Tutorial 1: Asymptotic Analysis

August 22, 2022

Gu Zhenhao

* Partly adopted from tutorial slides by Wang Zhi Jian.

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 teach problem-solving recipes.

- **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 * 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?

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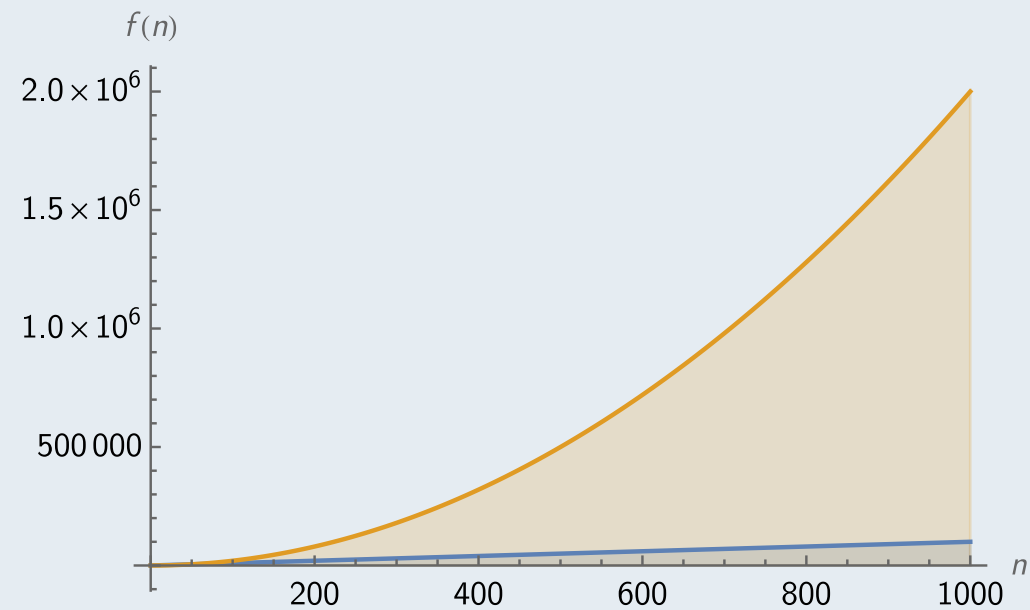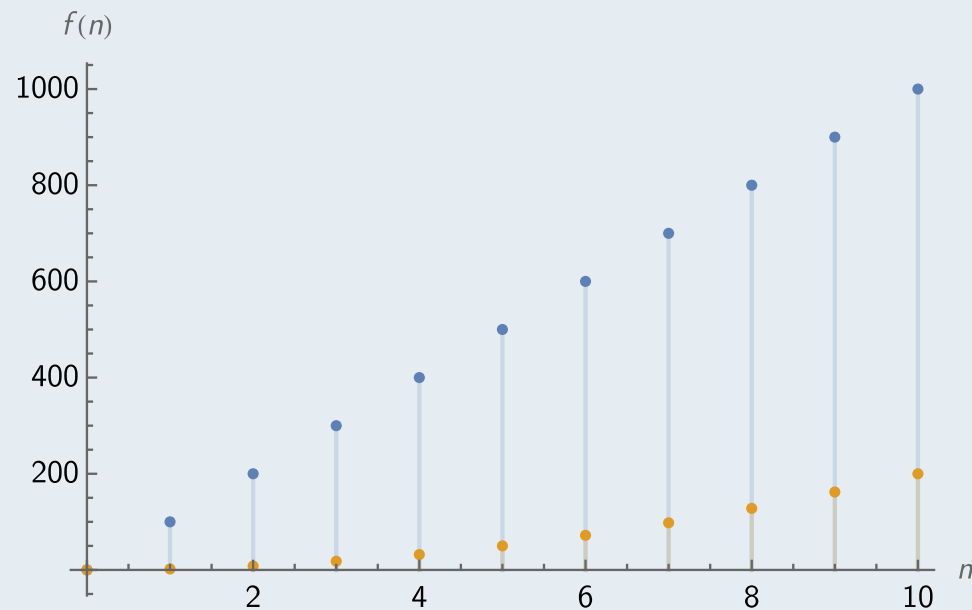

Figure: comparison of $f(n) = 100n$ (blue) and $f(n) = 2n^2$ (orange)

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. $n^2 + n = O(n^2)$

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

$$0 < c < 1$$

$$O(1) \prec O(\log n) \prec O(n^c) \prec O(n) \prec O(n \log n) \prec O(n^2) \prec O(n^3) \prec O(2^n) \prec O(n!)$$

Constant   Logarithmic   Fractional   Linear   Linearithmic   Polynomial   Exponential   Factorial
                           Power

You can find a more complete list here.

