CS2040

Tutorial 1: Asymptotic Analysis

Nicholas **Russell** Saerang (russellsaerang@u.nus.edu)

:)

INTRODUCTION

INTRODUCTION

- Nicholas **Russell** Saerang
- Year 4 Data Science and Analytics w/ minor in CS
- Took CS2040 in AY2020
- Taught CS2040(S) in AY2120, AY2210, AY2220, AY2320
- russellsaerang@u.nus.edu

>:)

ADMIN

Activities	Weightages
Tutorial attendance/participation	3%
Lab attendance	2%
In-lab Assignments	15% (1.5%/problem)
Take Home Assignments	12% (1.5%/problem)
Online Quiz	8% (4% each)
Midterm	20%
Final Exam	40%

The 3%

Attendance will be taken for all tutorials.

If you cannot attend a tutorial or miss a tutorial, do let me know, and in advance if possible.

Tips

Understand Material Well
Make sure you know **exactly how and why** everything works. There are no magic algorithms in CS2040.

Attempt Tutorials
Tutorial questions are hand-picked and supposed to expose you to a range of techniques that are applicable to a wide range of problems.

Do Extra Practice Questions
Exposure to more problems helps with problem solving. Ask me for more questions if you need.

01

BIG-O COMPLEXITY

Big-O Complexity

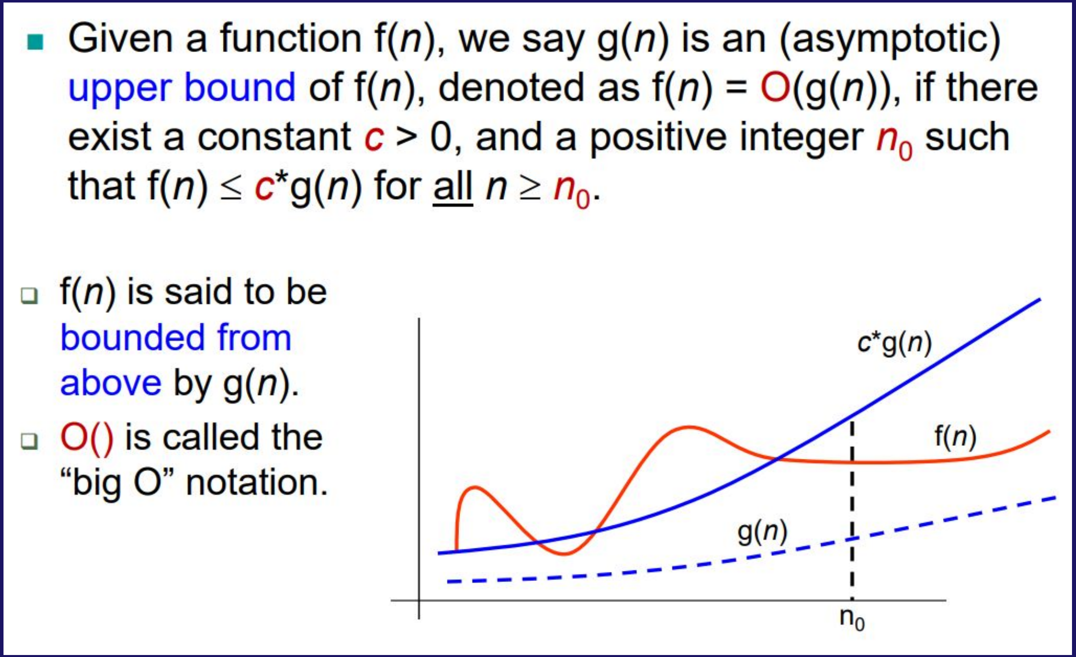
What you learnt in CS1010*

1. Get the dominant term
   e.g. $n^2 + n^4 + n^3 = O(n^4)$

2. Ignore constant factors, i.e. drop coefficient
   e.g. $3n^2 = O(n^2)$

Big-O Complexity

Big-O time complexity gives us an idea of the growth rate of a function.



- Given a function f(n), we say g(n) is an (asymptotic) upper bound of f(n), denoted as f(n) = O(g(n)), if there exist a constant $c > 0$, and a positive integer n_0 such that f(n) $\leq c$*g(n) for all $n \geq n_0$.

- f(n) is said to be bounded from above by g(n).
- O() is called the "big O" notation.

E.g.
$5n^2 = O(n^2)$
$c = 6$, $n_0 = 1$

$5n^2 \leq cn^2$
for all $n \geq n_0$

Big-O Complexity

$4n^2$	$\log_3 n$	$20n$	$n^{2.5}$
$n^{0.00000001}$	$\log n!$	n^n	2^n
2^{n+1}	2^{2n}	3^n	$n \log n$
$100n^{2/3}$	$\log[(\log n)^2]$	$n!$	$(n - 1)!$

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n$	$20n$	$n^{2.5}$
$n^{0.00000001}$	$\log n!$	n^n	2^n
2^{n+1}	2^{2n}	3^n	$n \log n$
$100n^{2/3}$	$\log[(\log n)^2]$	$n!$	$(n - 1)!$

