CENG 218 - Spring 2023

Recitation 10.04.2023

Exercise 1 (Algorithm Design)

Closest pair. Given an unsorted list of N integers, find the pair that is closest to each other. For example, given [15 4 8 2 5 13 18 21 10] the output is [4 5].

Hint: Sometimes solving a problem is equivalent to reducing it into a known problem.

Sorting first:

Comparison sort (Merge sort)

$O(N \lg N)$ sorting

+

$O(N)$ distance check

+

$O(N \lg N)$

[2 4 5 8 10 13 15 18 21]

d=2 d=1 d=3 d=2

naive solution

brute-force solution

Exercise 2 (Algorithm Design)

Find an element in a sorted and rotated array. Given a sorted array of N integers that has been rotated an unknown number of times, give an $O(\log N)$ algorithm that finds an element in the array. You may assume that the array was originally sorted in increasing order.

Example Input: Find 5 in array [15 16 19 20 25 1 3 4 5 7 10 14]

Output: 8 (the index of 5 in the array)

sorted & rotated

left = 1

8

[1 3 4 5 7 . . . . . 16 19 20 25]

1st: check the middle element & compare with the first & last

binary search: O(lgN)

N/2

m = 1
f = 15
last = 1
x = 25

15 ... 25 ... 14

?

5 is somewhere here

2nd step: [15 16 19 | 20 25 1]

m = 20
f = 15
l = 1
x = 25

sorted continue with binary search

eliminate half

3rd m, f, l,

O(lg N)

Exercise 3 (Algorithm Design)

You are given an *n* x *n* matrix, where every row and column is sorted in increasing order.
A 5x5 example is given on the right.

```
matrix = { {10, 20, 30, 37, 47},
           {15, 25, 35, 45, 55},
           {27, 29, 38, 48, 57},
           {32, 33, 39, 50, 60},
           {35, 39, 44, 53, 62}};
```

5×5

The problem is finding the location of a given key in the matrix or report if the key is not in the matrix. This is a search problem but instead of a 1D sorted array you search in a 2D sorted array. The following is an algorithm to solve this problem:

$T(n) = 3T(n/2) + \Theta(1)$

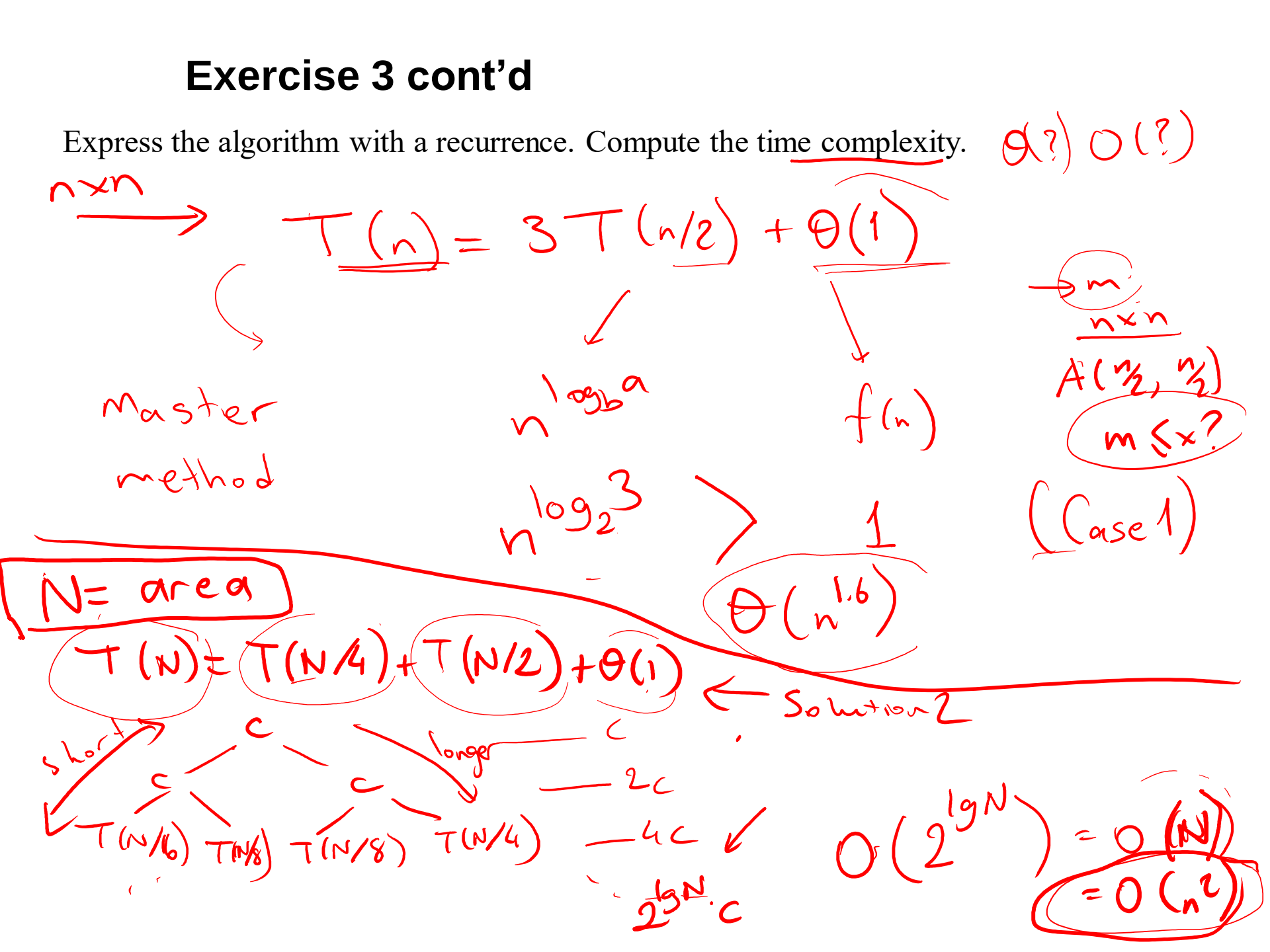
Algorithm:

0) Find the middle element in the matrix.
1) If the middle element is same as the key, return.
2) If the middle element is less than the key then 2a) Search submatrix on lower side of middle element (recursively) 2b) Search submatrix on the up-right (recursively)
3) If the middle element is greater than the key then 3a) search submatrix on left side of middle element (recursively) 3b) search submatrix on the up-right (recursively)

$n/2 \times n/4$

Exercise 3 cont'd

Express the algorithm with a recurrence. Compute the time complexity.



Exercise 4

Use the master method to solve the following recurrences.

a) $T(n) = 8T(n/4) + n^2$

$n^{\log_4 8}$ $\quad n^2 \quad \Rightarrow \Theta(n^2)$

b) $T(n) = 16T(n/4) + n^2$

$n^{\log_4 16} \simeq n^2 \Rightarrow \Theta(n^2 \lg n)$

polynomially same

(case 2)

Exercise 5

A divide-and-conquer algorithm to solve a problem divides the problem into two sub-problems recursively. For an n-sized problem, sizes of the subproblems are $n-1$ and $n/3$. Also a constant time is spent to divide the problem and combine the solutions.

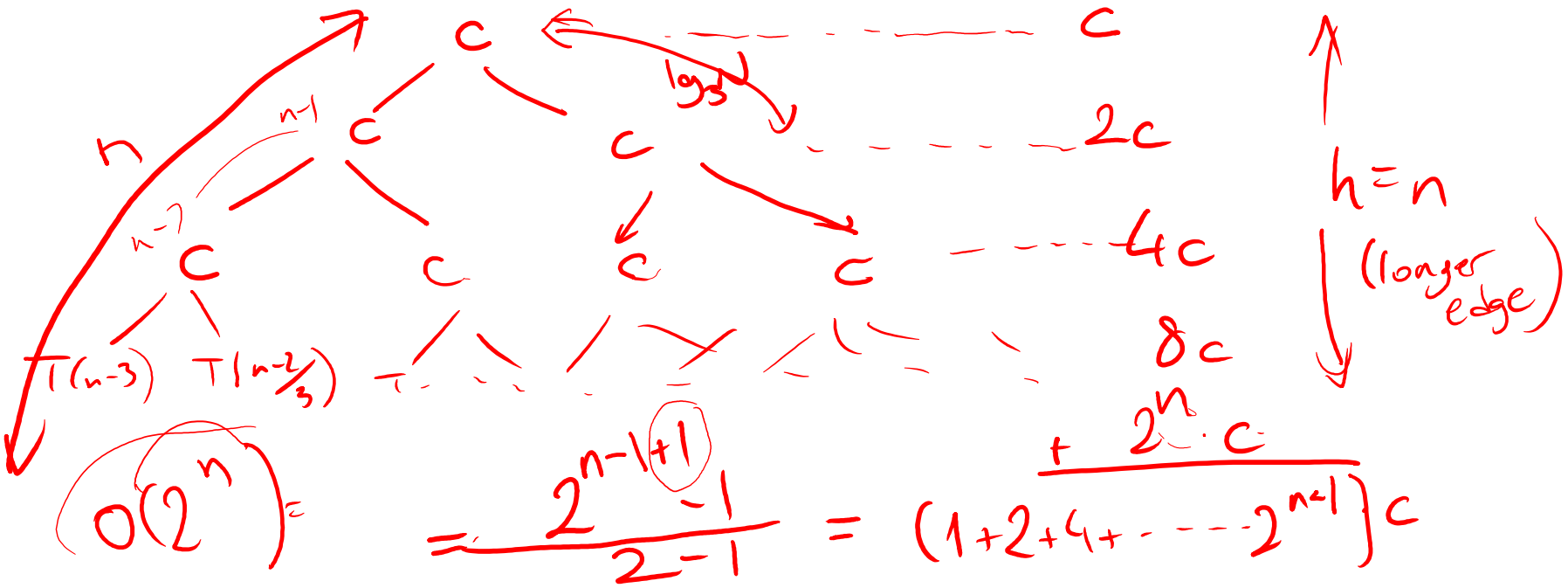
a) Write the recurrence relation.

b) Solve the recurrence with recursion tree and find a good upper bound.

a) $T(n) = T(n-1) + T(n/3) + \Theta(1)$

sum of rows

height



$O(2^n) = \frac{2^{n-1+1}-1}{2-1} = (1+2+4+\cdots+2^{n-1}+2^n)\cdot c$

Exercise 6

Use substitution method to verify that this recurrence is $O(3^n)$.

