

LECTURE 04 — ASYMPTOTIC NOTATIONS

COMPSCI 308 — DESIGN AND ANALYSIS OF ALGORITHMS

Xing Shi Cai <https://newptcai.gitlab.io>

Duke Kunshan University, Kunshan, China

Jan-Mar 2026

Asymptotic Analysis of Insertion Sort

Asymptotic Notations in Equations and Inequalities

Small o and Small ω Notations

Comparing Functions

Common Time Complexity

ASSIGNMENTS¹



Practice makes perfect!

Introduction to Algorithms (ITA)

Read —

- Section 3.1
- Section 3.2

Practice —

- Exercises 3.2: 1–7.
- Problems 3-1, 3-2, 3-3, 3-4.

¹ ⚪ Assignments will not be collected; however, quiz problems will be selected from them. (This includes both Readings and Exercises.)

ASYMPTOTIC ANALYSIS OF INSERTION SORT

INSERTION SORT

When $A.length = n$, what is $T(n)$, the total running time of the algorithm?

INSERTION-SORT(A, n)

```
1: for  $i = 2$  to  $n$  do
2:    $key = A[i]$ 
3:   // Insert  $A[i]$  into the sorted subarray  $A[1 : i - 1]$ .
4:    $j = i - 1$ 
5:   while  $j > 0$  and  $A[j] > key$  do
6:      $A[j + 1] = A[j]$ 
7:      $j = j - 1$ 
8:    $A[j + 1] = key$ 
```

INSERTION SORT

Let's introduce some notation —

- Let c_ℓ be the cost (running time) of line ℓ .
- Let t_i be the number of times line 5 is executed for each i .

INSERTION-SORT(A, n)

```
1: for  $i = 2$  to  $n$  do
2:    $key = A[i]$ 
3:   // Insert  $A[i]$  into the sorted subarray  $A[1 : i - 1]$ .
4:    $j = i - 1$ 
5:   while  $j > 0$  and  $A[j] > key$  do
6:      $A[j + 1] = A[j]$ 
7:      $j = j - 1$ 
8:    $A[j + 1] = key$ 
```

	cost	times
c_1	n	
c_2	$n - 1$	
0	$n - 1$	
c_4	$n - 1$	
c_5	$\sum_{i=2}^n t_i$	
c_6	$\sum_{i=2}^n (t_i - 1)$	
c_7	$\sum_{i=2}^n (t_i - 1)$	
c_8	$n - 1$	

BEST-CASE ANALYSIS

Given that

$$T(n) = c_1n + c_2(n - 1) + c_4(n - 1) + c_5 \sum_{i=2}^n t_i + c_6 \sum_{i=2}^n (t_i - 1) + c_7 \sum_{i=2}^n (t_i - 1) + c_8(n - 1)$$

how small can it be?

- 💡 What is the smallest possible value for t_i ?

WORST-CASE ANALYSIS

Given that

$$T(n) = c_1n + c_2(n - 1) + c_4(n - 1) + c_5 \sum_{i=2}^n t_i + c_6 \sum_{i=2}^n (t_i - 1) + c_7 \sum_{i=2}^n (t_i - 1) + c_8(n - 1)$$

how large can it be?

 What is the largest possible value for t_i ?

WORST-CASE ANALYSIS

Given that

$$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5 \sum_{i=2}^n t_i + c_6 \sum_{i=2}^n (t_i - 1) + c_7 \sum_{i=2}^n (t_i - 1) + c_8(n-1)$$

how large can it be?

If $t_i = i$ (the largest possible value of t_i), we have

$$\begin{aligned} T(n) &= c_1n + c_2(n-1) + c_4(n-1) + c_5 \left(\frac{n(n+1)}{2} - 1 \right) \\ &\quad + c_6 \left(\frac{n(n-1)}{2} \right) + c_7 \left(\frac{n(n-1)}{2} \right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2} \right) n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8 \right) n \\ &\quad - (c_2 + c_4 + c_5 + c_8). \end{aligned}$$

Let $T(n)$ be the worst case running time of insertion sort on Random Access Machine (RAM). In other words,

$$T(n) = \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2} \right) n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8 \right) n - (c_2 + c_4 + c_5 + c_8).$$

🎂 Which of the following are correct?

