RUNTIME ANALYSIS OF FUNCTIONS

ANALOGY TO REAL NUMBERS

$f(n) = O(g(n))$	$a \leq b$
$f(n) = \Omega(g(n))$	$a \geq b$
$f(n) = \Theta(g(n))$	$a = b$
$f(n) = o(g(n))$	$a < b$
$f(n) = \omega(g(n))$	$a > b$

EXAMPLES

$logn$	$\in$	$O(log^2 n)$	$log^2 n$	$\in$	$O(log^2 n)$	$nlogn$	$\notin$	$O(log^2 n)$
$logn$	$\notin$	$\Theta(log^2 n)$	$log^2 n$	$\in$	$\Theta(log^2 n)$	$nlogn$	$\notin$	$\Theta(log^2 n)$
$logn$	$\notin$	$\Omega(log^2 n)$	$log^2 n$	$\in$	$\Omega(log^2 n)$	$nlogn$	$\in$	$\Omega(log^2 n)$
$logn$	$\in$	$o(log^2 n)$	$log^2 n$	$\notin$	$o(log^2 n)$	$nlogn$	$\notin$	$o(log^2 n)$
$logn$	$\notin$	$\omega(log^2 n)$	$log^2 n$	$\notin$	$\omega(log^2 n)$	$nlogn$	$\in$	$\omega(log^2 n)$

logn

$log^2 n$ nlogn → • $log^2 n$ grows foster than logn. (because $log^2 n = (logn)^2$)

• n grows foster than logn

So, $log^2 n = logn * logn$ grows slower than $n * logn$.

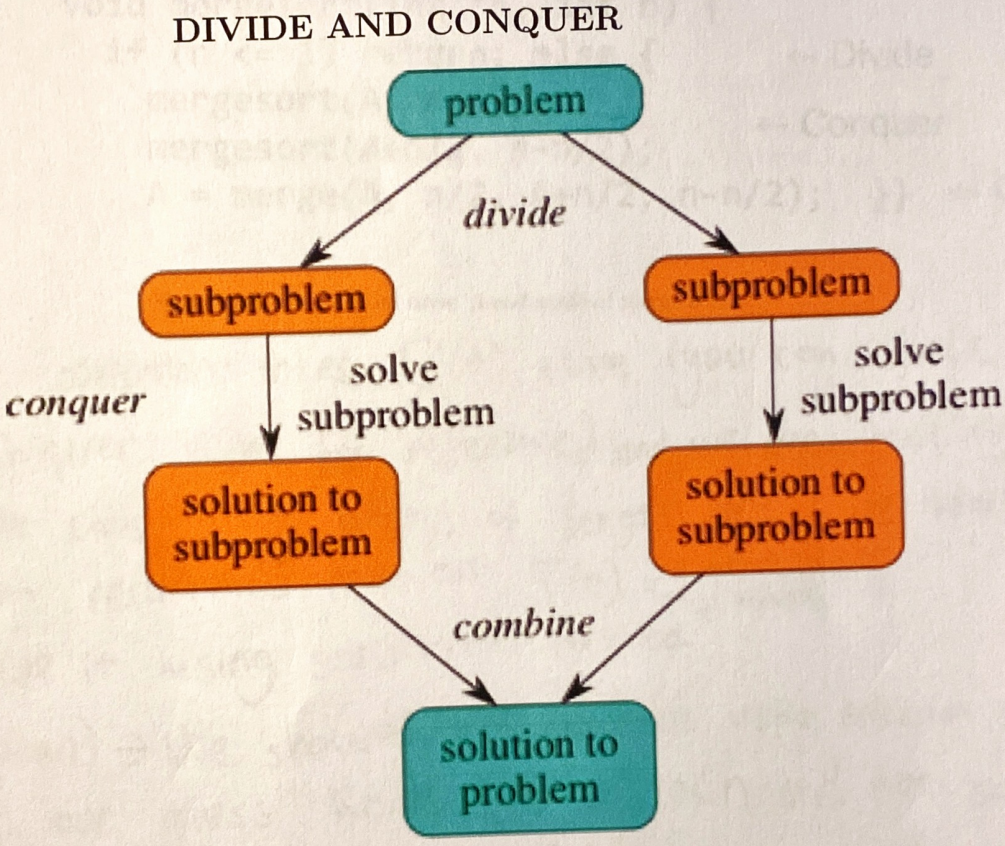


Figure 1: Taken from https://www.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/divide-and-conquer-algorithms

```
void mergesort(int *A, int n) {
  if (n <= 1) return; else {          ← Divide
    mergesort(A, n/2);                ← Conquer
    mergesort(A+n/2, n-n/2);
    A = merge(A, n/2, A+n/2, n-n/2); }}  ← Combine
```

Figure 2: Example from class: pseudocode of merge sort

Since merge operation takes $\Theta(n)$ time (you can check Divide-and-Conquer slides for reference), and we are making two recursive calls using arrays of length $n/2$, we have the following recurrence relation: $T(n) = 2T(n/2) + n$.