$$T(n) = 3T(n-1) + \Theta(1)$$

$T(n) \le 3 \cdot (c \cdot 3^{n-1}) + 1$ ✓

$T(n) \le c \cdot 3^n + 1$

desired − residual

✗ didn't work

$T(n) \le c \cdot 3^n$

$\leftarrow T(n-1) \le c 3^{(n-1)}$

$T(n/2) \le c 3^{n/2}$

$O(3^n - d)$ $T(n) \le c \cdot 3^n - d$

$\leftarrow T(n-1) \le c 3^{n-1} - d$

$T(n) \le 3 \cdot (c 3^{n-1} - d) + 1$

$\le 3 \cdot c 3^{n-1} - 3d + 1$

$\le c \cdot 3^n - (3d - 1)$

desired

residual ≥ 0

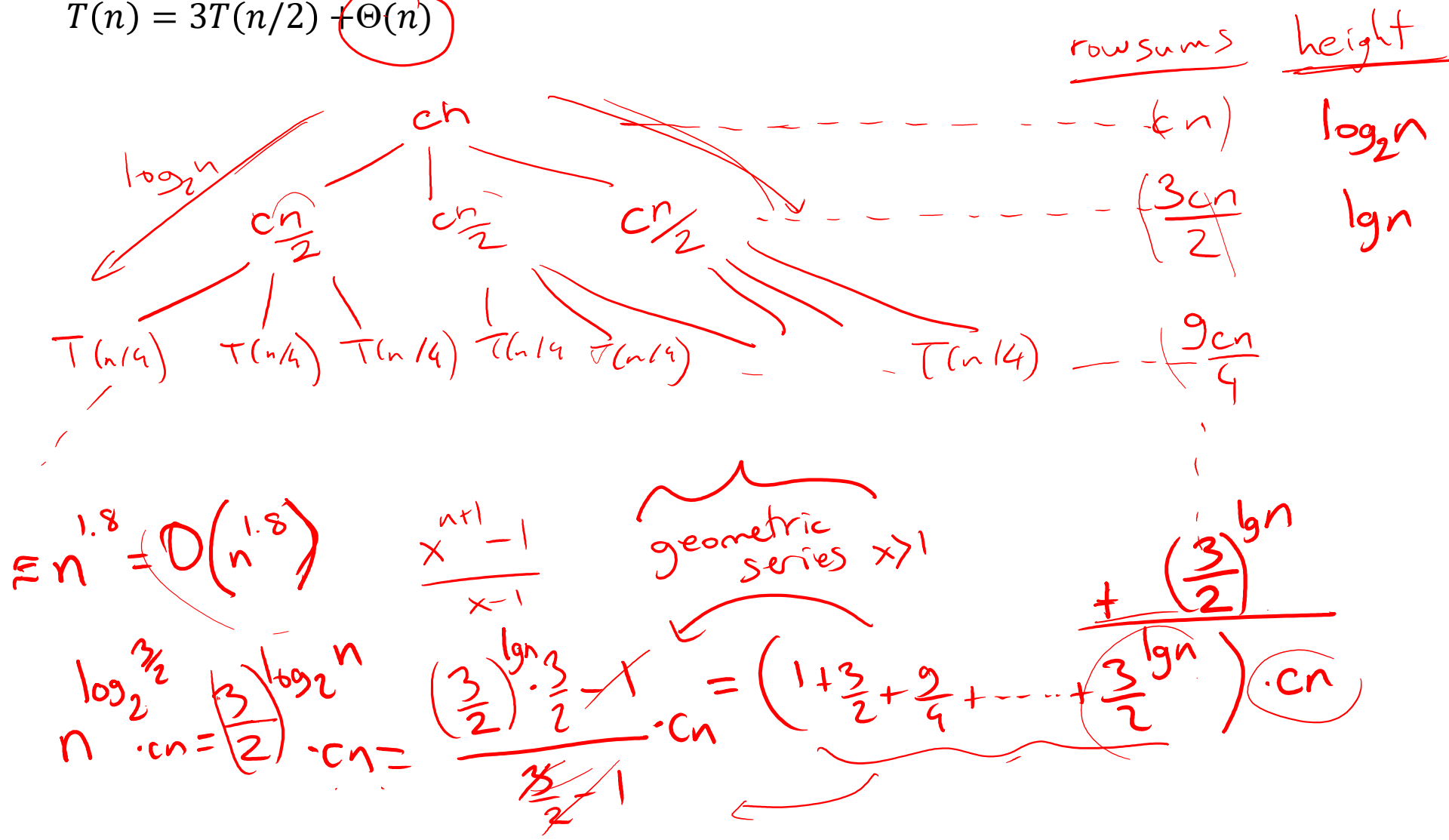
$c = 1$

$d \ge \frac{1}{3}$ ✓ Inductive step:

$T(1) = 1 \cdot 3^n$ ✓ Basis step

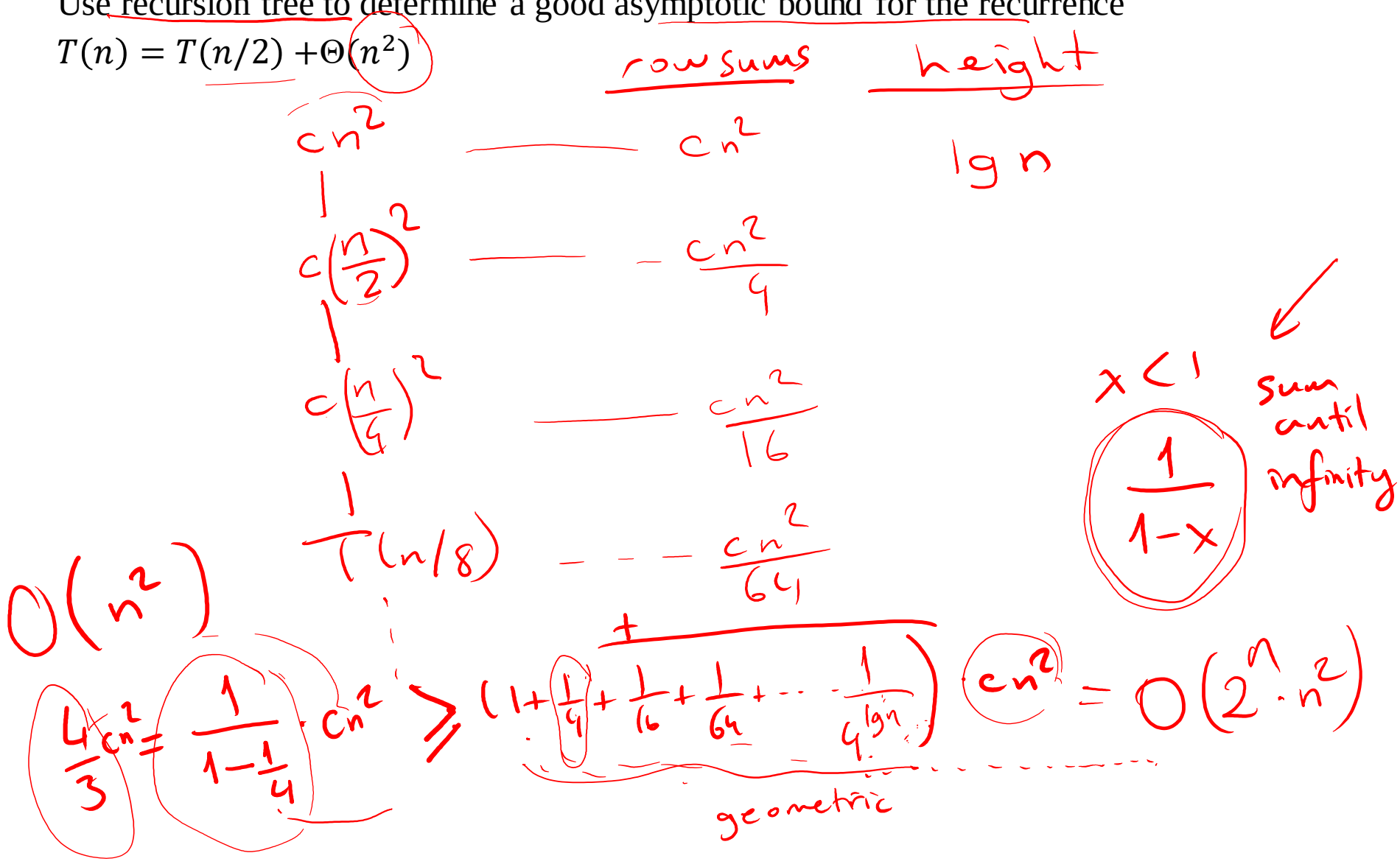
Exercise 7 (4.4.1 in the book)

a) Use recursion tree to determine a good asymptotic bound for the recurrence $T(n) = 3T(n/2) + \Theta(n)$



Exercise 8 (4.4.2 in the book)