Drop the coefficient! :D

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n$	$n^{2.5}$
$n^{0.00000001}$	$\log n!$	n^n	2^n
2^{n+1}	2^{2n}	3^n	$n \log n$
$100n^{2/3}$	$\log[(\log n)^2]$	$n!$	$(n - 1)!$

The base of the logarithm does not matter.

$$\log_3 n = \frac{\log_k n}{\log_k 3} = \frac{1}{\log_k 3}\log_k n = O(\log_k n)$$

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5}$
$n^{0.00000001}$	$\log n!$	n^n	2^n
$2^{n + 1}$	2^{2n}	3^n	$n \log n$
$100n^{2/3}$	$\log[(\log n)^2]$	$n!$	$(n - 1)!$

Again, drop the coefficient! :D

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5} = O(n^{2.5})$
$n^{0.00000001}$	$\log n!$	n^n	2^n
2^{n+1}	2^{2n}	3^n	$n \log n$
$100n^{2/3}$	$\log[(\log n)^2]$	$n!$	$(n - 1)!$

Constant in exponential matters when the base is a variable, which is n in this case.

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5} = O(n^{2.5})$
$n^{0.00000001}$ $= O(n^{0.00000001})$	$\log n!$	n^n	2^n
2^{n+1}	2^{2n}	3^n	$n \log n$
$100n^{2/3}$	$\log[(\log n)^2]$	$n!$	$(n - 1)!$

Constant in exponential matters when the base is a variable, which is n in this case.

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5} = O(n^{2.5})$
$n^{0.00000001} = O(n^{0.00000001})$	$\log n! = O(n \log n)$	n^n	2^n
2^{n+1}	2^{2n}	3^n	$n \log n$
$100n^{2/3}$	$\log[(\log n)^2]$	$n!$	$(n-1)!$

Note that

$$\log n! = \sum_{i=1}^{n} \log i \leq \sum_{i=1}^{n} \log n = n \log n = O(n \log n)$$

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5} = O(n^{2.5})$
$n^{0.00000001} = O(n^{0.00000001})$	$\log n! = O(n \log n)$	$n^n = O(n^n)$	2^n
2^{n+1}	2^{2n}	3^n	$n \log n$
$100n^{2/3}$	$\log[(\log n)^2]$	$n!$	$(n - 1)!$

Nothing to simplify here :)

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5} = O(n^{2.5})$
$n^{0.00000001} = O(n^{0.00000001})$	$\log n! = O(n \log n)$	$n^n = O(n^n)$	$2^n = O(2^n)$
2^{n+1}	2^{2n}	3^n	$n \log n$
$100n^{2/3}$	$\log[(\log n)^2]$	$n!$	$(n - 1)!$

Nothing to simplify here :)
Take note that variable in exponent matters!

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5} = O(n^{2.5})$
$n^{0.00000001} = O(n^{0.00000001})$	$\log n! = O(n \log n)$	$n^n = O(n^n)$	$2^n = O(2^n)$
$2^{n+1} = O(2^n)$	2^{2n}	3^n	$n \log n$
$100n^{2/3}$	$\log[(\log n)^2]$	$n!$	$(n - 1)!$

Constant in exponent does not matter when the base is also a constant, in this case, 2.

$$2^{n+1} = 2 \cdot 2^n = O(2^n)$$

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5} = O(n^{2.5})$
$n^{0.00000001} = O(n^{0.00000001})$	$\log n! = O(n \log n)$	$n^n = O(n^n)$	$2^n = O(2^n)$
$2^{n+1} = O(2^n)$	$2^{2n} = O(2^{2n}) = O(4^n)$	3^n	$n \log n$
$100n^{2/3}$	$\log[(\log n)^2]$	$n!$	$(n-1)!$

Coefficient in exponent matters.

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5} = O(n^{2.5})$
$n^{0.00000001} = O(n^{0.00000001})$	$\log n! = O(n \log n)$	$n^n = O(n^n)$	$2^n = O(2^n)$
$2^{n+1} = O(2^n)$	$2^{2n} = O(2^{2n}) = O(4^n)$	$3^n = O(3^n)$	$n \log n$
$100n^{2/3}$	$\log[(\log n)^2]$	$n!$	$(n - 1)!$

Variable in exponent also matters.

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5} = O(n^{2.5})$
$n^{0.00000001} = O(n^{0.00000001})$	$\log n! = O(n \log n)$	$n^n = O(n^n)$	$2^n = O(2^n)$
$2^{n+1} = O(2^n)$	$2^{2n} = O(2^{2n}) = O(4^n)$	$3^n = O(3^n)$	$n \log n = O(n \log n)$
$100n^{2/3}$	$\log[(\log n)^2]$	$n!$	$(n - 1)!$

Nothing to simplify here :)

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5} = O(n^{2.5})$
$n^{0.00000001} = O(n^{0.00000001})$	$\log n! = O(n \log n)$	$n^n = O(n^n)$	$2^n = O(2^n)$
$2^{n+1} = O(2^n)$	$2^{2n} = O(2^{2n}) = O(4^n)$	$3^n = O(3^n)$	$n \log n = O(n \log n)$
$100n^{2/3} = O(n^{2/3})$	$\log[(\log n)^2]$	$n!$	$(n - 1)!$

Drop the coefficient, and constant in exponent with variable base matters.

