CSCI 570 Spring 2025

Homework 5

Name: Abhishek Soundalgekar
USC ID: 2089011000

Question 1
Answer 1

a $$T(n) = 9T\left(\frac{n}{5}\right) + n\log n$$

Using Master's theorem for the divide and conquer recurrences, we get:

$a = 9$, $b = 5$, $[f(n) = n\log n]$ → equation 1

$$n^{\log_b a} = n^{\log_5 9} = n^{1.3652} \rightarrow \text{equation 2}$$

Comparing $f(n)$ & $n^{\log_b a}$ (equation 1 & 2)

$f(n) = n\log n$ is $O\left(n^{1.3652-\epsilon}\right)$ for some $\epsilon > 0$ we can observe that $n^{\log_b a}$ grows faster than $f(n)$. Hence this falls under the 'case 1' of Master Theorem $\left(f(n) < n^{\log_b a}\right)$

$$T(n) = \Theta\left(n^{\log_b a}\right) = \Theta\left(n^{\log_5 9}\right) = \Theta\left(n^{1.3652}\right)$$

b] $T(n) = \sqrt{2025}\, T\left(\frac{n}{3}\right) + n^{\sqrt{2025}}$

$T(n) = 45\, T\left(\frac{n}{3}\right) + n^{45}$

Using the Master Theorem we get

$a = 45, \quad b = 3$

$f(n) = n^{45} \longrightarrow$ eq/1

$n^{\log_b a} = n^{\log_3 45} = n^{3.46497} \rightarrow$ eq/2

from eq/1 & eq/2 we get

$f(n) = n^{45}$ grows exsymptotically faster than $n^{\log_b a} = n^{3.46497}$.

To check if f T(n) fall under 'case 3' of Master Theorem, we need to check if it satisfies the following constraint

a. $f\left(\frac{n}{b}\right) \leq c \cdot f(n)$ where $c < 1$

$$45 \cdot \left(\frac{n}{3}\right)^{\sqrt{2025}} \leq c \cdot n^{\sqrt{2025}}$$

$$45 \cdot \frac{n^{45}}{3^{45}} \leq c \cdot n^{45}$$

$$\frac{45}{3^{45}} \leq c$$

For the value of c which we got, the given equation $T(n)$ falls under case 3 of Master Theorem, since it satisfies the required constraint

$$T(n) = \Theta(f(n)) = \Theta(n^{45}).$$

c] $T(n) = 9T\left(\frac{n}{3}\right) + n^2 \log n$

Using the Master Theorem we get

$a = 9, \quad b = 3, \quad f(n) = n^2 \log n$

$$n^{\log_b^a} = n^{\log_3^9} = n^{\log_3^{3^2}} = n^2$$

~~$f(n)$ grows asymptotically faster than~~
~~$n^{\log_b^a} = n^2$~~

We observe this as case 2 of master theorem.

~~The book~~ if

According to case 2 of the theorem we get

$$f(n) = \Theta\left(n^{\log_b^a} \cdot \log^k n\right) \text{ where } k \geq 0$$

Then $T(n) = \Theta\left(n^{\log_b^a} \log^{k+1} n\right)$

Substituting the values we get

$$T(n) = \Theta(n^2 \log^{1+1} n) = \Theta(n^2 \log^2 n)$$

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

d.] $$T(n) = 10T\left(\frac{n}{2}\right) + 2^n$$

Using the Master theorem we get:

$a = 10, \quad b = 2$

$f(n) = 2^n$

$$\log_b^a = \log_2 10 = 3.3219$$

$$n^{\log_b^a} = n^{3.3219}$$

For larger value of n: $f(n) > n^{\log_b^a}$

Therefore, $f(n)$ grows faster than $n^{\log_b^a}$

We require to check if $T(n)$ satisfies the constraint

$$a\, f\left(\frac{n}{b}\right) \leq c \cdot f(n)$$

$$10 \cdot 2^{(n/2)} \leq C \cdot 2^n$$

$$10 \cdot \frac{2^{n/2}}{2^n} \leq C$$

$$10 \cdot 2^{n/2 - n} \leq C$$

$$10 \cdot 2^{-n/2} \leq C$$

for large value of n, we can get $0 < C < 1$ such that it satisfies the constraint. Hence this falls under case 3.

