Week 2 Tuesday Lecture Notes

Before Class: Read DPV 2.1 and 2.2:

Summary: Readings discuss Recurrence Relations and Multiplication reduction example to improve runtime speed. Reducing an algorithm from 4 to 3 multiplications doesn't seem important but when applied recursively the reduction makes a big difference and does affect Big-O Complexity.

- Wednesday at 5:45-6:45pm there will be a recitation to discuss Week 1 review problem answers and questions for the quiz on Thursday.

Week 1 problem answers have been posted on canvas (Try to work through the problems on your own first before looking at the solutions).

Lecture: Continue Recurrence Relation, Recursion Tree and the Master Theorem: Start on PowerPoint 2 Page 8

Induction can be viewed as a form of Recursion.

Recurrence Relation to Describe the runtime.

Telescoping: Similar to recursion. Calculating how many layers of recursion there are. Much more math heavy than using a Recursion Tree.

Taking $T(n) = T(n-4) + c$

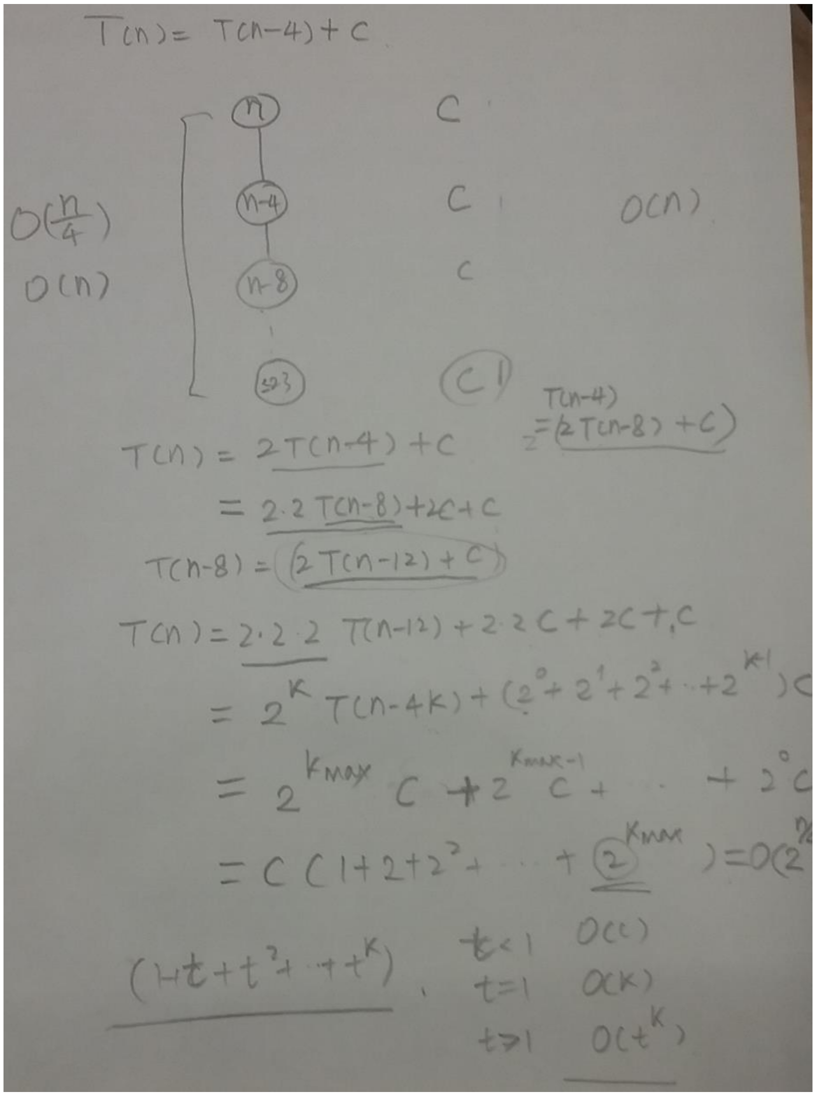
$== T(n-4) = T(n-8) + 2c$

$== T(n) = T(n-4k) + kc$

Runtime of this recurrence relation: $O(n)$

Asymptotic complexity analysis simplifies $O(c + n/4)$ to $O(n)$

**Picture of Linear Tree for T(n) = T(n-4) + c. **



What about T(n) = 2T(n-4) + c? Using Telescoping we step through the problem like this:

= 2T(n-8) + c

= 2*2*T(n-8) +2C + C

T(n-8+ = 2T(n-12) + C

T(n) = 2* 2 * 2 T(n-12)+4C+2C+C

$= 2^k\ T(n-4k) + (2^0 + 2^1 + 2^2 + \ldots + 2^{k-1})C$

$= 2^{kmax} C + 2^{kmax-1} C + \ldots. + 2^0 C$

$= C(1+2+2^2+2^{kmax})$

$= O(2^{n/4})$

The Geometric Series: $(1+t+t^2+\ldots+t^k)$

- $t<1$ $O(t)$
- $t=1$ $O(k)$
- $t>1$ $O(t^k)$
-

What about T(n) = 2T(n-4) + c? Using Recursive Tree:

Picture of Recursive Tree

Big-O $(2^{n/4})$ Same but much easier to process

Merge_sort Recurrence:

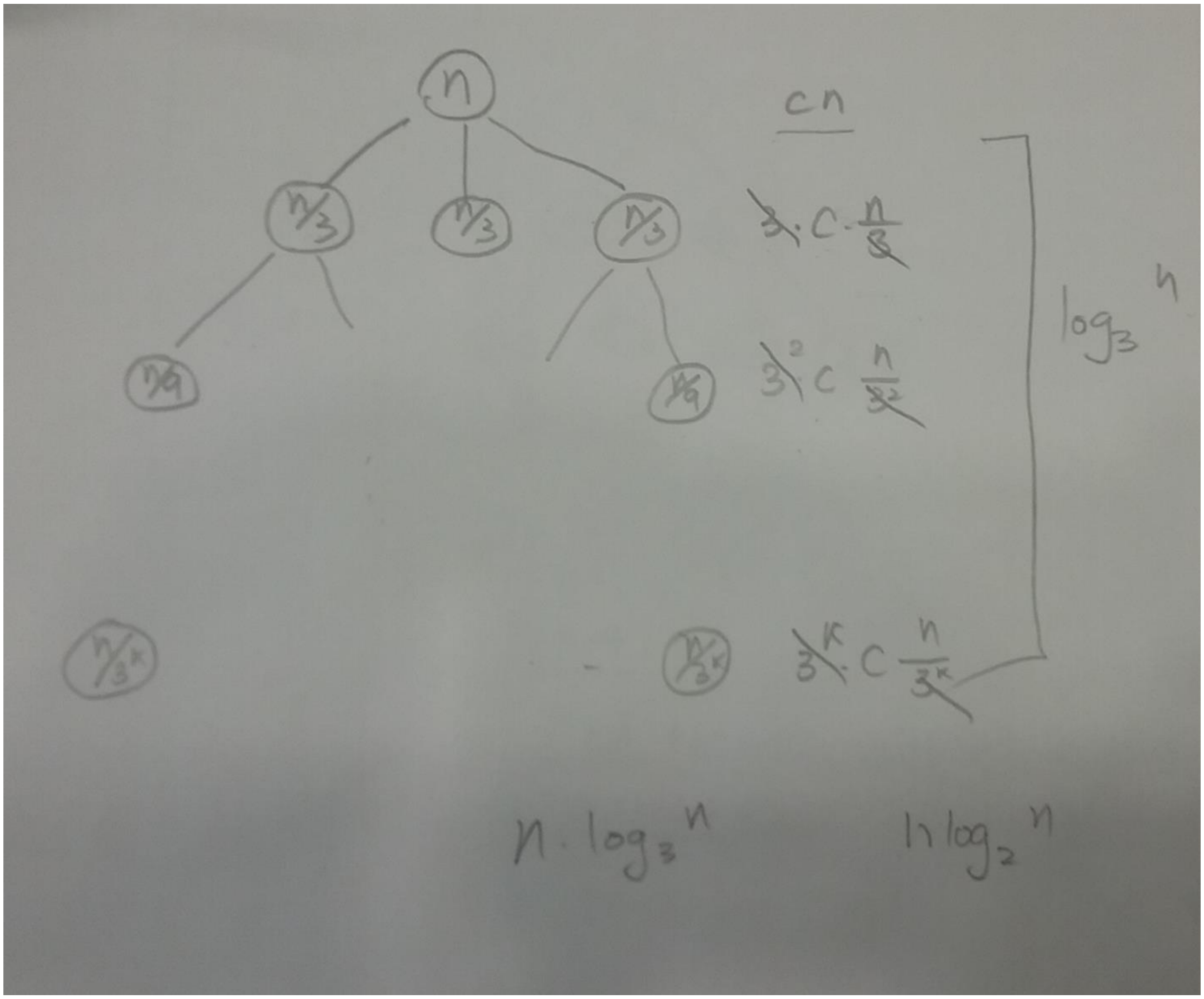
Runtime: 1. Break into half – constant time - c

2. Sort the two half size arrays – 2T(n/2)

3. Merge two together – Constant * n – cn

Recurrence: T(n) = 2T(n/2) + cn

Solve Using a Recursion Tree: Should output O(n log n)