1. $T(n) = O(n^3)$
2. $T(n) = O(2n^2)$
3. $T(n) = O(n^2)$
4. $T(n) = \Theta(n^2)$
5. $T(n) = \Omega(n)$
6. $T(n) = \Omega(2n)$
7. $T(n) = \Omega(\sqrt{n})$

ASYMPTOTIC NOTATIONS IN EQUATIONS AND INEQUALITIES

ASYMPTOTIC NOTATIONS IN EQUATIONS

Given equations like

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n) \quad (1)$$

we interpret $\Theta(n)$ as an **anonymous function** $f(n)$ which satisfies

$$f(n) = \Theta(n)$$

How can we show that (1) is correct?

ASYMPTOTIC NOTATIONS IN EQUATIONS

Given equations like

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n) \quad (1)$$

we interpret $\Theta(n)$ as an **anonymous function** $f(n)$ which satisfies

$$f(n) = \Theta(n)$$



Is the following equation correct?

$$2n^2 + 3n + 1 = n^2 + \Omega(n)$$

THE NUMBER OF ANONYMOUS FUNCTIONS

Each appearance of an asymptotic notation in an equation is interpreted as an **anonymous function**.

Thus,

$$n^3 + O(n^2) + O(n)$$

should be interpreted as

$$n^3 + f(n) + g(n)$$

where

$$f(n) = O(n^2), \quad g(n) = O(n)$$

However, it is uncommon to see such equations since we can usually combine big-O notations.

AVOID ANONYMOUS FUNCTIONS IN SUMS

⚠️ Thus, the sum

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

should *not* be interpreted as

$$O(1) + O(2) + \cdots + O(n)$$

but

$$\sum_{i=1}^n f(i)$$

where $f(i) = O(i)$ for all $i \in \{1, \dots, n\}$ is an anonymous function.

💣 However, it does not really make sense to write expressions like

$$f(3) = O(3)$$

MERGE SORT²

The worst-case running time of merge sort is

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

Later we will see that this equation is enough to find the asymptotic behaviour of $T(n)$.

 So there is no need to figure out what the $\Theta(n)$ part is exactly.

²Merge sort will be covered in lecture 5.

ASYMPTOTIC NOTATIONS ON BOTH SIDES OF EQUATIONS

We interpret the equation

$$2n^2 + \Theta(n) = \Theta(n^2) \quad (2)$$

as — For any $f(n) = \Theta(n)$, $2n^2 + f(n) = \Theta(n^2)$.

 Is (2) correct?

ASYMPTOTIC NOTATIONS ON BOTH SIDES OF EQUATIONS

We interpret the equation

$$2n^2 + \Theta(n) = \Theta(n^2) \quad (2)$$

as — For any $f(n) = \Theta(n)$, $2n^2 + f(n) = \Theta(n^2)$.

 Is (2) correct? Answer: Yes. Since $f(n) = \Theta(n)$, there exist $c_1, c_2, n_0 > 0$ such that

$$c_1 n \leq f(n) \leq c_2 n, \quad \text{for all } n \geq n_0.$$

Adding $2n^2$ to the inequality leads to

$$2n^2 + c_1 n \leq 2n^2 + f(n) \leq 2n^2 + c_2 n.$$

In other words,

$$2n^2 \leq 2n^2 + f(n) \leq (2 + c_2)n^2, \quad \text{for all } n \geq n_0.$$

Thus, $2n^2 + f(n) = \Theta(n^2)$ by definition.

ASYMPTOTIC NOTATIONS ON BOTH SIDES OF EQUATIONS

We interpret the equation

$$2n^2 + \Theta(n) = \Theta(n^2) \quad (2)$$

as — For any $f(n) = \Theta(n)$, $2n^2 + f(n) = \Theta(n^2)$.

 Is $n^3 + \Theta(n^2) = \Theta(n^2)$ correct?

ASYMPTOTIC NOTATIONS ON BOTH SIDES OF EQUATIONS

We interpret the equation

$$2n^2 + \Theta(n) = \Theta(n^2) \quad (2)$$

as — For any $f(n) = \Theta(n)$, $2n^2 + f(n) = \Theta(n^2)$.

 Is $n^3 + \Theta(n^2) = \Theta(n^2)$ correct?

Answer: No. By definition, any $g(n) = \Theta(n^2)$ must satisfy $g(n) \leq c \cdot n^2$ for some $c > 0$ and all $n \geq n_0$ where $n_0 \geq 0$ is a constant.

However, for any $f(n) = \Theta(n^2)$, we have:

$$\lim_{n \rightarrow \infty} \frac{n^3 + f(n)}{n^2} = \lim_{n \rightarrow \infty} \left(n + \frac{f(n)}{n^2} \right) = \infty.$$

Because this ratio grows without bound, no constant c can satisfy the upper bound requirement. Therefore, $n^3 + f(n) \neq \Theta(n^2)$.