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

e.] $T(n) = 3T\left(\frac{n}{4}\right) + n\log^2 n$

Using the Master Theorem we get

$a = 3$, $b = 4$

$f(n) = n\log^2 n$

$n^{\log_b^a} = n^{\log_4^3} = n^{0.7925}$

$f(n) = n\log^2 n$ is $\Omega\left(n^{0.7925+\epsilon}\right)$ for some $\epsilon > 0$

$f(n)$ grows faster than $n^{\log_b^a}$

We require to check if $T(n)$ satisfies the constraint

$$a\, f(n/b) \leq c \cdot f(n)$$

$$3 \cdot \frac{n}{4} \cdot \left(\log \frac{n}{4}\right)^2 \leq c \cdot n\log^2 n$$

$$\frac{3}{4} \cdot (\log n - \log 4)^2 \leq c \cdot (\log n)^2$$

$$\frac{3}{4} \cdot (\log n - 2)^2 \leq c \cdot (\log n)^2$$

For large value of n the value of $(\log n - 2)$ will be close to $\log n$ itself

$$\frac{3}{4}(\log n)^2 \leq c \cdot (\log n)^2$$

$$\frac{3}{4} \leq c$$

There will be some constant c satisfying the condition $c < 1$ and all sufficiently large n.

Hence this falls under case 3.

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

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

Question 2
Answer 2

a] Designing a divide & conquer algorithm to find a given value k in M.

Base case: If the matrix is 1×1, directly check if the element equals to k.

Divide step: Split the n×n matrix M into four quadrants of size $n/2 \times n/2$

Conquer step:
1. Find the pivot element (middle element)
$M_P = M[n/2][n/2]$.
2. If $k = M_P$, return position
3. If $k < M_P$, recursively search the top-left, top-right and bottom-left quadrants (eliminate the bottom-right quadrant).
4. If $k > M_P$, recursively search the top-right, bottom-left, and bottom-right quadrants (eliminate the top-left quadrant).

Combine step: Return position (true) if any recursive call finds k; otherwise return false

b] Recurrence Relation

$$T(n) = 3T\left(\frac{n}{2}\right) + O(1)$$

→ At each step, the matrix is divided into four $n/2 \times n/2$ quadrants.

→ After comparing k with the middle element, three quadrants are recursively searched.

→ The $O(1)$ term is for the constant-time comparison with the middle element.

c] Solving the recurrence using the Master Theorem:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \qquad a \geq 1,\ b \geq 1$$

$$T(n) = 3T\left(\frac{n}{2}\right) + O(1)$$

$a = 3,\ b = 2$

$f(n) = O(1)$

$$n^{\log_b a} = n^{\log_2 3} = n^{1.585}$$

We can observe that $f(n)$ is asymptotically smaller than $n^{\log_b^a}$, thus we can apply the case 1 of the Master's theorem.

Case 1 of the Master's theorem states that

if $f(n) = O\left(n^{\log_b^a - \epsilon}\right)$ for some $\epsilon > 0$,

then $T(n) = \Theta\left(n^{\log_b^a}\right)$

∴ The solution for the recursive relation is

$$T(n) = \Theta\left(n^{1.585}\right)$$

Question 3
Answer 3

The bank wants to determine if there exists a majority user (an account with $> n/2$ equivalent cards) in a collection of n bank cards.
The only operation allowed is using an equivalence tester to check if two cards belong to the same account. The goal is to solve this with $O(n \log n)$ invocations of the tester.

a] Divide & Conquer Algorithm:

1. Base Case: If $n = 1$, the single card is trivially the majority.
2. Divide: Split the n cards into two halves $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$.
3. Conquer:
   3.1 Recursively check each half for a majority user.
   3.2 For each half, if a majority exists, return a representative card of that majority.
4. Combine:
   4.1 If neither half has a majority, the whole set has no majority.

