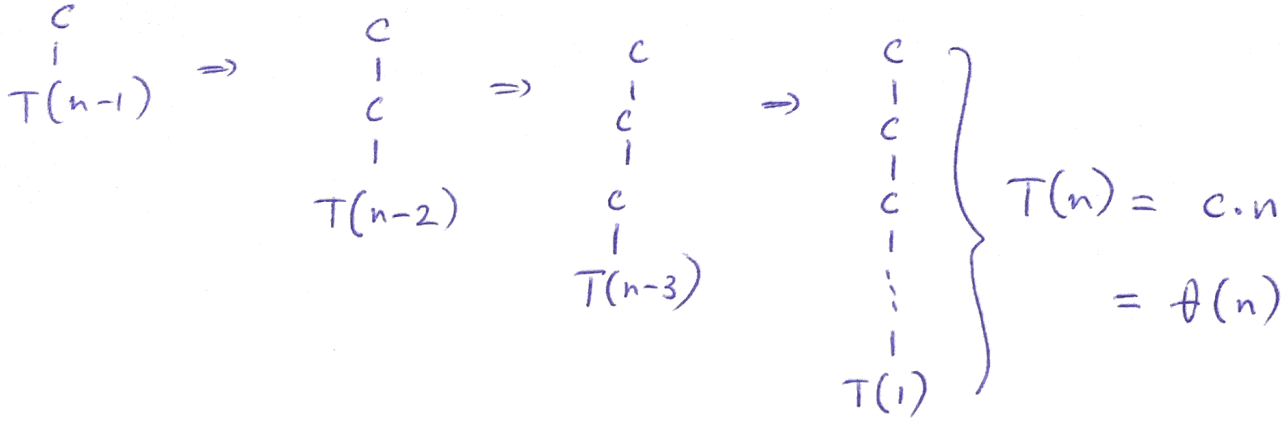
Thus, case 3 of Master Method

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

(2) Solution : $T(n) = T(n-1) + \theta(1)$

$= T(n-1) + c$ (c : pos. const)

$T(n) = T(n-1) + \theta(n)$



13

2. **Recurrence** (15 points)

An algorithm solves a problem of size n by recursively solving one sub-problems of size (n – 1), and then combining the solutions in constant time.

$$T(n) = T(n-1) + T(1) + \theta(c')$$

$a = 1$

$b = 1 \Rightarrow$ violates the Master Method

- can not apply M.M.
- Can not apply Recursion Tree — why ?
- Substitution Method:

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

$T(n) \leq cn$ (c: pos. const)

Assume $T(n-1) \leq c(n-1) = cn - c$

Substitute: $cn - c + T(1) + \theta(c')$

$= cn - c$

$\leq cn$ (if $-c \leq 0 \Rightarrow c \geq 0$ is possibly always.)

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

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

$T(n) \geq cn$ (c: pos. const)

Assume $T(n-1) \geq c(n-1)$

$= cn - c$

Substitute: $cn - c + T(1) + \theta(c')$

$= cn - c \geq cn$ if $-c \geq 0 \Rightarrow c < 0$

$\Rightarrow$ impossible!

Thus: $T(n) = O(n)$

$\theta(n)$ -2

15 3. **Substitution method** (15 points)

Use the substitution method to prove that recurrence T(n) = 3T(n/3) + cn is in the tight bound of *nlgn* (where c is a positive constant). Please prove for both upper bound and lower bound.

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

Substitution Method:

Upper bound

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

$T(n) \leq d\,n \lg n$ (d: pos. const.)

Assume: $T\left(\frac{n}{3}\right) = d \cdot \frac{n}{3} \cdot \lg\left(\frac{n}{3}\right) = d\frac{n}{3}(\lg n - \lg 3)$

$= \frac{d}{3} n \lg n - \frac{d}{3} n \lg 3$

Substitute: $3\left(\frac{d}{3} n \lg n - \frac{d}{3} n \lg 3\right) + cn$

$= d n \lg n - d n \lg 3 + cn$

Compare: $\leq d n \lg n$ if $(-d n \lg 3 + cn) \leq 0$

$\Rightarrow \quad cn \leq d n \lg 3$

$\Rightarrow \quad c \leq d \lg 3$ ✓

$\therefore T(n) = O(n \lg n)$ ✓

Guess: $T(n) = \Omega(n \lg n)$ ✓

$T(n) \geq dn \lg n$ (d: pos. const.)

