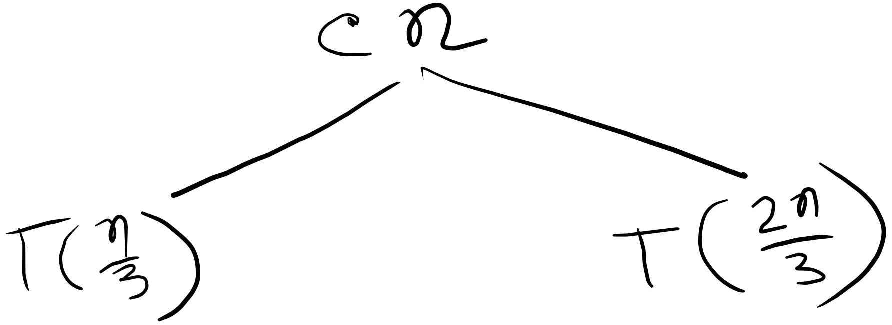
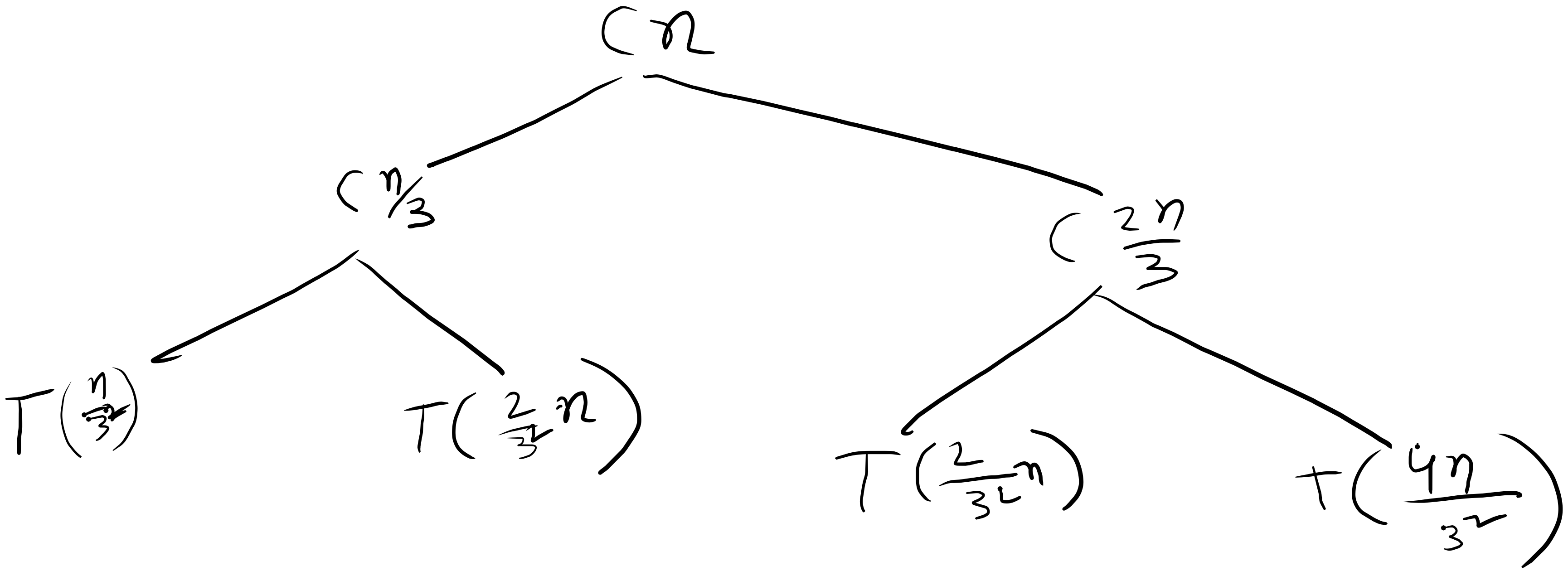
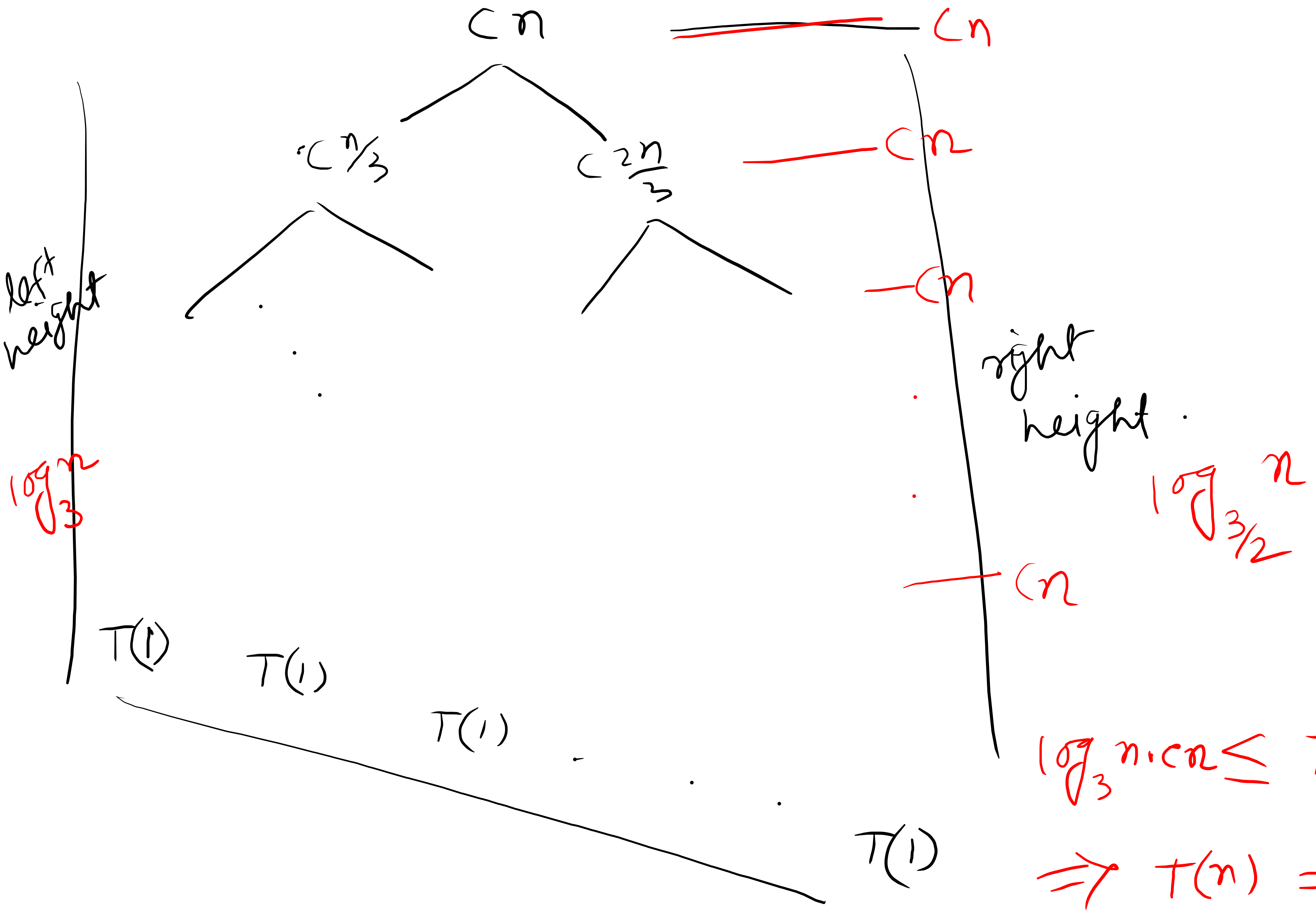
Recursion tree method

