CS2040S TUTORIAL 1

Lucia Tirta Gunawan
luciatirtag@u.nus.edu

WELCOME TO THE FIRST TUTORIAL!!

ABOUT ME

Lucia Tirta Gunawan

Year 3 – Computer Science

Telegram: @luciatirta

WhatsApp: +65 93593665

Email: luciatirtag@u.nus.edu

Likes:

GET TO KNOW EACH OTHER!

1. Name
2. Year, Course
3. Hobby/random facts

JOIN TELEGRAM GROUP T03

https://t.me/joinchat/OCUneEtRmxNiMDA1

ADMIN STUFF

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%

LET'S START...

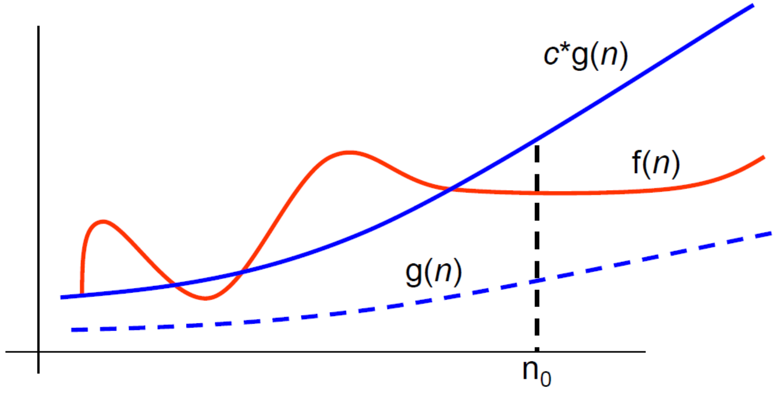
Q1. BIG O COMPLEXITY

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 $n \geq n_0$.

For example:
$5n^2 = O(n^2)$
$c = 6$, $n_0 = 1$
$5n^2 \leq cn^2$ 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.



Q1. BIG O COMPLEXITY (CONT.)

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

Q1. BIG O COMPLEXITY (CONT.)

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

Just drop the coefficient :D

Q1. BIG O COMPLEXITY (CONT.)

$4n^2 = O(n^2)$	**$\log_3 n = O(\log n)$**	**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)!

Constant base in logarithm does not matter. Why?

Because if the base is constant, we can change it to another constant

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

Q1. BIG O COMPLEXITY (CONT.)

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

Just drop the coefficient again :D

Q1. BIG O COMPLEXITY (CONT.)

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

DON'T drop the constant!! Constant in exponent MATTERS (when base is a variable, in this case, n)

Q1. BIG O COMPLEXITY (CONT.)

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

DON'T drop the constant!! Constant in exponent MATTERS (when base is a variable, in this case, n)

Q1. BIG O COMPLEXITY (CONT.)

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

$\log ab = \log a + \log b$

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

Q1. BIG O COMPLEXITY (CONT.)

$4n^2 = O(n^2)$	**$\log_3 n = O(\log n)$**	**$20n = O(n)$**	**$n^{2.5} = O(n^{2.5})$**
$n^{0.00000001} = O(n^{0.00000001})$	$\log(n!) = O(n \log n)$	$n^n = O(n^n)$	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

Q1. BIG O COMPLEXITY (CONT.)

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

Variable in exponent matters!

Q1. BIG O COMPLEXITY (CONT.)

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

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

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

Q1. BIG O COMPLEXITY (CONT.)

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

Coefficient in exponent matters!

Q1. BIG O COMPLEXITY (CONT.)

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

Variable in exponent matters!

Q1. BIG O COMPLEXITY (CONT.)

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

Quite obvious :D

Q1. BIG O COMPLEXITY (CONT.)

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

Coefficient can be dropped. Constant in exponent with variable base (in this case n) matters

Q1. BIG O COMPLEXITY (CONT.)

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

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

Q1. BIG O COMPLEXITY (CONT.)

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

Quite obvious :D

Q1. BIG O COMPLEXITY (CONT.)

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

Why not O(n!)??

If $(n-1)! = O(n!)$ can I say $n = O(n^2)$??

NOPE, they differ by a factor of n, TOO BIG, Not a tight bound

Q1. BIG O COMPLEXITY (CONT.)

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

- **Logarithmic 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.** $n^n = O(n^n)$

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

Q2. TIME COMPLEXITY ANALYSIS

(a)

```
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
...	...	...
$n-1$	$n-1$	$n-1$

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

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

Answer: $O(n^2)$

Each execution of System.out.println("*"); runs in O(1) time

Hence, the code fragment runs in $O(n^2)$ time.

Q2. TIME COMPLEXITY ANALYSIS (CONT.)

(b)
```
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	$8 = 2^3$
...	...
???	n

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

Answer: O(log n)

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

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

Q2. TIME COMPLEXITY ANALYSIS (CONT.)

