Tutorial 1: Asymptotic Analysis

August 22, 2022

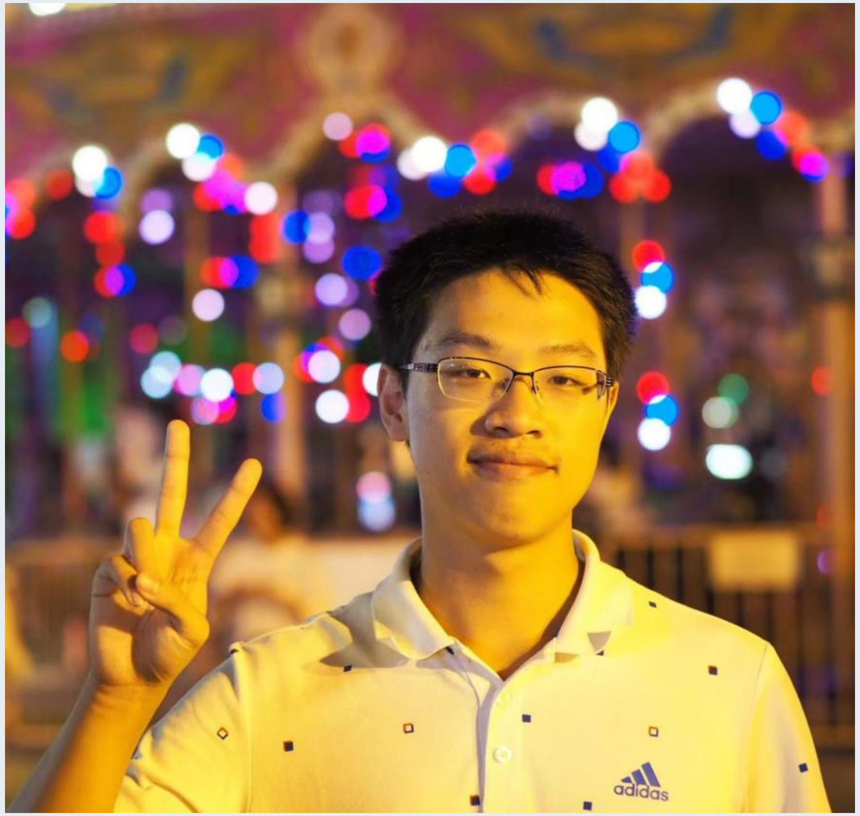
Gu Zhenhao

** Partly adopted from tutorial slides by Wang Zhi Jian.*

About Me

Hello, I am Gu Zhenhao (Gary)!

- Year 2, MComp (CS) candidate,
- **Interests**: Algorithm & theory research, Computational Biology, open-world/MOBA games,
- **E-mail**: zhenhao.gu@u.nus.edu
- **Telegram**: @garygzh
- **GitHub**: GZHoffie



About this Tutorial

- **Time**: 10:00 – 11:00 A.M. every Monday.
- **Zoom Link**: See Canvas → Zoom or Telegram group.
- **Content**: Review key concepts in lectures and discuss problem-solving recipes.

About this Tutorial

- **Slides**: will be uploaded to Telegram group and my repository.
- **Notes**:
  1. Attendance & participation will be taken. (3% of course grade)
  2. Try to think of the tutorial problems ahead of time.

Slides with star *

Slides with star * contain additional content that are **not required in CS2040S**, e.g.

1. More complicated exercises,
2. Hints to understand the concepts,
3. More advanced topics.

Big-O Notation

How to describe the efficiency of an algorithm?

Big-O Notation

Purpose: to measure the worst possible performance of an algorithm given <u>large input size</u> n.