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5} = O(n^{2.5})$
$n^{0.00000001} = O(n^{0.00000001})$	$\log n! = O(n \log n)$	$n^n = O(n^n)$	$2^n = O(2^n)$
$2^{n+1} = O(2^n)$	$2^{2n} = O(2^{2n}) = O(4^n)$	$3^n = O(3^n)$	$n \log n = O(n \log n)$
$100n^{2/3} = O(n^{2/3})$	$\log[(\log n)^2] = O(\log \log n)$	$n!$	$(n - 1)!$

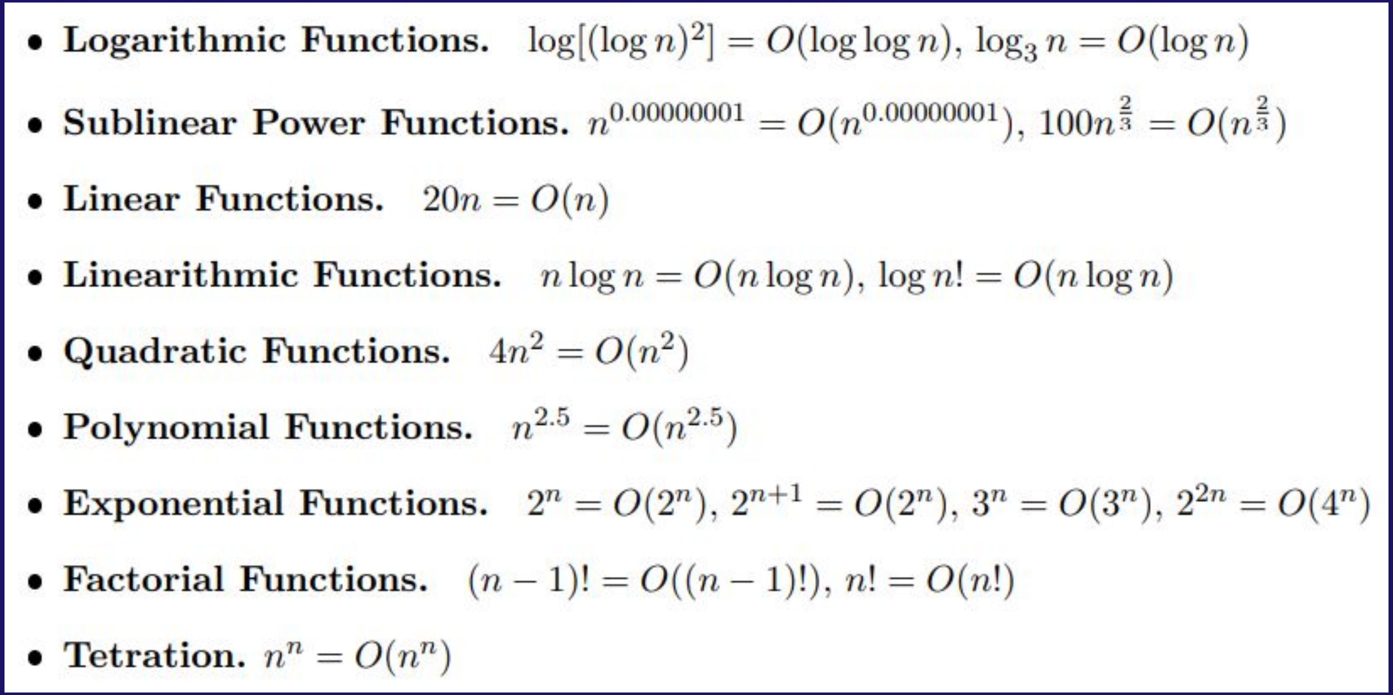
$$\log[(\log n)^2] = 2\log[\log n] = O(\log \log n)$$

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5} = O(n^{2.5})$
$n^{0.00000001} = O(n^{0.00000001})$	$\log n! = O(n \log n)$	$n^n = O(n^n)$	$2^n = O(2^n)$
$2^{n+1} = O(2^n)$	$2^{2n} = O(2^{2n}) = O(4^n)$	$3^n = O(3^n)$	$n \log n = O(n \log n)$
$100n^{2/3} = O(n^{2/3})$	$\log[(\log n)^2] = O(\log \log n)$	$n! = O(n!)$	$(n-1)! = O((n-1)!)$

Why not $O(n!)$ also? Because they differ by a factor of n, which is TOO BIG.