Let us solve it using substitution method*:

Guess: $O(n \log n)$ → We showed this in class using recursion trees.

We assume our guess holds for all $m < n$ and for some constant $c > 0$. We pick $m = \frac{n}{2} \Rightarrow T(n/2) \leq c\frac{n}{2}\log(\frac{n}{2}) \Rightarrow$

$$T(n) \leq 2c\frac{n}{2}\log(\frac{n}{2}) + n$$
$$= cn\log(\tfrac{n}{2}) + n$$
$$= cn(\log n - \underbrace{\log 2}_{1}) + n$$
$$= cn\log n - cn + n \leq n\log n \text{ for } c = 1.$$

*Refer to CLRS book page 83-84 for a good explanation of this method.

EXERCISES FROM THE BOOK

3-2 *Relative asymptotic growths*

Indicate, for each pair of expressions (A, B) in the table below, whether A is O, o, Ω, ω, or Θ of B. Assume that $k \geq 1$, $\epsilon > 0$, and $c > 1$ are constants. Your answer should be in the form of the table with "yes" or "no" written in each box.

	A	B	O	o	Ω	ω	Θ
a.	$\lg^k n$	n^ϵ	✓	✓	✗	✗	✗
b.	n^k	c^n	✓	✓	✗	✗	✗
c.	$\sqrt{n}$	$n^{\sin n}$	✗	✗	✗	✗	✗
d.	2^n	$2^{n/2}$	✗	✗	✓	✓	✗
e.	$n^{\lg c}$	$c^{\lg n}$	✓	✗	✓	✗	✓
f.	$\lg(n!)$	$\lg(n^n)$	✓	✓	✗	✗	✗

a) n grows faster than $\log n$

b) n^k is polynomial, c^n is exponential. Exponential grows faster than polynomial.

c) $\sin n$ oscillates between $[1,1]$. Cannot lower/upper bound $n^{\sin n}$ using $\sqrt{n} = n^{\frac{1}{2}}$.

d) 2^n grows faster than $2^{n/2} = 2^{\frac{1}{2}\cdot n} = \sqrt{2}^n$ since $2 > \sqrt{2}$.

e) $n^{\log c} = c^{\log n}$ (property of logarithm function)

f) Since $\log(n!)$ and $\log(n^n)$ are both in log, we compare the inner function with each other: $n! = 1.2.3 \ldots n$ & $n^n = \underbrace{n \cdot n \cdot n \ldots n}_{n \text{ times}}$. So n^n grows faster than $n!$

4.4-2

Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = T(n/2) + n^2$. Use the substitution method to verify your answer.