What if my formula is not on the list?

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

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

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

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!

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	$n \log n$
$100n^{\frac{2}{3}}$	$\log[(\log n)^2]$	$n!$	$(n-1)!$

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

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)

Based on change of base rule we can ignore the base in logarithm factor,

e.g. $\log_k n = O(\log n)$

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

* Find the answer in the appendix!

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	$n \log n$
$100n^{\frac{2}{3}}$	$\log[(\log n)^2]$	$n!$	$(n-1)!$

Find the tightest big-O complexity:

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

Find the tightest big-O complexity:

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

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

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

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!

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 $O(n!)$?

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

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	$n \log n$
$100n^{\frac{2}{3}}$	$\log[(\log n)^2]$	$n!$	$(n-1)!$

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	$n \log n$
$100n^{\frac{2}{3}}$	$\log[(\log n)^2]$	$n!$	$(n-1)!$

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

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	$n \log n$
$100n^{\frac{2}{3}}$	$\log[(\log n)^2]$	$n!$	$(n-1)!$

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

Find the tightest big-O complexity:

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

Find the tightest big-O complexity:

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

Find the tightest big-O complexity:

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

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

Find the tightest big-O complexity:

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

Find the tightest big-O complexity:

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

Question: Can we say $(n-1)! = O(n!)$?

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

Find the tightest big-O complexity:

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

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

Running Time Analysis

Given an algorithm, how to determine its run time?

```
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.

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

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

Each run costs $O(1)$ time.

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

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

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

$$\text{each run costs } O(1).$$

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

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

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

```java
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 $O(1)$.

```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.

```
while (n > 0) {

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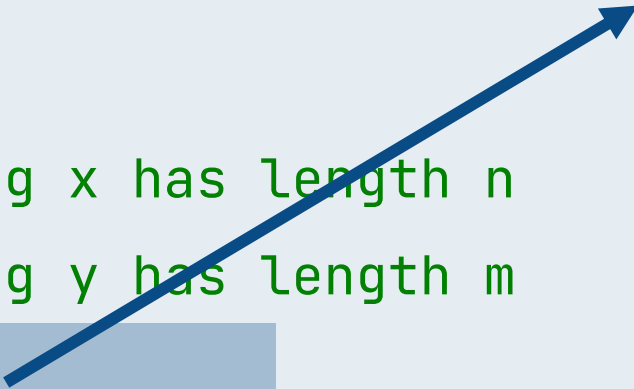
Run

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

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

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

Java strings are immutable.