Big-O Complexity



- **Logarithmic Functions.** $\log[(\log n)^2] = O(\log\log n)$, $\log_3 n = O(\log n)$
- **Sublinear Power Functions.** $n^{0.00000001} = O(n^{0.00000001})$, $100n^{\frac{2}{3}} = O(n^{\frac{2}{3}})$
- **Linear Functions.** $20n = O(n)$
- **Linearithmic Functions.** $n\log n = O(n\log n)$, $\log n! = O(n\log n)$
- **Quadratic Functions.** $4n^2 = O(n^2)$
- **Polynomial Functions.** $n^{2.5} = O(n^{2.5})$
- **Exponential Functions.** $2^n = O(2^n)$, $2^{n+1} = O(2^n)$, $3^n = O(3^n)$, $2^{2n} = O(4^n)$
- **Factorial Functions.** $(n-1)! = O((n-1)!)$, $n! = O(n!)$
- **Tetration.** $n^n = O(n^n)$

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5} = O(n^{2.5})$
$n^{0.00000001} = O(n^{0.00000001})$	$\log n! = O(n \log n)$	$n^n = O(n^n)$	$2^n = O(2^n)$
$2^{n+1} = O(2^n)$	$2^{2n} = O(2^{2n}) = O(4^n)$	$3^n = O(3^n)$	$n \log n = O(n \log n)$
$100n^{2/3} = O(n^{2/3})$	$\log[(\log n)^2] = O(\log \log n)$	$n! = O(n!)$	$(n - 1)! = O((n - 1)!)$

Logarithmic, sublinear, linear, linearithmic, quadratic, polynomial, exponential, factorial, tetration

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5} = O(n^{2.5})$
$n^{0.00000001} = O(n^{0.00000001})$	$\log n! = O(n \log n)$	$n^n = O(n^n)$	$2^n = O(2^n)$
$2^{n+1} = O(2^n)$	$2^{2n} = O(2^{2n}) = O(4^n)$	$3^n = O(3^n)$	$n \log n = O(n \log n)$
$100n^{2/3} = O(n^{2/3})$	$\log[(\log n)^2] = O(\log \log n)$	$n! = O(n!)$	$(n - 1)! = O((n - 1)!)$

Logarithmic, sublinear, linear, linearithmic, quadratic, polynomial, exponential, factorial, tetration

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5} = O(n^{2.5})$
$n^{0.00000001} = O(n^{0.00000001})$	$\log n! = O(n \log n)$	$n^n = O(n^n)$	$2^n = O(2^n)$
$2^{n+1} = O(2^n)$	$2^{2n} = O(2^{2n}) = O(4^n)$	$3^n = O(3^n)$	$n \log n = O(n \log n)$
$100n^{2/3} = O(n^{2/3})$	$\log[(\log n)^2] = O(\log \log n)$	$n! = O(n!)$	$(n - 1)! = O((n - 1)!)$

Logarithmic, sublinear, linear, linearithmic, quadratic, polynomial, exponential, factorial, tetration

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5} = O(n^{2.5})$
$n^{0.00000001} = O(n^{0.00000001})$	$\log n! = O(n \log n)$	$n^n = O(n^n)$	$2^n = O(2^n)$
$2^{n+1} = O(2^n)$	$2^{2n} = O(2^{2n}) = O(4^n)$	$3^n = O(3^n)$	$n \log n = O(n \log n)$
$100n^{2/3} = O(n^{2/3})$	$\log[(\log n)^2] = O(\log \log n)$	$n! = O(n!)$	$(n - 1)! = O((n - 1)!)$

Logarithmic, sublinear, linear, linearithmic, quadratic, polynomial, exponential, factorial, tetration

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5} = O(n^{2.5})$
$n^{0.00000001} = O(n^{0.00000001})$	$\log n! = O(n \log n)$	$n^n = O(n^n)$	$2^n = O(2^n)$
$2^{n+1} = O(2^n)$	$2^{2n} = O(2^{2n}) = O(4^n)$	$3^n = O(3^n)$	$n \log n = O(n \log n)$
$100n^{2/3} = O(n^{2/3})$	$\log[(\log n)^2] = O(\log \log n)$	$n! = O(n!)$	$(n-1)! = O((n-1)!)$

Logarithmic, sublinear, linear, linearithmic, quadratic, polynomial, exponential, factorial, tetration

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5} = O(n^{2.5})$
$n^{0.00000001} = O(n^{0.00000001})$	$\log n! = O(n \log n)$	$n^n = O(n^n)$	$2^n = O(2^n)$
$2^{n+1} = O(2^n)$	$2^{2n} = O(2^{2n}) = O(4^n)$	$3^n = O(3^n)$	$n \log n = O(n \log n)$
$100n^{2/3} = O(n^{2/3})$	$\log[(\log n)^2] = O(\log \log n)$	$n! = O(n!)$	$(n - 1)! = O((n - 1)!)$

Logarithmic, sublinear, linear, linearithmic, quadratic, polynomial, exponential, factorial, tetration

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5} = O(n^{2.5})$
$n^{0.00000001} = O(n^{0.00000001})$	$\log n! = O(n \log n)$	$n^n = O(n^n)$	$2^n = O(2^n)$
$2^{n+1} = O(2^n)$	$2^{2n} = O(2^{2n}) = O(4^n)$	$3^n = O(3^n)$	$n \log n = O(n \log n)$
$100n^{2/3} = O(n^{2/3})$	$\log[(\log n)^2] = O(\log \log n)$	$n! = O(n!)$	$(n - 1)! = O((n - 1)!)$

Logarithmic, sublinear, linear, linearithmic, quadratic, polynomial, exponential, factorial, tetration