$T\left(\frac{n}{3}\right) = d\frac{n}{3}\lg\left(\frac{n}{3}\right)$

$= \frac{dn}{3}(\lg n - \lg 3) = \frac{d}{3} n \lg n - \frac{d}{3} n \lg 3$

Substitute: $3\left(\frac{d}{3} n \lg n - \frac{d}{3} n \lg 3\right) + cn$

$= d \cdot n \lg n - dn \lg 3 + cn$

Compare $\geq dn \lg n$ if $(-dn \lg 3 + cn \geq 0)$

$\Rightarrow -d \lg 3 + c \geq 0$

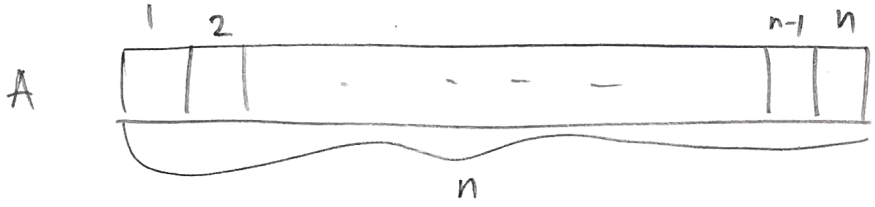
$\Rightarrow c \geq d \lg 3$

$\therefore T(n) = \Omega(n \lg n)$

$\therefore T(n) = \Theta(n \lg n)$ ✓

4. **Design and analysis of an Algorithm (20 points)**
You are given an array *A*, which stores *n* non-negative integers. Design a **divide-and-conquer** algorithm that accepts *A* and *n* as inputs only and returns the **index of the maximum value** in the array *A*. You may use a helper function if needed.

(1) (12 points) **Pseudocode**: *(please use textbook conventions)*



(1)

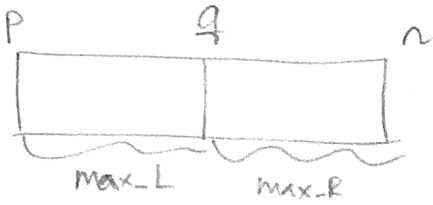
	total cost if or exec	Cost
Function index_of_Max (n) {		
maxInd = 1	1	c_1
For i = 1 to n	n+1	c_2
if A[maxInd] ≤ A[maxInd + i]	n	c_3
maxInd = maxInd + i	O(n)	c_4
return maxInd	1	c_5
}		

Can't solve the problem

$$T(n) = c_1 \cdot 1 + c_2(n+1) + c_3 n + c_4 n + c_5 \cdot 1$$
$$= n(c_2 + c_3 + c_4) + c_1 + c_2 + c_5$$
$$= cn + \theta(1) = \theta(n)$$

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

④ Solution



combine
max(max_L, max_R)

DCMax(A, n)

return DCMax_helper(A, 1, n)

DCMax_helper(A, p, r)

C_1	1	if p == r
C_2	2	return p
C_3	3	$q = \lfloor \frac{p+r}{2} \rfloor$
C_4	4	max_L_index = DCMax_helper(A, p, q)
C_5	5	max_R_index (A, q+1, r)
C_6	6	if A[max_L_index] > A[max_R_index]
C_7	7	return max_L_index
C_8	8	else return max_R_index

$$T(n) = C_1 + C_3 + T(\tfrac{n}{2}) + T(\tfrac{n}{2}) + C_6 + C_7$$

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

(2) (8 points) **Analysis:** Derive a recurrence for the running time of your algorithm (**You don't need to solve the recurrence**). Justify your answer by listing the cost for executing each line of code and the number of executions for each line. You may mark the cost with your algorithm on the previous page or specify the cost using the line numbers on this page.

8

The Answer is on the previous page.