(c)

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

Answer: O(n log n)

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)

Q2. TIME COMPLEXITY ANALYSIS (CONT.)

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

Answer: O(n)

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	$\frac{n}{4}$	$\frac{n}{4}$
$\dots$	$\dots$	$\dots$
$O(\log n)$	1	1

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

Q2. TIME COMPLEXITY ANALYSIS (CONT.)

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

Answer: O(n+m)

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

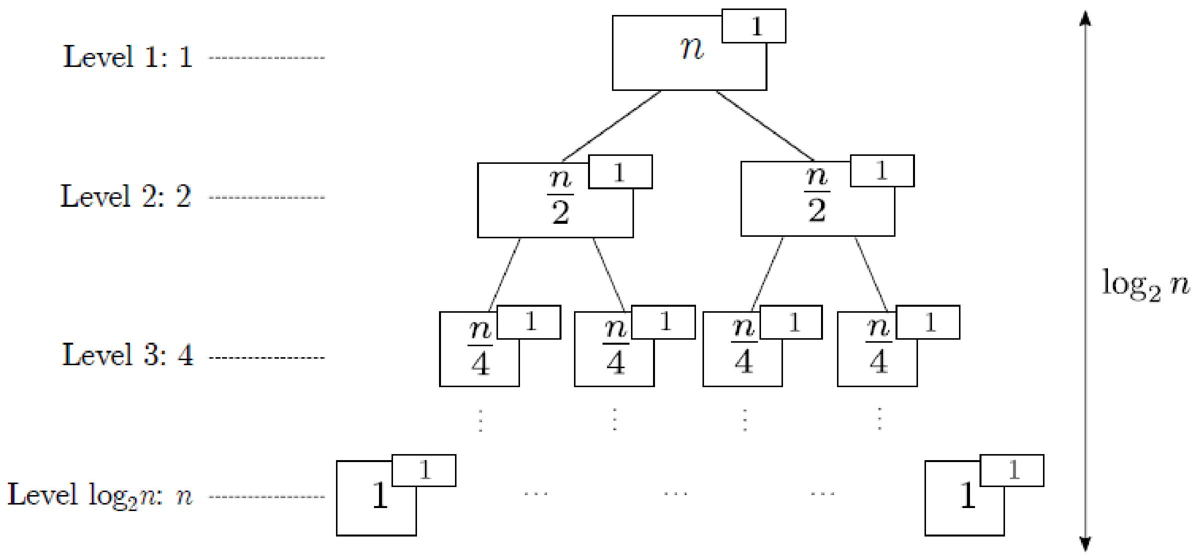
Q2. TIME COMPLEXITY ANALYSIS (CONT.)

(f)

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

$$
\begin{aligned}
1 + 2 + 4 + \cdots + n &= \sum_{i=0}^{\log_2 n} 2^i \\
&= \frac{1 - 2^{\log_2 n + 1}}{1 - 2} \\
&= 2n - 1 \\
&= O(n)
\end{aligned}
$$

Answer: O(n)

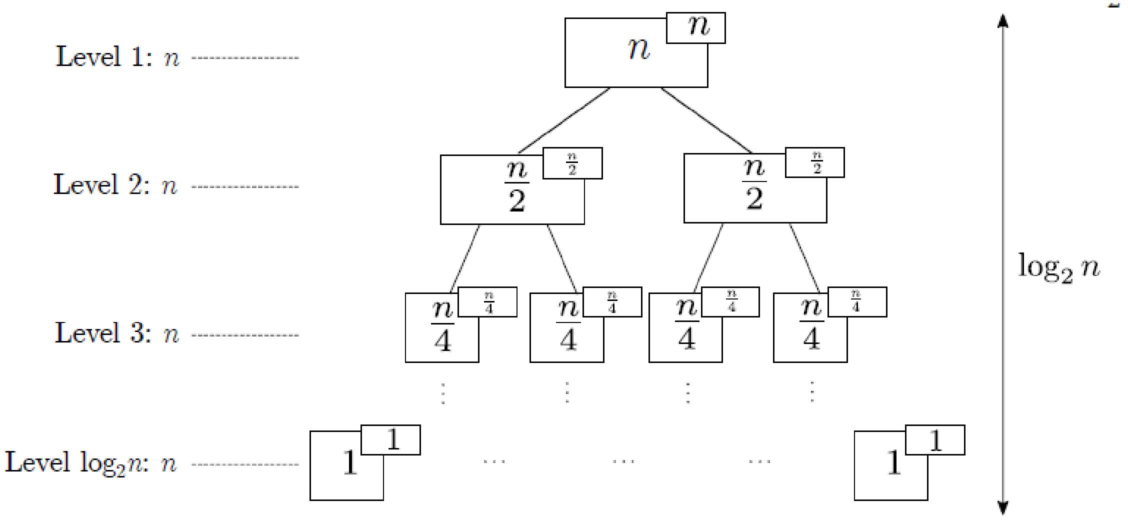


Q2. TIME COMPLEXITY ANALYSIS (CONT.)