CHAINING ASYMPTOTIC EQUATIONS

We interpret

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n) = \Theta(n^2) \quad (3)$$

as two separate equations

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n)$$

and

$$2n^2 + \Theta(n) = \Theta(n^2)$$



Does (3) hold?

SMALL o AND SMALL ω NOTATIONS

TIGHT ASYMPTOTIC UPPER BOUND

The upper bound provided O may or may not be *tight*.

For example, $2n^2 = O(n^2)$ is tight in the sense that

$$\lim_{n \rightarrow \infty} \frac{2n^2}{n^2} = 2.$$

On the other hand, $n = O(n^2)$ is *not* tight in the sense that

$$\lim_{n \rightarrow \infty} \frac{n}{n^2} = 0.$$

In the latter case, we can replace the O by o and write

$$n = o(n^2).$$

o -NOTATIONS

Formally, we write $f(n) = o(g(n))$ if for any $c > 0$, there exists n_0 such that

$$0 \leq f(n) < cg(n), \quad \text{for all } n \geq n_0.$$



Can you see why $f(n) = o(g(n))$ if and only if

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0.$$

ω -NOTATIONS

Formally, we write $f(n) = \omega(g(n))$ if for any $c > 0$, there exists n_0 such that

$$0 \leq cg(n) < f(n), \quad \text{for all } n \geq n_0.$$

In other words, $f(n) = \omega(g(n))$ if and only if

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty.$$

COMPARING FUNCTIONS

TRANSITIVITY IN ASYMPTOTIC NOTATIONS

Transitivity means that if $f(n)$ is related to $g(n)$ and $g(n)$ is related to $h(n)$ by the same notation, then $f(n)$ is also related to $h(n)$ by that notation:

Example – Transitivity in real life:

- You are friends with Alex.
- Alex is friends with Billy.
- Billy is friends with Cindy.
- Cindy is friends with David.
- David is friends with Taylor Swift.
- Therefore, you and Taylor Swift are also friends!

TRANSITIVITY IN ASYMPTOTIC NOTATIONS

Transitivity means that if $f(n)$ is related to $g(n)$ and $g(n)$ is related to $h(n)$ by the same notation, then $f(n)$ is also related to $h(n)$ by that notation:

- $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n)) \implies f(n) = \Theta(h(n))$
- $f(n) = O(g(n))$ and $g(n) = O(h(n)) \implies f(n) = O(h(n))$
- $f(n) = \Omega(g(n))$ and $g(n) = \Omega(h(n)) \implies f(n) = \Omega(h(n))$
- $f(n) = o(g(n))$ and $g(n) = o(h(n)) \implies f(n) = o(h(n))$
- $f(n) = \omega(g(n))$ and $g(n) = \omega(h(n)) \implies f(n) = \omega(h(n))$

REFLEXIVITY IN ASYMPTOTIC NOTATIONS

Reflexivity means that any function $f(n)$ is related to itself by some asymptotic notations:

$$f(n) = \Theta(f(n)),$$

$$f(n) = O(f(n)),$$

$$f(n) = \Omega(f(n))$$

REFLEXIVITY IN ASYMPTOTIC NOTATIONS

Reflexivity means that any function $f(n)$ is related to itself by some asymptotic notations:

$$\begin{aligned}f(n) &= \Theta(f(n)), \\f(n) &= O(f(n)), \\f(n) &= \Omega(f(n))\end{aligned}$$



Do we have

$$\begin{aligned}f(n) &= o(f(n)), \\f(n) &= \omega(f(n)).\end{aligned}$$

SYMMETRY IN ASYMPTOTIC NOTATIONS

Symmetry means that if $f(n)$ is related to $g(n)$ by an asymptotic notation, then $g(n)$ should be related to $f(n)$ by the same notation.

For example —

$$f(n) = \Theta(g(n)) \implies g(n) = \Theta(f(n)).$$

SYMMETRY IN ASYMPTOTIC NOTATIONS

Symmetry means that if $f(n)$ is related to $g(n)$ by an asymptotic notation, then $g(n)$ should be related to $f(n)$ by the same notation.