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

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

```java
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.

$$T(n)$$

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

Recurrence relation:

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

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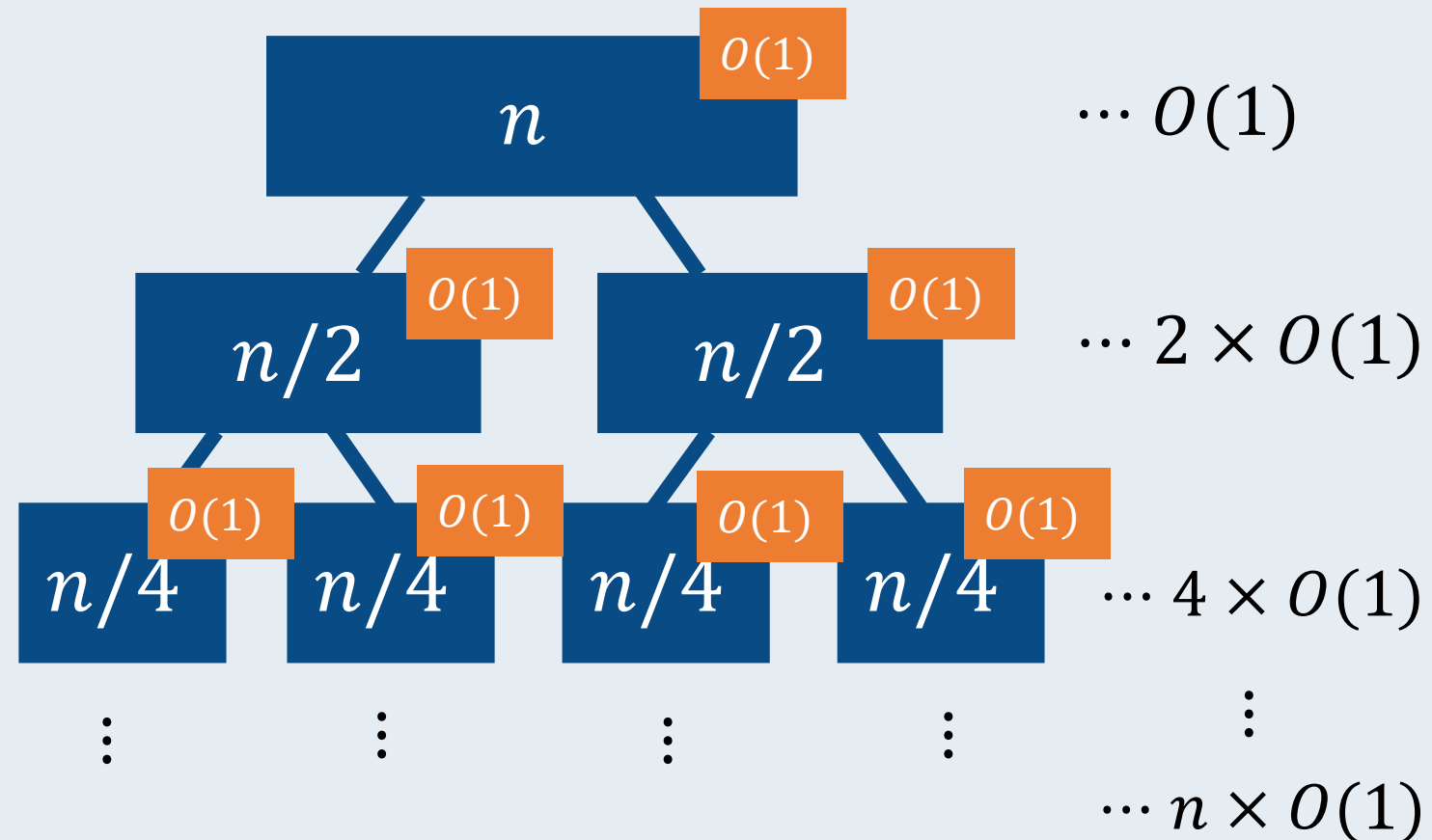
```java
void foo(int n){
    if (n <= 1)          O(1)
        return;
    System.out.println("*");
    foo(n/2);            2T(n/2)
    foo(n/2);
}
```

$$T(n) = 2T(n/2) + O(1)$$
$$= 4T(n/4) + O(1) + O(1)$$
$$= \cdots$$
$$= nT(1) + \underbrace{O(1) + \cdots + O(1)}_{\log n \text{ of them}}$$
$$= nO(1) + O(\log n)$$
$$= O(n)$$

Alternatively, build a recursion tree:

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


n $O(1)$ $\cdots O(1)$

$n/2$ $O(1)$ $n/2$ $O(1)$ $\cdots 2 \times O(1)$

$n/4$ $O(1)$ $n/4$ $O(1)$ $n/4$ $O(1)$ $n/4$ $O(1)$ $\cdots 4 \times O(1)$

$\cdots n \times O(1)$

$T(n)$

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

Recurrence relation:

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

$T(n)$

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

$$T(n) = 2T(n/2) + O(n)$$
$$= 4T(n/4) + O(n) + O(n)$$
$$= \cdots$$
$$= nT(1) + \underbrace{O(n) + \cdots + O(n)}_{\log n \text{ of them}}$$
$$= nO(1) + O(n \log n)$$
$$= O(n \log n)$$

$$T(n, m)$$

```java
void foo(int n, int m){

    if (n <= 1) {

        for (int i = 0; i < m; i++) {

            System.out.println("*");

        }

        return;                    O(1) if n > 1
    }                              O(m) if n = 1

    foo(n/2, m);
    foo(n/2, m);                   2T(n/2, m)

}
```

Recurrence relation:

here $n > 1$

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

$T(n, m)$

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

$$T(n, m) = 2T(n/2, m) + O(1)$$
$$= 4T(n/4) + O(1) + O(1)$$
$$= \cdots$$
$$= nT(1) + \underbrace{O(1) + \cdots + O(1)}_{\log n \text{ of them}}$$
$$= nO(m) + O(\log n)$$
$$= O(mn)$$

Appendix

An easier way of solving recurrence relationship.

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

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

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, $T(n) = O\left(n^{\log_b a} \log n\right) = O(n \log n)$.

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

Not unless $a = 2$! 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. $a = 2$.

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: $f(n) \equiv g(n) \Leftrightarrow \frac{f(n)}{g(n)} \rightarrow C$ 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 $\equiv g(n)$.

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

Thanks a lot for your attention :-)