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5} = O(n^{2.5})$
$n^{0.00000001} = O(n^{0.00000001})$	$\log n! = O(n \log n)$	$n^n = O(n^n)$	$2^n = O(2^n)$
$2^{n+1} = O(2^n)$	$2^{2n} = O(2^{2n}) = O(4^n)$	$3^n = O(3^n)$	$n \log n = O(n \log n)$
$100n^{2/3} = O(n^{2/3})$	$\log[(\log n)^2] = O(\log \log n)$	$n! = O(n!)$	$(n - 1)! = O((n - 1)!)$

Logarithmic, sublinear, linear, linearithmic, quadratic, polynomial, exponential, factorial, **tetration**

Big-O Complexity

$4n^2 = O(n^2)$	$\log_3 n = O(\log n)$	$20n = O(n)$	$n^{2.5} = O(n^{2.5})$
$n^{0.00000001} = O(n^{0.00000001})$	$\log n! = O(n \log n)$	$n^n = O(n^n)$	$2^n = O(2^n)$
$2^{n+1} = O(2^n)$	$2^{2n} = O(2^{2n}) = O(4^n)$	$3^n = O(3^n)$	$n \log n = O(n \log n)$
$100n^{2/3} = O(n^{2/3})$	$\log[(\log n)^2] = O(\log \log n)$	$n! = O(n!)$	$(n - 1)! = O((n - 1)!)$

$$\log[(\log n)^2] \prec \log_3 n \prec n^{0.00000001} \prec 100n^{\frac{2}{3}} \prec 20n \prec n \log n \equiv \log n! \prec 4n^2 \prec n^{2.5} \prec 2^n \equiv 2^{n+1} \prec 3^n \prec 2^{2n} \prec (n-1)! \prec n! \prec n^n$$

02

TIME COMPLEXITY ANALYSIS

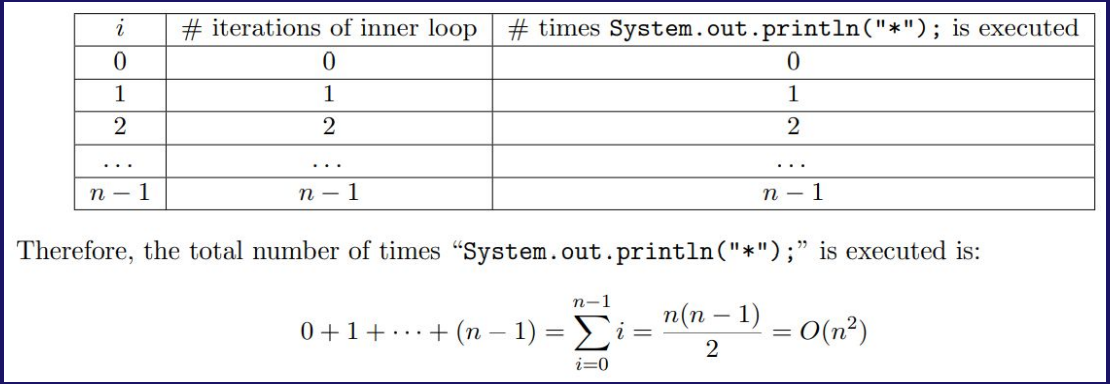
Time Complexity Analysis

Non-Recursive
Find the number of times each statement is executed, and then simplify using the rules mentioned earlier.

Recursive
Draw a recursion tree.

Problem 2a

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        System.out.println("*");
    }
}
```



i	# iterations of inner loop	# times `System.out.println("*");` is executed
0	0	0
1	1	1
2	2	2
$\cdots$	$\cdots$	$\cdots$
$n-1$	$n-1$	$n-1$

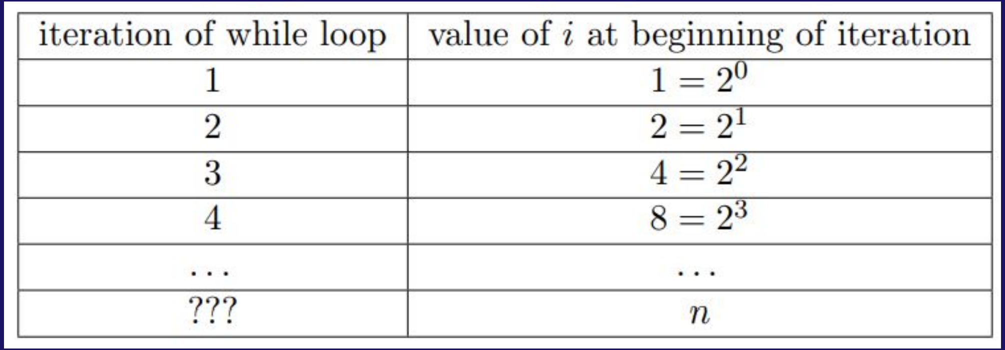
Therefore, the total number of times "`System.out.println("*");`" is executed is:

$$0 + 1 + \cdots + (n-1) = \sum_{i=0}^{n-1} i = \frac{n(n-1)}{2} = O(n^2)$$

Each execution of System.out.println("*"); runs in O(1) time. Hence, the code fragment runs in $O(n^2)$ time.

Problem 2b