For example —

$$f(n) = \Theta(g(n)) \implies g(n) = \Theta(f(n)).$$

🍰 Do we have

$$f(n) = O(g(n)) \implies g(n) = \Omega(f(n)),$$

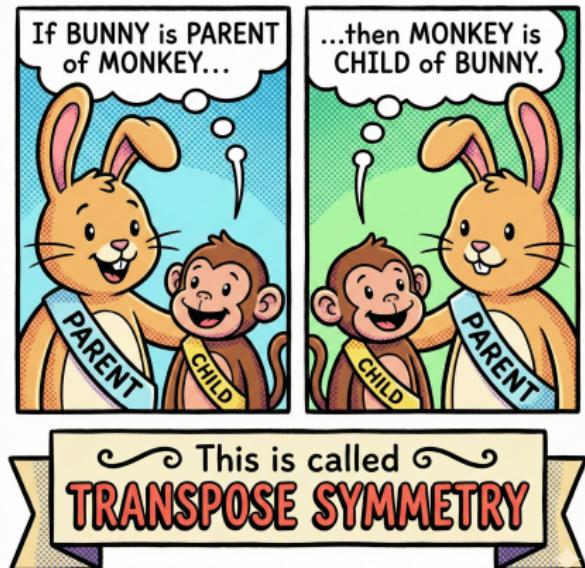
$$f(n) = \Omega(g(n)) \implies g(n) = O(f(n))$$

TRANSPOSE SYMMETRY IN ASYMPTOTIC NOTATIONS

Transpose Symmetry means that if $f(n)$ is related to $g(n)$ by one asymptotic notation, then $g(n)$ should be related to $f(n)$ by another notation:

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \underline{\hspace{2cm}}(f(n)),$$

$$f(n) = o(g(n)) \Leftrightarrow g(n) = \underline{\hspace{2cm}}(f(n)).$$



Trichotomy — The division into three mutually exclusive or contrasting groups, categories, or parts.

📌 For example, real numbers have a **trichotomy** — for any $a, b \in \mathbb{R}$, exactly one of the following must hold:

$$a < b, \quad a = b, \quad a > b.$$



Figure 1: A 🐰 facing a trichotomy

TRICHOTOMY IN ASYMPTOTIC NOTATIONS

We say that

- $f(n)$ is **asymptotically smaller** than $g(n)$ if $f(n) = o(g(n))$, denoted by $f(n) \prec g(n)$;
 - $f(n)$ is **asymptotically larger** than $g(n)$ if $f(n) = \omega(g(n))$, denoted by $f(n) \succ g(n)$.
- 💣 Not all functions are asymptotically comparable, e.g.:

$$f(n) = n, \quad g(n) = n^{1+\sin n}.$$

No TRICHOTOMY IN ASYMPTOTIC NOTATIONS

We say that

- $f(n)$ is **asymptotically smaller** than $g(n)$ if $f(n) = o(g(n))$, denoted by $f(n) \prec g(n)$;
 - $f(n)$ is **asymptotically larger** than $g(n)$ if $f(n) = \omega(g(n))$, denoted by $f(n) \succ g(n)$.
- 💣 Not all functions are asymptotically comparable, e.g.:

$$f(n) = n, \quad g(n) = n^{1+\sin n}.$$

✳️ Are monotonically increasing functions always asymptotically comparable?

COMMON TIME COMPLEXITY

COMMON TIME COMPLEXITY

Let $T(n)$ be the average/worst-case running time of an algorithm given an input of size n .

The **time complexity** of this algorithm is the rate of growth of $T(n)$.

Time complexity is a way to express how the running time of an algorithm grows as the size of the input to the algorithm increases.

Some common time complexities are:

- constant: $O(1)$
- log-logarithmic: $O(\log \log n)$
- logarithmic: $O(\log n)$
- polylogarithmic: $O(\text{poly}(\log n))$
- sub-linear: $o(n)$

COMMON TIME COMPLEXITY

Let $T(n)$ be the average/worst-case running time of an algorithm given an input of size n .

The **time complexity** of this algorithm is the rate of growth of $T(n)$.

Time complexity is a way to express how the running time of an algorithm grows as the size of the input to the algorithm increases.

Some common time complexities are:

- constant: $O(1)$
- log-logarithmic: $O(\log \log n)$
- logarithmic: $O(\log n)$
- polylogarithmic: $O(\text{poly}(\log n))$
- sub-linear: $o(n)$
- linear: $O(n)$
- linearithmic: $O(n \log n)$
- quadratic: $O(n^2)$
- cubic: $O(n^3)$
- exponential: $2^{O(n)}$

CONSTANT TIME COMPLEXITY

A constant time algorithm has running time $O(1)$.