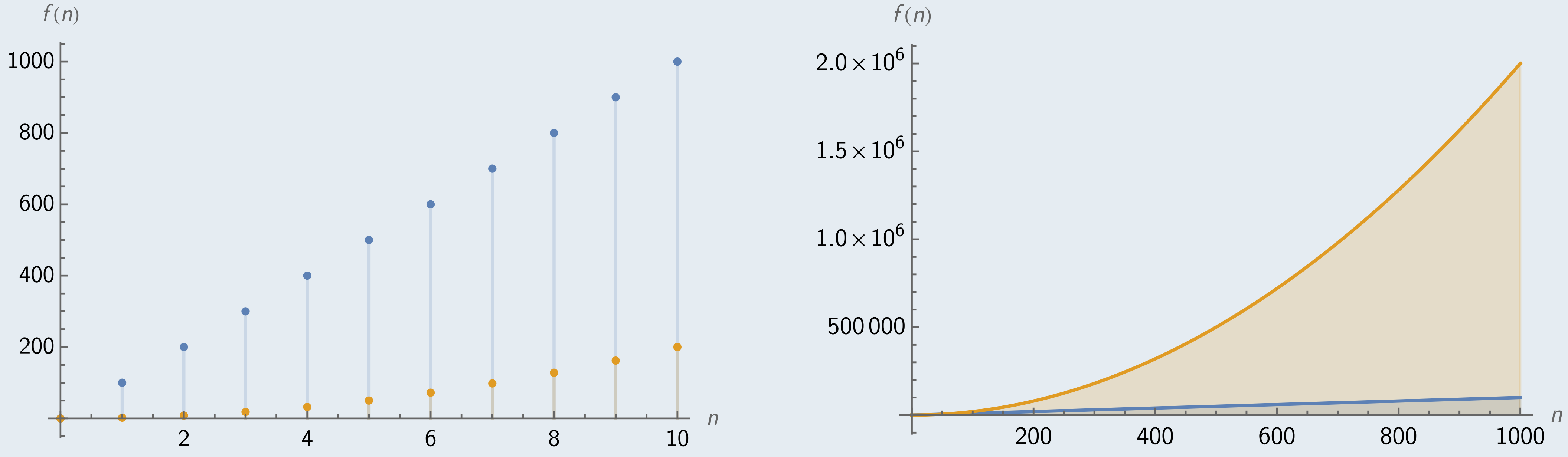


Figure: comparison of $f(n) = 100n$ (blue) and $f(n) = 2n^2$ (orange)

Big-O Notation

Recipe 1: Suppose we have running time $f(n)$, express it using big-O notation,

1. Ignore constant coefficients,
   e.g. $2n^2 + 100n \equiv n^2 + n$
2. Retain only the most dominant part.
   e.g. $n^2 + n = O(n^2)$

Big-O Notation

After simplifying all with big-O notation, we can compare them,

$$\underset{\text{Constant}}{O(1)} \prec \underset{\text{Logarithmic}}{O(\log n)} \prec \underset{\text{Fractional Power}}{O(n^{\overset{0<c<1}{c}})} \prec \underset{\text{Linear}}{O(n)} \prec \underset{\text{Linearithmic}}{O(n \log n)} \prec \underset{\text{Polynomial}}{O(n^2) \prec O(n^3)} \prec \underset{\text{Exponential}}{O(2^n)} \prec \underset{\text{Factorial}}{O(n!)}$$

You can find a more complete list here.

What if my formula is not on the list?

$f(n) \prec g(n) \Leftrightarrow \frac{f(n)}{g(n)} \to 0$ (or $\frac{g(n)}{f(n)} \to \infty$) as n approaches infinity.

e.g. we can verify that $O(2^n) \prec O(3^n)$ because

$$\frac{2^n}{3^n} = \left(\frac{2}{3}\right)^n \xrightarrow{n\to\infty} 0.$$

Problem 1

Rearrange the following functions in ascending order:

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

To compare them, use big-O notation!

Problem 1

Find the tightest big-O complexity:

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

$$4n^2 \equiv n^2 = O(n^2)$$

Problem 1

Find the tightest big-O complexity:

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

$$\log_3 n = \frac{\log n}{\log 3} \equiv \log n = O(\log n)$$

Note: $\log_b a = \log_k a \,/\, \log_k b$ (change of base rule)

Pause and Ponder 1

Based on change of base rule we can ignore the base in logarithm factor, e.g. $\log_k n = O(\log n)$

Question 1: can we write $n^{\log_a b} = O(n^{\log b})$?

Question 2: can we write $\log_b n = O(\log n)$ if b is some <u>variable</u> dependent on n?