```
int i = 1;
while (i <= n) {
    System.out.println("*");
    i = 2 * i;
}
```



iteration of while loop	value of i at beginning of iteration
1	$1 = 2^0$
2	$2 = 2^1$
3	$4 = 2^2$
4	$8 = 2^3$
$\cdots$	$\cdots$
???	n

In the kth iteration, the value of i is 2^{k-1}. Iteration stops when $i = 2^{k-1} > n$.

Problem 2b

$$\begin{aligned} 2^{k-1} &> n \\ \log_2 2^{k-1} &> \log_2 n \\ k - 1 &> \log_2 n \\ k &> 1 + \log_2 n = O(\log n) \end{aligned}$$

While loop terminates after O(log n) iterations. Since the print and multiplication runs in O(1) time, the code fragment runs in O(log n) time.

Problem 2c

```
int i = n;
while (i > 0) {
    for (int j = 0; j < n; j++)
        System.out.println("*");
    i = i / 2;
}
```

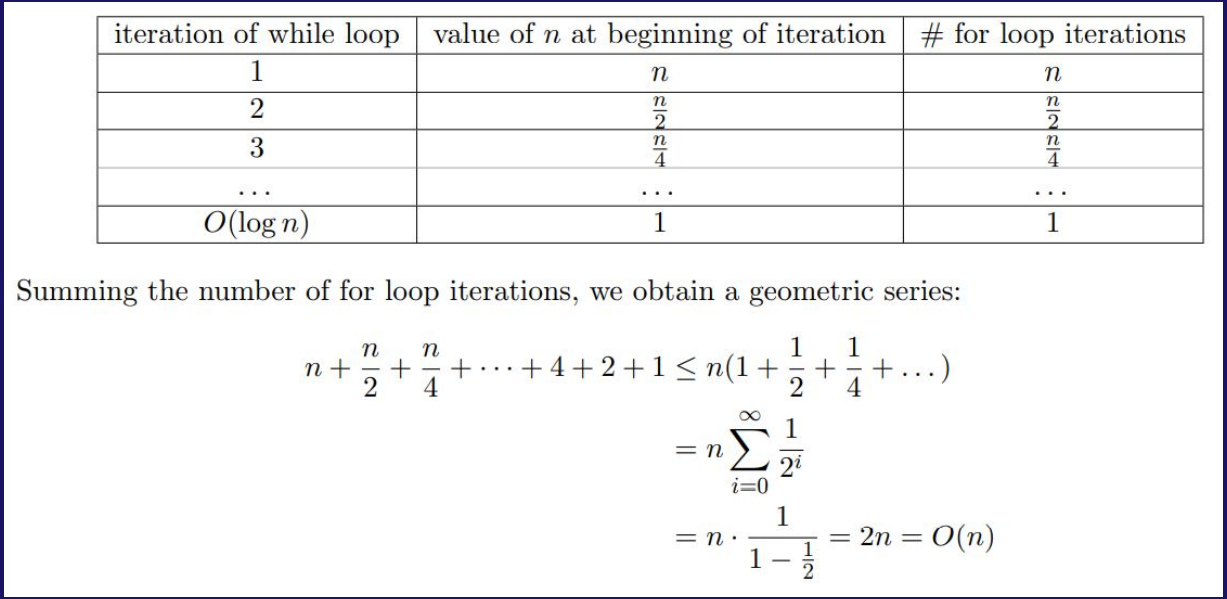
Same like problem 2b, the while loop terminates after O(log n) iterations.

In every iteration, we have inner for loop that runs O(n) time. Value of n does not change, so the number of iterations the inner loop runs is independent of the outer loop.

Therefore, the total number of statements executed can be taken by multiplying both values together. Therefore, the time complexity is O(n log n).

Problem 2d

```
while (n > 0) {
    for (int j = 0; j < n; j++)
        System.out.println("*");
    n = n / 2;
}
```



iteration of while loop	value of n at beginning of iteration	# for loop iterations
1	n	n
2	$\frac{n}{2}$	$\frac{n}{2}$
3	$\frac{n}{4}$	$\frac{n}{4}$
...	...	...
$O(\log n)$	1	1

Summing the number of for loop iterations, we obtain a geometric series:

$$
\begin{aligned}
n + \frac{n}{2} + \frac{n}{4} + \cdots + 4 + 2 + 1 &\leq n(1 + \frac{1}{2} + \frac{1}{4} + \dots) \\
&= n \sum_{i=0}^{\infty} \frac{1}{2^i} \\
&= n \cdot \frac{1}{1 - \frac{1}{2}} = 2n = O(n)
\end{aligned}
$$

Problem 2e

```
String x; // String x has length n
String y; // String y has length m
String z = x + y;
System.out.println(z);
```

The answer is $O(n + m)$.

A common misconception: two strings of variable length does not take O(1) time (in Java API).

Problem 2e

String Concatenation & Time Complexity

If you have a string and add a letter to its back *n* times, the time complexity is O(n^2)! That's because strings are immutable and you'll create a new copy each time. At each concatenation, the time complexity is on average O(n), so that is too slow.

Here's some ways to do it in Python or Java:

PYTHON

Store all the letters you want into a list.
lst = ["g", "h", "i", "k", "a", "b" …]
Join them all at once using "".**join(lst)** in O(n) time!

JAVA

Use the StringBuilder class.