(g)

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

Answer: O(n log n)



In each level, the total work done is n. In level i, there are 2^{i-1} nodes, the total work done is $2^{i-1} * (n/2^{i-1}) = n$

There is log n levels

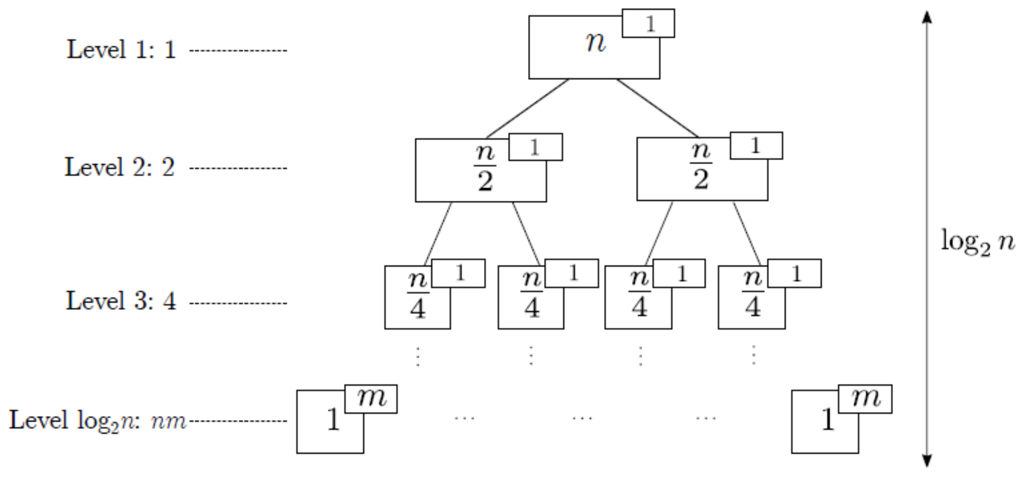
Total work across all levels is O(n log n)

Q2. TIME COMPLEXITY ANALYSIS (CONT.)

(h)

```
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);
}
```

Answer: O(nm)



$$
\begin{aligned}
1 + 2 + \cdots + 2^{\log_2 n - 1} + m \cdot 2^{\log_2 n} &= \sum_{i=0}^{\log_2 n - 1} 2^i + nm \\
&= \frac{1 - 2^{\log_2 n}}{1 - 2} + nm \\
&= n - 1 + nm \\
&= O(nm)
\end{aligned}
$$

ADDITIONAL NOTES

Master Theorem

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

T(n) $= O(n^k)$ if $a < b^k$

$= O(n^k \log n)$ if $a = b^k$

$= O\left(n^{\log_b a}\right)$ if $a > b^k$

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

In the above example, $a = 2$,
$b = 2$, $k = 0$
$a > b^k$
So the complexity is $O(n)$