** Find the answer in the appendix!*

Problem 1

Find the tightest big-O complexity:

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

Problem 1

Find the tightest big-O complexity:

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

Problem 1

Find the tightest big-O complexity:

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

Problem 1

Find the tightest big-O complexity:

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

$$\log n! = \log(n \times (n-1) \times \cdots \times 2 \times 1) = \log n + \log(n-1) + \cdots + \log 1$$
$$< \log n + \log n + \cdots + \log n = n \log n = O(n \log n)$$

Note: $\log ab = \log a + \log b$, $\log(a/b) = \log a - \log b$.

Pause and Ponder 2

$\log n! = \log(n \times (n-1) \times \cdots \times 2 \times 1) = \log n + \log(n-1) + \cdots + \log 1$
$< \log n + \log n + \cdots + \log n = n \log n = O(n \log n)$

Question: How do we know $O(n \log n)$ is the tightest bound? Is $\log n! \equiv n \log n$?

** Find the answer in the appendix!*

Problem 1

Find the tightest big-O complexity:

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

Question: Which one is larger, $O(n^n)$ or $O(n!)$?

$\boldsymbol{O(n!)} \prec \boldsymbol{O(n^n)}$. We can verify that $\dfrac{n!}{n^n} = \dfrac{n \cdot (n-1) \cdot \ldots \cdot 1}{n \cdot n \cdot \ldots \cdot n} < \dfrac{1}{n} \xrightarrow{n \to \infty} 0.$

Problem 1

Find the tightest big-O complexity:

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

Problem 1

Find the tightest big-O complexity:

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

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

Problem 1

Find the tightest big-O complexity:

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

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

Problem 1

Find the tightest big-O complexity:

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

Problem 1

Find the tightest big-O complexity:

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

Problem 1

Find the tightest big-O complexity:

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

Problem 1

Find the tightest big-O complexity:

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

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

Note: $\log b^c = c \log b$

Problem 1

Find the tightest big-O complexity:

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

Problem 1

Find the tightest big-O complexity:

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

Question: Can we say $O(n!)$ is the tightest bound for $(n-1)!$?

No! We have $(n-1)! \prec n!$ Because $\dfrac{(n-1)!}{n!} = \dfrac{1}{n} \xrightarrow{n\to\infty} 0.$

Problem 1

Find the tightest big-O complexity:

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

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

Running Time Analysis

Given an algorithm, how to determine its run time?

Non-recursive Algorithms

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

Recipe 2: Given a piece of code,

1. Count how many times each line is executed,
2. Multiply with the run time of each line,
3. Express the run time using big-O notation.

Problem 2.a

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

Run $0 + 1 + 2 + \cdots + (n - 1) = n(n-1)/2 = O(n^2)$ time.

Each run costs $O(1)$ time.

In total $O(n^2)$ time.

Problem 2.b

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

Run $\overbrace{1 + 1 + \cdots + 1}^{\log n \text{ ones}} = O(\log n)$ times,

each run costs $O(1)$.

Problem 2.c

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

Run $\overbrace{n + n + \cdots + n}^{\log n \text{ ones}} = O(n \log n)$ times, each run costs $O(1)$.

Problem 2.c

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

Run $\log n$ times, each run costs $O(1)$.

Problem 2.c

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

In total, $n \log n + \log n = O(n \log n)$ time.

Problem 2.d

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

Run

$$
\begin{aligned}
&n + \frac{n}{2} + \frac{n}{4} + \cdots + 1 \\
&= 2^{\log n} + 2^{\log n - 1} + \cdots + 2^0 \\
&= 2^{\log n + 1} - 1 \text{ (geometric series)} \\
&= 2n - 1 \\
&= O(n)
\end{aligned}
$$

Times, each run costs $O(1)$.

Problem 2.e

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

Concatenating Java strings:

1. Copy string x to z,
2. Content of y is appended to z.

Run 1 time, which costs $O(m + n)$.