```
StringBuilder sb = new StringBuilder();
sb.append("g");  sb.append("h");  sb.append("i");  …  → this is O(1) time!
System.out.println(sb.toString())  // print out the entire string in one go!
```

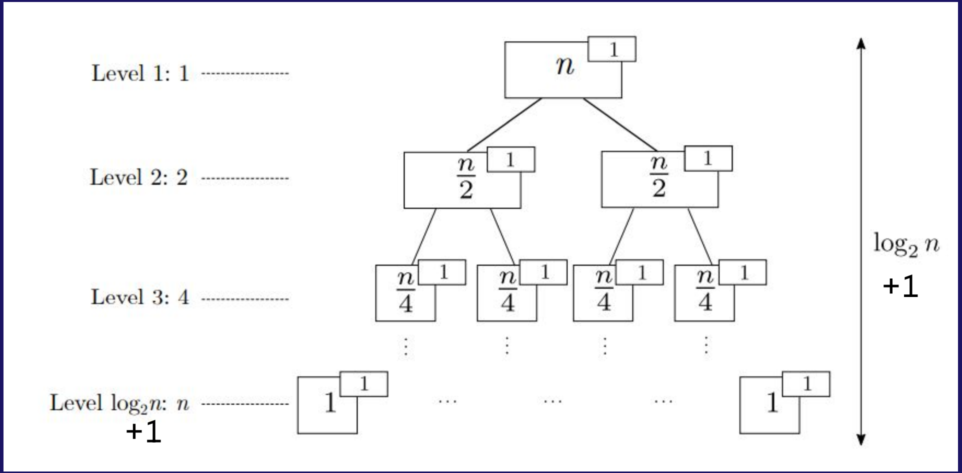
Problem 2f

```
void foo(int n){
    if (n <= 1)
        return;
    System.out.println("*");
    foo(n/2);
    foo(n/2);
}
```

$$1 + 2 + 4 + 8 + \ldots + n = \sum_{i=0}^{\log_2 n} 2^i$$

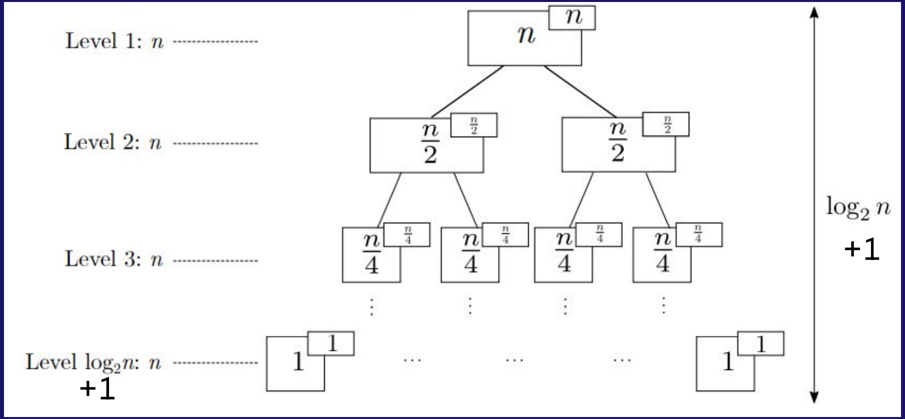
$$= \frac{2^{\log_2 n+1} - 1}{2 - 1}$$

$$= 2n - 1 = O(n)$$



Problem 2g