Ex^m $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn$ Find $T(n)$ using recursion tree.

$T(n)$







$$\log_3 n \cdot cn \le T(n) \le \log_{3/2} n \cdot cn$$

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

The master method

This method applies to the recurrences of the form

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

$a \geq 1$, $b > 1$ and f is asymptotically positive

Three common cases based on comparing $f(n)$ and $n^{\log_b a}$

Case 1: $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ for $\epsilon > 0$

$f(n)$ grows polynomially slower than $n^{\log_b a}$ by a factor n^{ϵ}

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

case 2 $f(n) = \theta\left(n^{\log_b a} \lg^k n\right)$

Solution: $T(n) = \theta\left(n^{\log_b a} \cdot \lg^{k+1} n\right)$

case 3: $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$

Addition condition

$a f\left(\frac{n}{b}\right) \leq c f(n)$ for some $c < 1$

regularity condition

Solution: $T(n) = \theta\left(f(n)\right)$

Ex^m i) $T(n) = 4T(\frac{n}{2}) + n$

$a = 4, \; b = 2, \; n^{\log_b a} = n^{\log_2 4} = n^2$

$f(n) = n$

$f(n) = O(n^{2-1}) = O(n^{\log_b a - 1})$

case 1 of master method applies here

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

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

$T(n) = \theta(n^2 \lg n)$

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

$n^3 = \Omega(n^{2+1})$

$af(\frac{n}{b}) \leq cf(n)$ for some $c < 1$

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

$\frac{n^3}{2} \leq c n^3$

$c \geq \frac{1}{2}$

any value of c where, $.5 \leq c < 1$ satisfies regularity condition

solution $T(n) = \Theta(n^3)$

iv) $T(n) = 4T\left(\frac{n}{2}\right) + \frac{n^2}{\lg n}$

$f(n) = \frac{n^2}{\lg n}$ $\qquad$ $n^{\log_b a} = n^2$

Master method cannot applied here

$f(n) = n^2 \rightarrow$ upper bound

$f(n) = n \Rightarrow$ lower bound.

your solution lies between these values.

H.W. Try examples

Algorithm design techniques

Divide and conquer paradigm

1. Divide the problem (instance) into subproblems.
2. Conquer recursively solving the subproblems.
3. Combine the solutions of the subproblems.

Ex^m Binary search problem

Def^n: Given an array of n sorted numbers and an number K verify whether K is in the array or not.

simple algorithm:

Traverse the array one by one and compare each element

running time:- $O(n)$

can we do better?

Divide : check for the middle element

Conquer : Recursively search one subarray

Combine : Trivial.

BinarySearch (A, K, low, high) — $T(n)$

if low > high — $\Theta(1)$

return no — $\Theta(1)$

else

mid = $\left\lfloor \frac{low+high}{2} \right\rfloor$ — $\Theta(1)$

if K == A[mid] — $\Theta(1)$

return yes — $\Theta(1)$

else if K > A[mid] — $\Theta(1)$

BinarySearch (A, K, mid+1, high) — $T(\frac{n}{2})$

$\Theta(1)$

else

BinarySearch (A, K, low, mid−1) — $T(\frac{n}{2})$

return no — $\Theta(1)$

total time

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

master method

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