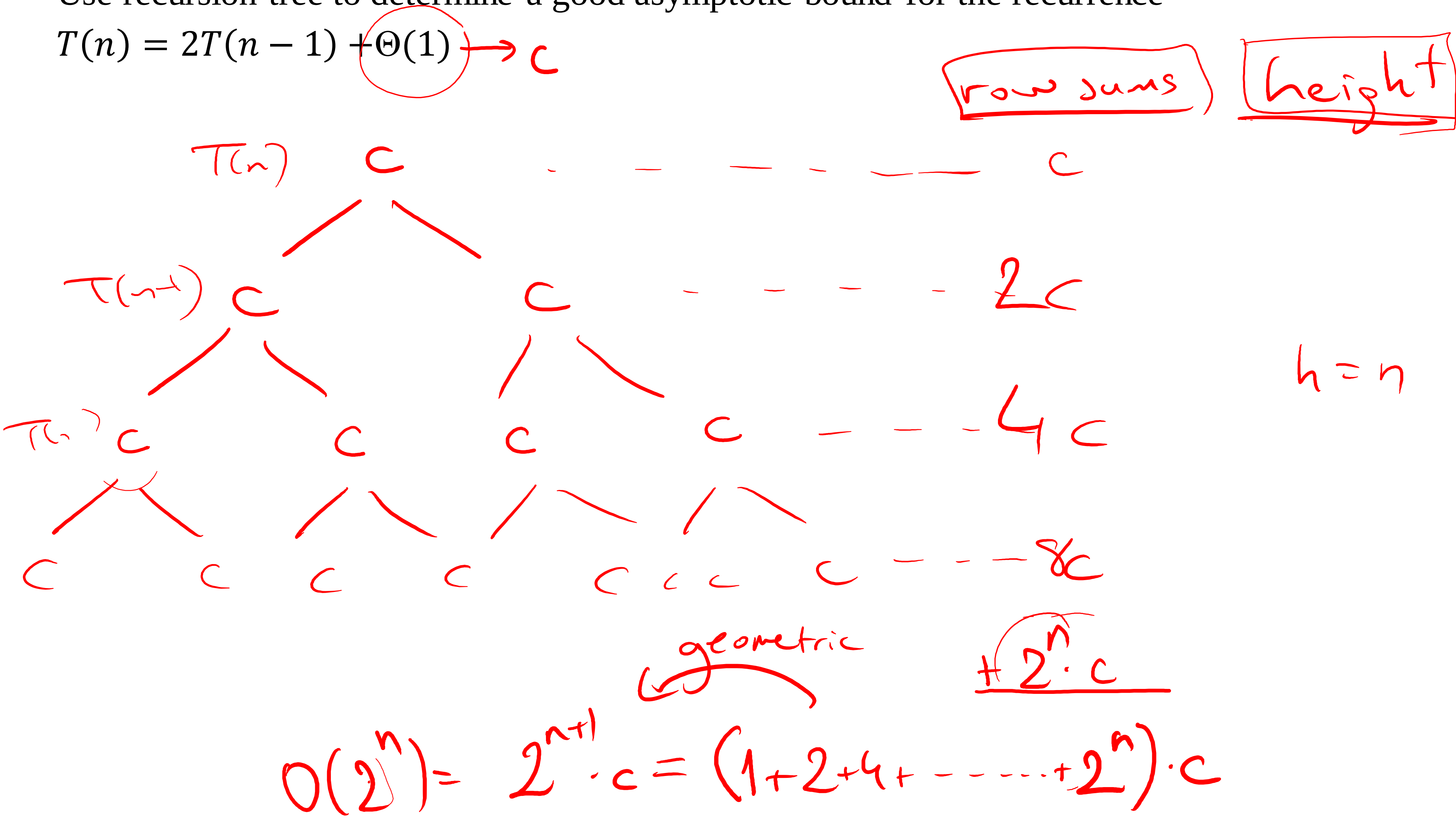
Use recursion tree to determine a good asymptotic bound for the recurrence $T(n) = T(n/2) + \Theta(n^2)$



Exercise 9 (4.4.4 in the book)