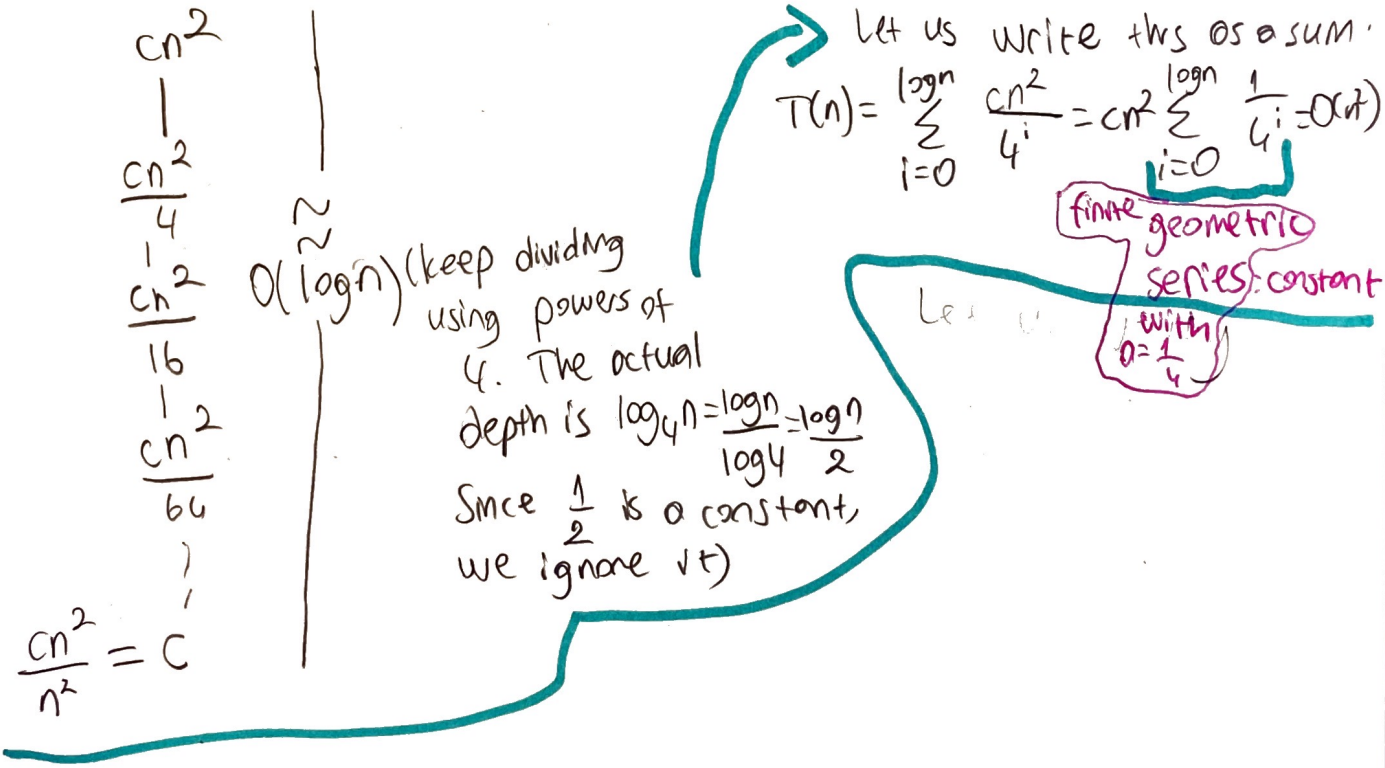
$T(n/2) = T(n/4) + \left(\frac{n}{2}\right)^2 = T(n/4) + \frac{n^2}{4}$

$T(n/4) = T(n/8) + \left(\frac{n}{4}\right)^2 = T(n/8) + \frac{n^2}{16}$

$T(n/8) = T(n/16) + \left(\frac{n}{8}\right)^2 = T(n/16) + \frac{n^2}{64}$

We have 1 recursive call with input of half size. and at each recursive call, we are doing $O(n^2)$ amount of work.

Let us build recursion tree.



4.4-2 -cont: Let us verify our guess using substitution method.

Recurrence: $T(n) = T(\lfloor n/2 \rfloor) + n^2$

Guess: $O(n^2)$

We assume our guess holds for all $m < n$ and for some constant $c > 0$. We pick $m = \lfloor n/2 \rfloor \Rightarrow T(\lfloor n/2 \rfloor) \leq c\left(\frac{n}{2}\right)^2 = \frac{cn^2}{4} \Rightarrow$

$T(n) \leq \frac{cn^2}{4} + n^2 = \left(\frac{c+4}{4}\right) n^2 = c_1 n^2$ holds for all $c > 0$.

c_1 constant
call this c_1

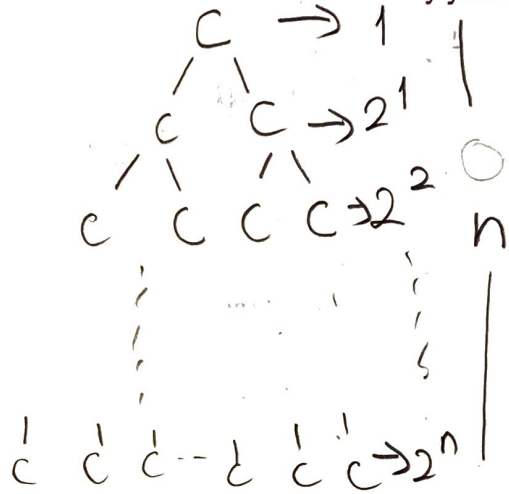
$\Rightarrow T(n) = O(n^2)$

4.4-4

Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = 2T(n-1) + 1$. Use the substitution method to verify your answer.

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

$T(n-2) = 2T(n-3)+1$



(since we go on 1 by 1 until we hit $\Theta(1)$)

Let us write this as a sum:

$$T(n) = \sum_{i=0}^{n} c2^i = O(2^n)$$

Let us verify our result using substitution method:

Guess: $O(2^n)$

We assume our guess holds for all $m < n$ and for some constant $c > 0$. We pick $m = n-1 \Rightarrow T(n-1) \leq c2^{n-1} \Rightarrow$

$T(n) \leq 2 \cdot c2^{n-1} + 1$

$= c2^n + 1$, holds for all $c > 0$. $\Rightarrow$ Thus, $T(n) = O(2^n)$.