Recursive Algorithms

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

Recipe 3: Given a piece of code,

1. Denote times used as $T(n)$,
2. Build the recursive formula and solve it.

Problem 2.f

$T(n)$

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

1

$2T(n/2)$

Recurrence relation:

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

Problem 2.f

$T(n)$

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

1

$2T(n/2)$

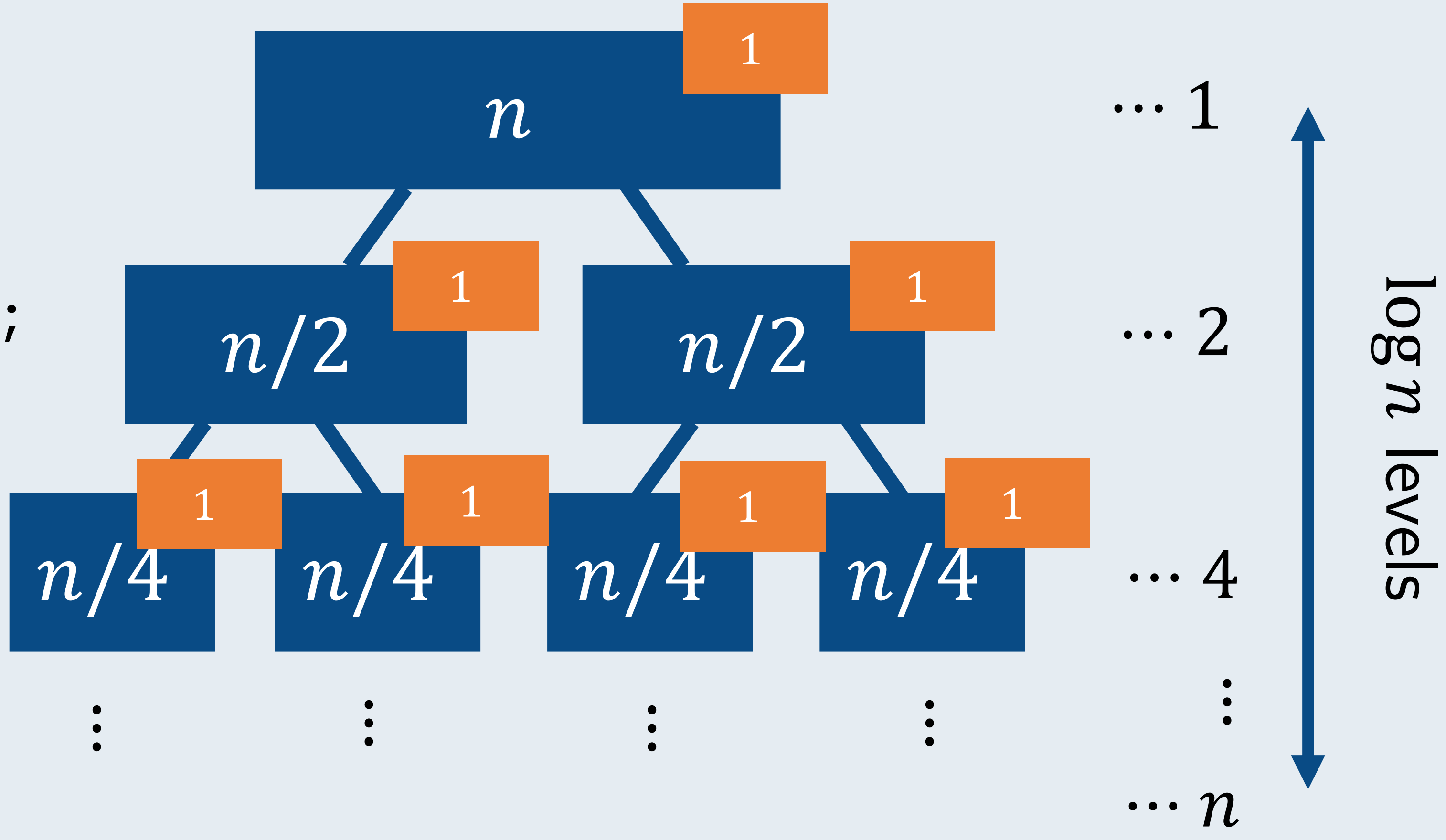
$T(n/2) = 2T(n/4) + 1$

$$
\begin{aligned}
T(n) &= 2\boxed{T(n/2)} + 1 \\
&= 4T(n/4) + 3 \\
&= 8T(n/8) + 7 \\
&= \cdots \\
&= nT(1) + (n-1) \\
&= 2n - 1 \\
&= O(n)
\end{aligned}
$$

Problem 2.f

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

Alternatively, build a recursion tree:



Problem 2.g

$T(n)$

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

n

$2T(n/2)$

Recurrence relation:

$$T(n) = 2T(n/2) + n$$

Problem 2.g

$T(n)$

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

n

$2T(n/2)$

$T(n/2) = 2T(n/4) + n/2$

$$
\begin{aligned}
T(n) &= 2\boxed{T(n/2)} + n \\
&= 4T(n/4) + n + n \\
&= \cdots \\
&= nT(1) + \underbrace{n + n + \cdots + n}_{\log n \text{ of them}} \\
&= n + n \log n \\
&= O(n \log n)
\end{aligned}
$$

Problem 2.h

$T(n, m)$

```java
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);
}
```

1 if $n > 1$

m if $n = 1$

$2T(n/2, m)$

Recurrence relation:

here $n > 1$

$$\begin{cases} T(n, m) = 2T(n/2, m) + 1 \\ T(1, m) = m \end{cases}$$

Problem 2.h

$T(n, m)$

```
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);
}
```

1 if $n > 1$

m if $n = 1$

$2T(n/2, m)$

$$
\begin{aligned}
T(n, m) &= 2T(n/2, m) + 1 \\
&= 4T(n/4, m) + 3 \\
&= \cdots \\
&= nT(1, m) + (n - 1) \\
&= mn + n - 1 \\
&= O(mn)
\end{aligned}
$$

Appendix

Master Theorem *

An easier way of solving recurrence relationship.

Master Theorem. Given a recurrence relationship $T(n) = aT(n/b) + f(n)$,

$$T(n) = \begin{cases} O\left(n^{\log_b a}\right), & \text{if } f(n) \prec n^{\log_b a} \\ O\left(n^{\log_b a} \log n\right), & \text{if } f(n) \equiv n^{\log_b a} \\ O\left(f(n)\right), & \text{if } f(n) \succ n^{\log_b a} \end{cases}$$

Master Theorem *

Recall in problem 2.g, we have $T(n) = 2T(n/2) + O(n)$, here

$$\begin{cases} a = 2 \\ b = 2 \\ f(n) = O(n) \equiv n^{\log_b a} \end{cases}$$

Therefore it's the second case, $T(n) = O\left(n^{\log_b a} \log n\right) = O(n \log n)$.

Pause and Ponder 1

Question 1: can we write $n^{\log_a b} = O(n^{\log b})$?

Not unless $\boldsymbol{a} = 2$! because

$$n^{\log_a b} = n^{\frac{\log b}{\log a}} = \left(n^{\log b}\right)^{\frac{1}{\log a}}$$

Is only equivalent to $n^{\log b}$ when $1/\log a$ is 1, i.e. $a = 2$.

Pause and Ponder 1

Question 2: can we write $\log_b n = O(\log n)$ if b is some variable dependent on n?

Not unless *b* is a constant value! We still have

$$\log_b n = \frac{\log n}{\log b}$$

But this time $1/\log b$ cannot be treated as constant coefficient and ignored.

Pause and Ponder 2 *

Question: How do we know $O(n \log n)$ is the tightest bound? Is $\log n! \equiv n \log n$?

To prove $f(n) \equiv g(n)$, we have two ways:

Method 1: $f(n) \equiv g(n) \Leftrightarrow \frac{f(n)}{g(n)} \to C$ as n goes to infinity (C is a positive constant).

Method 2: find a lower bound and an upper bound for $f(n)$ that are both $\equiv g(n)$.

Pause and Ponder 2: Method 1 *

Question: How do we know $O(n \log n)$ is the tightest bound? Is $\log n! \equiv n \log n$?

We actually have

$$\frac{\log n!}{n \log n} \xrightarrow{n \to \infty} 1 > 0$$

But this result is actually quite hard to get (need some calculus)... Method 2 would be more feasible.

Pause and Ponder 2: Method 2 *

Upper bound is already proved:

$\log n! = \log(n \times (n-1) \times \cdots \times 2 \times 1) = \log n + \log(n-1) + \cdots + \log 1 < \log n + \log n + \cdots + \log n \equiv n \log n$

Lower bound:

$$\log n! > \log n + \log(n-1) + \cdots + \log(n/2)$$

$$> \underbrace{\log(n/2) + \log(n/2) + \cdots + \log(n/2)}_{n/2 \text{ of them}} = \frac{n \log(n/2)}{2} = \frac{n \log n - n}{2} \equiv n \log n$$

End of File

Thank you very much for your attention :-)