```
void foo(int n){
    if (n <= 1)
        return;
    for (int i = 0; i < n; i++) {
        System.out.println("*");
    }
    foo(n/2);
    foo(n/2);
}
```



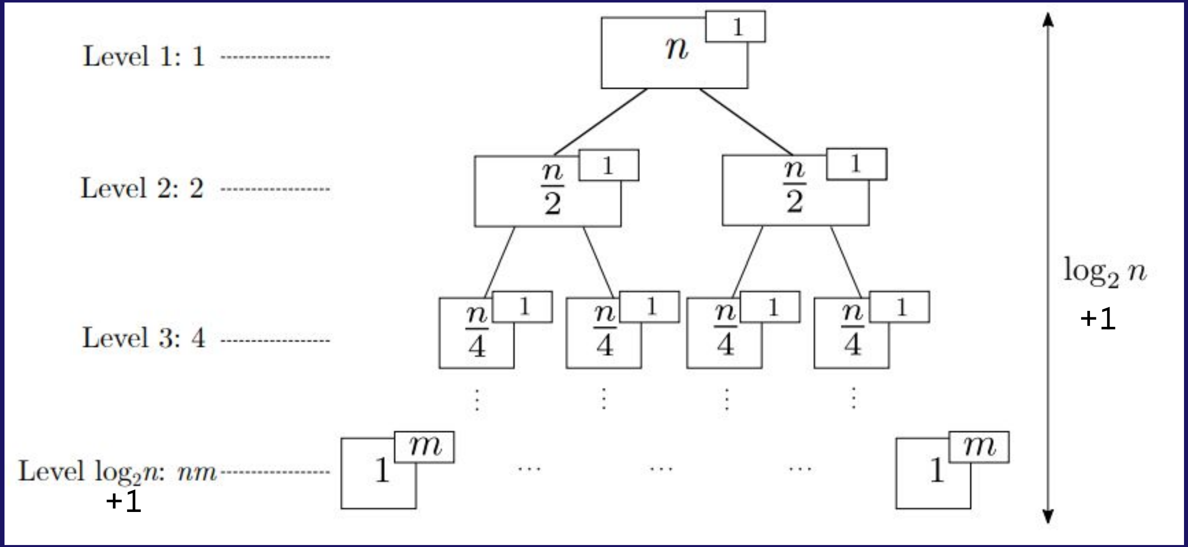
There are 2^{i-1} nodes on the ith level, so the total work done on the ith level is $2^{i-1} \cdot n/2^{i-1} = n$.

As there are $O(\log_2 n)$ levels in the recursion tree, the total work done across all levels is $n \cdot O(\log_2 n) = O(n \log n)$.

Problem 2h

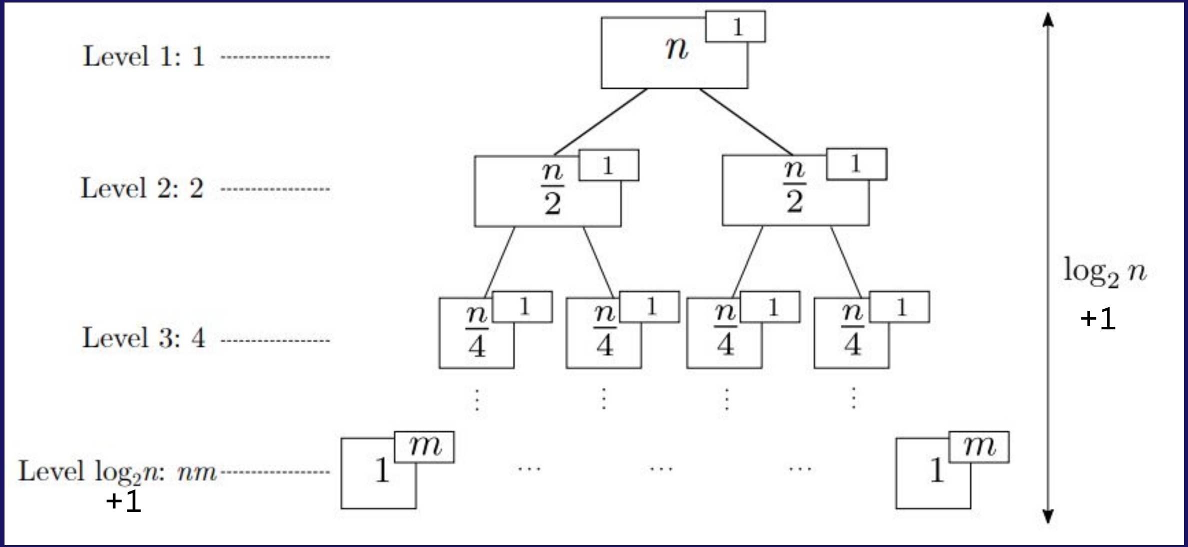
```
void foo(int n, int m){
    if (n <= 1) {
        for (int i = 0; i < m; i++) {
            System.out.println("*");
        }
        return;
    }
    foo(n/2, m);
    foo(n/2, m);
}
```

Very similar to Problem 2f.



Problem 2h

$$1 + 2 + 4 + \ldots + 2^{\log_2 n - 1} + m \cdot 2^{\log_2 n} = \frac{2^{\log_2 n} - 1}{2 - 1} + mn$$
$$= n - 1 + mn = O(mn)$$



03

PRACTICE MAKES PERFECT?

More practice!

```
void foo(int n) {
    if (n <= 1)
        return;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            System.out.println("*");
        }
    }
    foo(n/2);
    foo(n/2);
}
```

What’s the runtime of the method above? $O(n^2)$

$$f_1(n) = 7.2 + 34n^3 + 3254n$$

$$f_2(n) = n^2 \log n + 25n \log^2 n$$

$$f_3(n) = 2^{4\log_2 n} + 5n^5$$

$$f_4(n) = 2^{2n^2+4n+7}$$

$$f_5(n) = 1/n$$

$$f_6(n) = \log_4(n) + \log_8(n)$$

$$f_7(n) = \log\log\log n \ + \ \log\log(n^4)$$

$$f_8(n) = (1 - 4/n)^{2n}$$

$$f_9(n) = \log(\sqrt{n}) + \sqrt{\log(n)}$$

$$f_1(n) = 7.2 + 34n^3 + 3254n = O(n^3)$$

$$f_2(n) = n^2 \log n + 25n \log^2 n = O(n^2 \log n)$$

$$f_3(n) = 2^{4 \log_2 n} + 5n^5 = O(n^5)$$

$$f_4(n) = 2^{2n^2+4n+7} = O\left(2^{2n^2+4n}\right) = O\left(4^{n^2+2n}\right)$$

$$f_5(n) = 1/n = O(1)$$

$$f_6(n) = \log_4(n) + \log_8(n) = O(\log n)$$

$$f_7(n) = \log\log\log n + \log\log(n^4) = O(\log\log n)$$

$$f_8(n) = (1 - 4/n)^{2n} = O(1)$$

$$f_9(n) = \log(\sqrt{n}) + \sqrt{\log(n)} = O(\log n)$$

THE END!