Use recursion tree to determine a good asymptotic bound for the recurrence

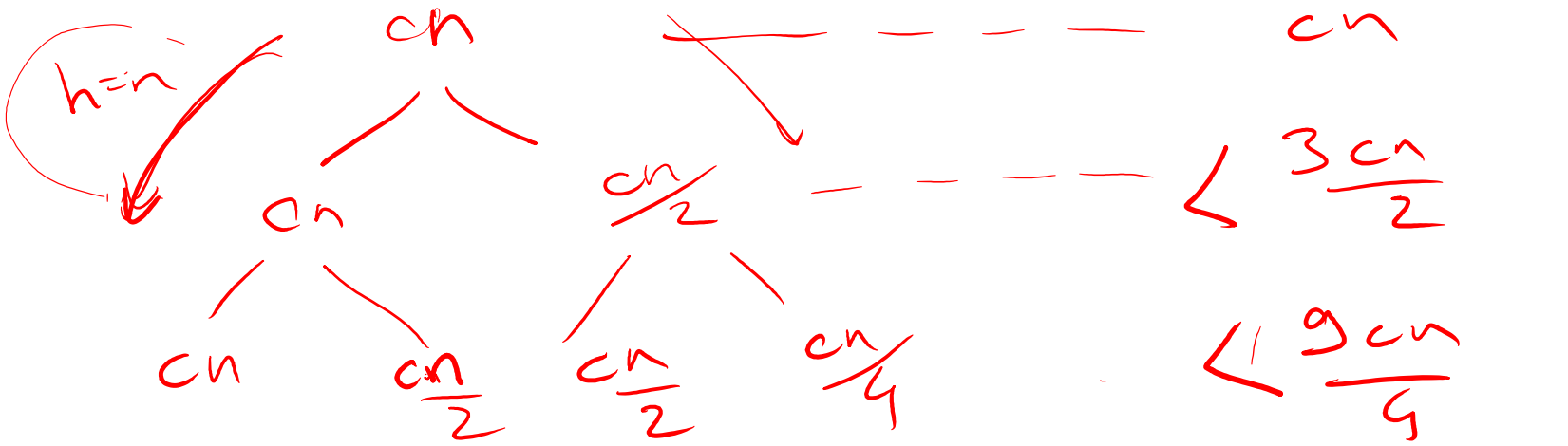
$T(n) = 2T(n-1) + \Theta(1)$



Exercise 10 (4.4.5 in the book)