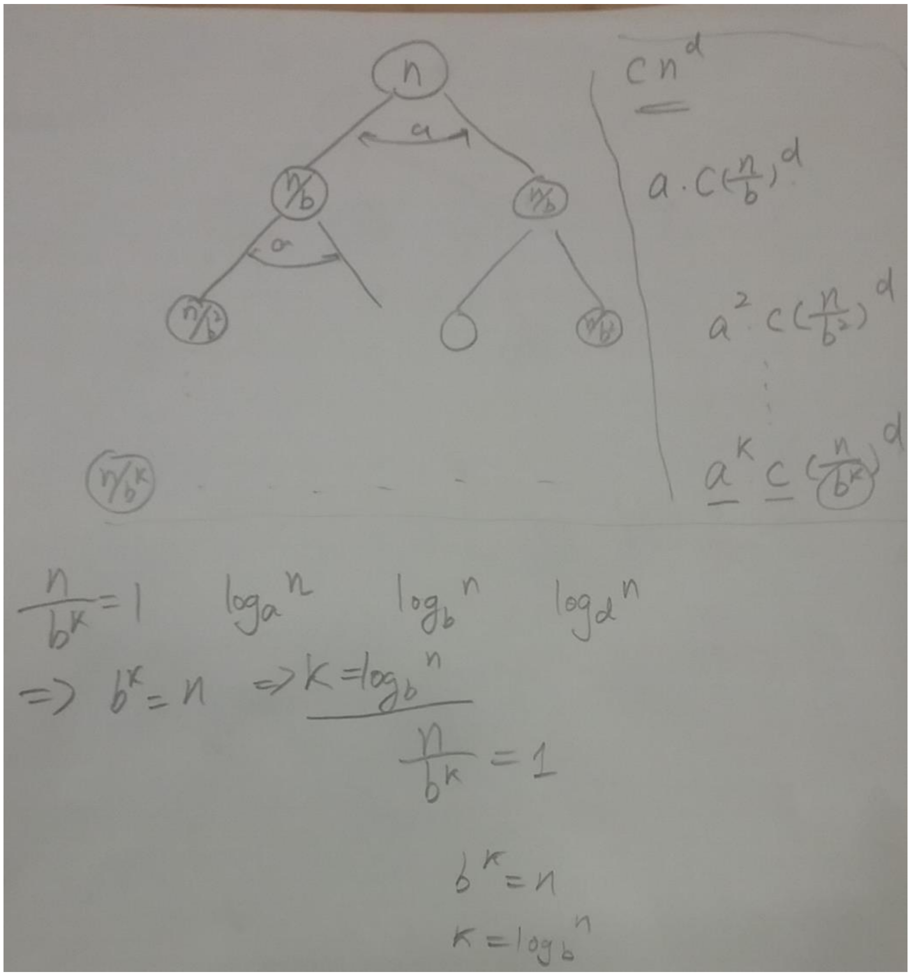


T(n) = 3T(n/3) + cn Example Picture*

More Generally…

Master Equation: T(n) = aT(n/b) + cn^d for constants a,b,c,d.

Recursion Tree for This General Case.



Master Theorem: for Recursion $T(n) = aT(n/b) + cn^d$

Case 1: $a < b^d$, or equiv. $d > \log_b a$, $T(n) = O(n^d)$

Case 1: $a = b^d$, or equiv. $d > \log_b a$, $T(n) = O(n^d \log n)$

Case 1: $a < b^d$, or equiv. $d > \log_b a$, $T(n) = O(n^{\log_b a})$

Start of Lecture 3: Divide and Conquer:

1. Break up large problem into smaller problems
2. Solve those smaller problems independently

3. Glue the small problems back into a large problem.

-Hardest part is gluing those smaller problems back together into the large problem.

Inversion Counting:

An array containing numbers 1-n

Number of inversions are number of pairs (i,j) such that $i<j$ and $A[i] > A[j]$

Example: Array **Input** (1,4,2,5,3)

Inversions: 4 and 2, 4 and 3, 5 and 3.

Simplified: Inversions are any out of order pairs.

Movie Example:

Set of n movies: (See PowerPoint)

Mine: 1,2,3,4,…,n

Yours: 5,4,1,…,n

How many differences there are?

Count how many inversions there are.

We can use brute force to compute this in $O(n^2)$

Can we be faster? (Using Divide and Conquer)

- Break the array in half.
- Count the left half
- Count the right half
- Count the inversions between these two halfs.

Ended on Page 5 of Lecture 3 PowerPoint:

<u>End of Week 2 Tuesday Lecture Notes</u>

~Information composed by Notetaker Scott Russell for CS 325 **DAS** student