4.2 If both halves have the same majority, return this majority.

4.3 If the halves have different majorities or only one has a majority:

4.3.1 For each candidate (up to two), count its global frequency by comparing it with all n cards using the equivalence tester.

4.3.2 If any candidate has $> n/2$ equivalents, it is the majority; otherwise, no majority exists.

b.] Algorithm's Recurrence Relation

$$T(n) = 2T\left(\left\lceil \frac{n}{2} \right\rceil\right) + O(n)$$

Divide → the splitting into two subproblems of size $\lceil n/2 \rceil$, leads to $2T(\lceil n/2 \rceil)$

Combine → Checking up to two candidates against all n cards requires $O(n)$ equivalence tests.

c] Solving the recurrence relation using the Master Theorem

$$T(n) = 2T\left(\left\lceil \frac{n}{2} \right\rceil\right) + O(n).$$

$a = 2$ (no. of subproblems)
$b = 2$ (division factor)
$f(n) = O(n)$ (cost of the non-recursive work).

$$n^{\log_b^a} = n^{\log_2^2} = n.$$

We observe $f(n) = O(n^{\log_b^a})$ which falls into case 2 of the Master Theorem.

Case 2 of the theorem is if $f(n) = O(n^{\log_b^a})$

then $T(n) = O(n^{\log_b^a} \cdot \log n)$

$\therefore$ we get $T(n) = O(n \log n)$.

For large n, $\left\lceil \frac{n}{2} \right\rceil \approx \frac{n}{2}$

Master theorem holds for general n, even if n is not a power of b.

Question 4
Answer 4

Emily wants to build the largest staircase possible using n marbles. A staircase of size k has k rows, where the i^{th} row contains i marbles. The total marbles required for a staircase of size k is $g(k) = \frac{k(k+1)}{2}$. The goal is to find the largest k such that $g(k) \leq n \leq g(k+1)$

a] Divide & conquer Algorithm (Binary search)

1. set lbound = 1 (min. possible rows) and ubound = n (max possible rows).
2. while lbound ≤ ubound :
   2.1 compute midpoint $k = lbound + \left[\frac{ubound - lbound}{2}\right]$
   2.2 calculate $g(k) = \frac{k(k+1)}{2}$.
   2.3 situations :
      2.3.1 If $g(k) = n$; Return k (exact match)
      2.3.2 If $g(k) < n$; search the upper half by setting lbound = k+1.
      2.3.3 If $g(k) > n$; search the lower half by setting ubound = k-1.

3. Result:

The largest valid k is ubound (since the loop exits when lbound > ubound)

b.] Recurrence Relation

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

→ At each step, the search space (possible value of k) is halved.

→ The O(1) term accounts for computing g(k) and comparisons.

c.] Solving the Recurrence and Time complexity

$$T(n) = T(n/2) + O(1)$$

Using the Master Theorem we get:

$a = 1, b = 2$, $f(n) = O(1)$.

$$n^{\log_b a} = n^{\log_2 1} = n^0 = 1$$

we observe that $f(n) = \Theta(n^{\log_b a})$

This corresponds to case 2 of the Master's Theorem where $f(n) = \Theta(n^{\log_b a} \log^k n)$ with $k = 0$

Applying case 2 we get

$$T(n) = \Theta\left(n^{\log_b^a} \log^{k+1} n\right)$$

Substituting the values we get

$$T(n) = \Theta\left(1 \cdot \log^{0+1} n\right) = \Theta(\log n)$$

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

∴ Algorithm uses binary search to find the largest k such that $\frac{K(K+1)}{2} \leq n$.

Recurrence : $T(n) = T(n/2) + O(1)$

Time complexity : $O(\log n)$