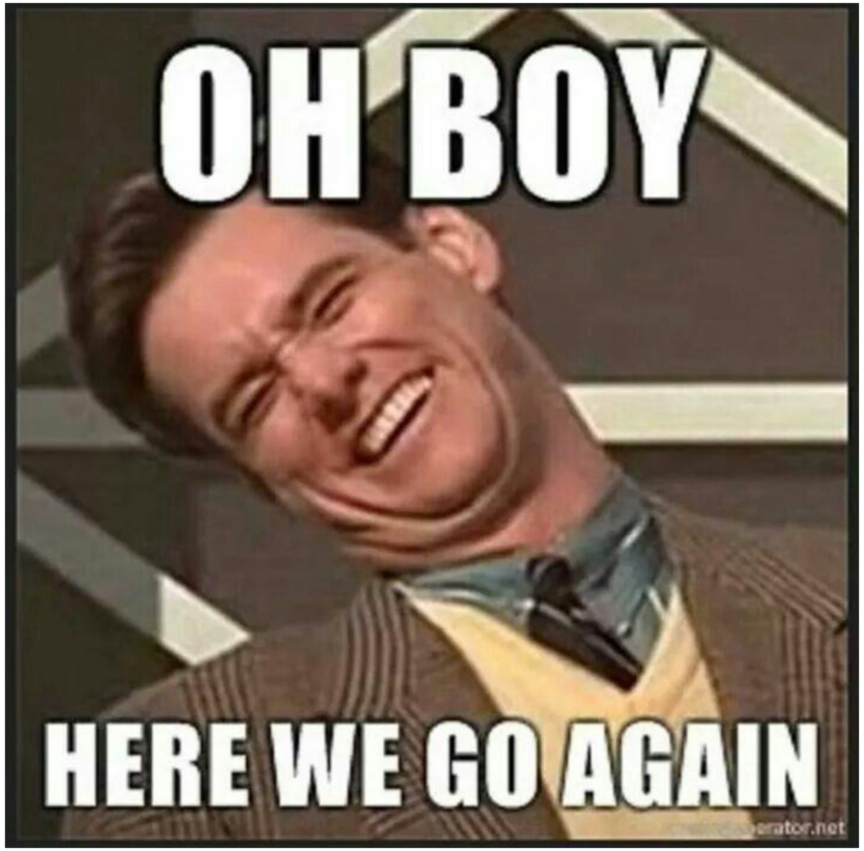
ADDITIONAL NOTES

Prove big O with limit:

$$f(n), g(n) > 0$$

$$\lim_{n \to \infty} \left(\frac{f(n)}{g(n)} \right) < \infty \to f(n) = O(g(n))$$

MORE EXERCISE???



EXTRA PRACTICE

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

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

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

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

$f_5(n) = 1/n$

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

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

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

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

EXTRA PRACTICE (ANS)

$f_1(n) = O(n^3)$

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

$f_3(n) = O(n^5)$

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

$f_5(n) = O(1)$

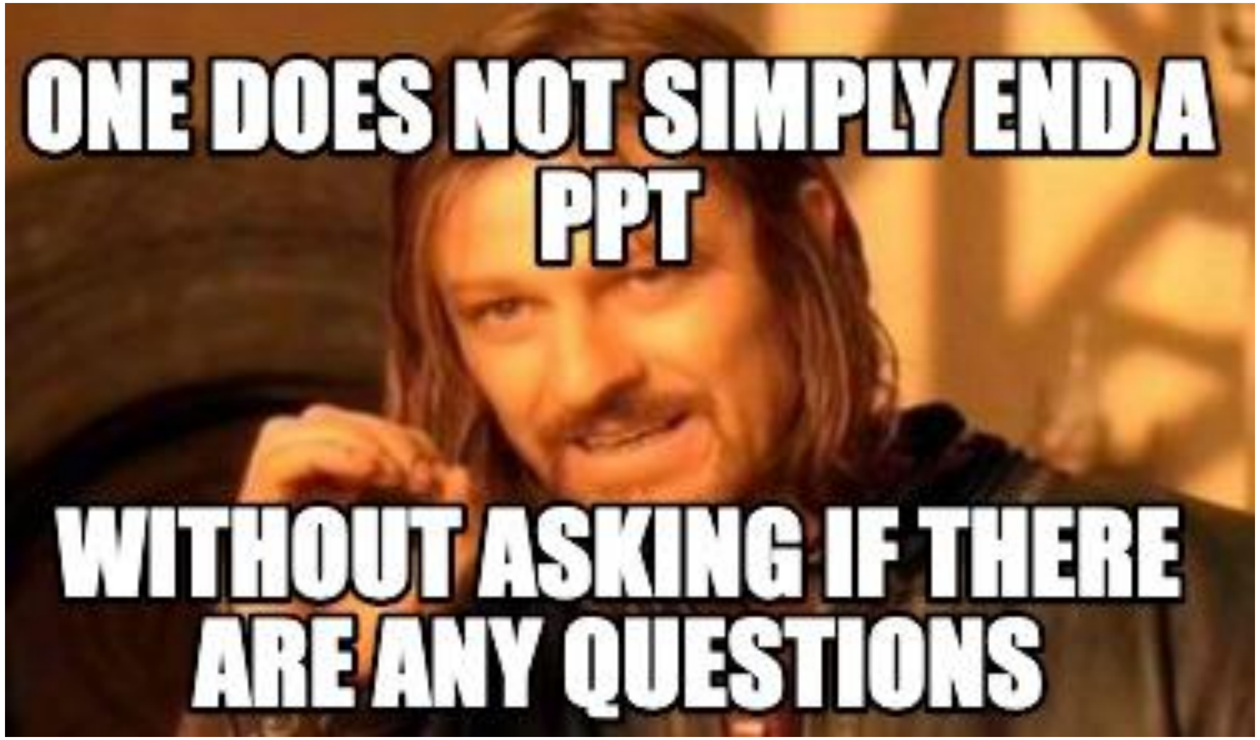
$f_6(n) = O(\log n)$

$f_7(n) = \log \log n$

$f_8(n) = O(1)$

$f_9(n) = O(\log n)$

ANY QUESTION?



SEE YOU IN THE NEXT TUTORIAL!