Use recursion tree to determine a good asymptotic bound for the recurrence

$T(n) = T(n-1) + T(n/2) + \Theta(n)$ → cn



$O(1.5^n \cdot n)$

$O\left(\left(\frac{3}{2}\right)^n \cdot n\right) = O\left(\frac{\left(\frac{3}{2}\right)^{n+1} - 1}{\frac{3}{2} - 1} \cdot cn\right) = \left(1 + \frac{3}{2} + \frac{9}{4} + \cdots + \left(\frac{3}{2}\right)^n\right) \cdot cn$

geometric

$\left(\frac{3}{2}\right)^n \cdot cn$

Exercise 11 (4.4.5 with substitution method)

Use substitution method to determine if $T(n) = T(n-1) + T(n/2) + n$ is $O(n2^n)$.

$T(n) \leq c\cdot(n-1)\cdot 2^n + c\frac{n}{2}2^{n/2} + n$

$\leq \frac{cn\cdot 2^n}{2} - \frac{c2^n}{2} + c\frac{n}{2}\cdot 2^{n/2} + n$

$\leq cn2^n$

$\leq \frac{cn2^n}{2} + \frac{cn2^n}{2} - \frac{c2^n}{2} + n$

$\leq cn\cdot 2^n - \left(\frac{c2^n}{2} - n\right)$

desired — residual ≥ 0 ?

Basis step: $T(1) \leq 1\cdot 1\cdot 2^1$ ✓ $c=1$

$T(n) \leq c\cdot n\cdot 2^n$

$T(n-1) \leq c(n-1)2^{n-1}$

$T(n/2) \leq c\frac{n}{2}\cdot 2^{n/2}$

$T(n)$ → subproblem 2 → subproblem 2 → ... → sum

@ $n=1$ $c\cdot\frac{2}{2} - 1$ $c=1$ res ≥ 0

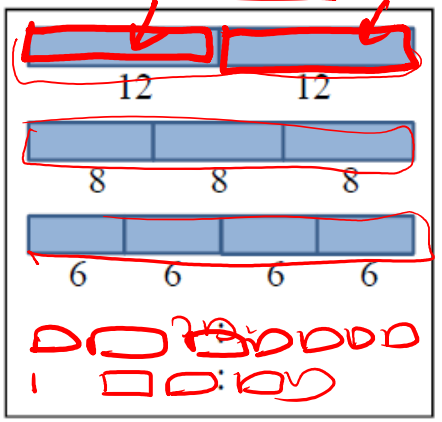
@ $n=2$ $c\cdot\frac{4}{2} - 2$ $c=1$ res ≥ 0

$n>2$ $c\cdot\frac{2^n}{2} - n$ res ≥ 0

Exercise 12 (2020 midterm exam question)

Suppose you want to sort 4100 numbers of 24 bits each. You decided to use Radix sort.

a) Remember in Radix Sort we can decompose our original number into pieces of *r*-bit numbers and we can use a complementary stable sort (e.g. counting sort) to sort those pieces. In this sense, 24-bit numbers can be decomposed into 12 pieces of 2-bit numbers, 8 pieces of 3-bit numbers, 6 pieces of 4-bit numbers, 4 pieces of 6-bit numbers etc. What is the optimum value of *r* in our problem? I.e. which *r* minimizes the running time of Radix Sort?



b) What is the time complexity of Radix Sort with the optimum *r* ?

c) Is the time complexity you found in (b) better than the time complexity of Mergesort?

a) Optimum running time of Radix Sort happens @ $r = \lg n$

$r = \lg 4100$ $r = 12$ $(n = 4100)$

b) $\Theta(d \cdot n)$ $2^b = n^d$ $2^{24} = (4100)^d = (2^{12})^d \rightarrow d = 2$

$\Theta(2n)$

c) $\Theta(n \lg n) = \Theta(n \cdot \lg(4100)) = \Theta(12n)$