In other words, its running time is bounded by a constant regardless of the input size.

📌 Examples include:

- Conditional branch (e.g., `if (x > 0) {...}`)
- Arithmetic/logic operation (e.g., `y = a + b`)
- Declare/initialize a variable (e.g., `int x = 0`)
- Follow a link in a linked list (e.g., `node = node->next`)
- Access element i in an array (e.g., `A[i]`)
- Compare/exchange two elements in an array (e.g., swap `A[i], A[j]`)

LINEAR TIME COMPLEXITY

A linear time algorithm has running time $O(n)$.

In other words, its running time is bounded by a constant factor times the input size.

A common reason for this is the algorithm processes the input in a single pass.

 Examples include finding the maximum element in an array.

 Can you think of another example?

A linear time algorithm has running time $O(n)$.

In other words, its running time is bounded by a constant factor times the input size.

A common reason for this is the algorithm processes the input in a single pass.

📌 Examples include finding the maximum element in an array.

🎂 Can you think of another example?

Finding the Median

The **median** of a set of numbers is the value separating the higher half from the lower half.

For example, the median of

0, 2, 3, 4, 11

is 3.

Can we find the median of an unsorted array in linear time?

LOGARITHMIC TIME COMPLEXITY

A logarithmic time algorithm has running time $O(\log n)$.

- Examples include searching for an element in a sorted array.

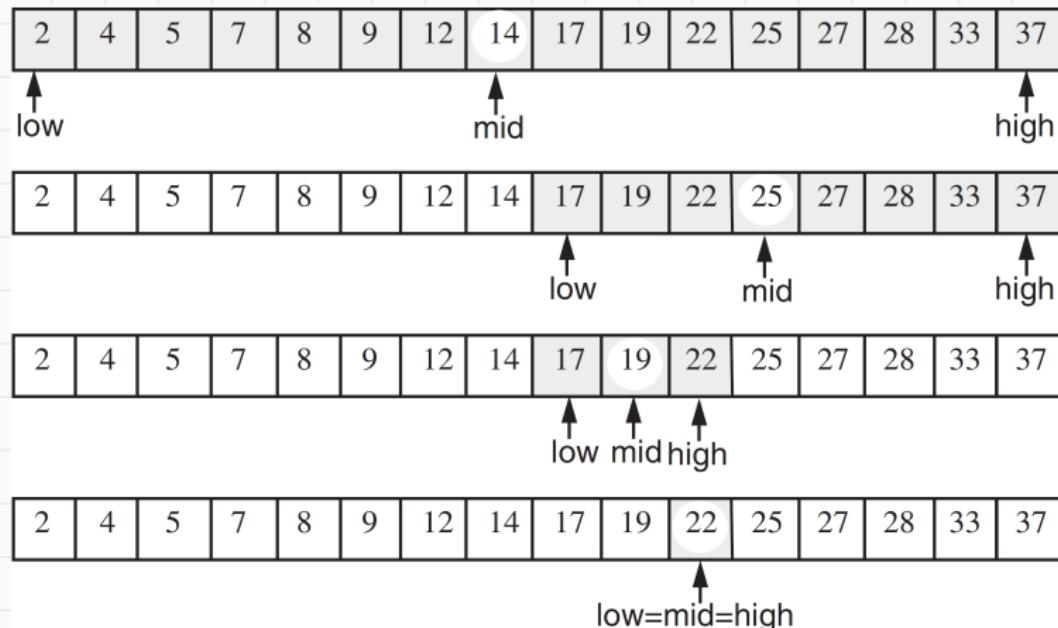


Figure 2: Binary search for 22 in A

LOGARITHMIC TIME COMPLEXITY

A logarithmic time algorithm has running time $O(\log n)$.

📌 Examples include searching for an element in a sorted array.

```
1: Binary-Search( $A[1..n]$ ,  $key$ )
2:  $low \leftarrow 1$ ;  $high \leftarrow n$ 
3: while  $low \leq high$  do
4:    $mid \leftarrow (low + high)/2$ 
5:   if  $A[mid] = key$  then return  $mid$ 
6:   else if  $A[mid] < key$  then
7:      $low \leftarrow mid + 1$ 
8:   else
9:      $high \leftarrow mid - 1$ 
10:  end if
11: return  $NULL$ 
```

Given the array of length 15,

$$A = \langle 12, 25, 32, 37, 41, 48, 58, 60, 66, 73, 74, 79, 83, 91, 95 \rangle$$

How many comparisons does it take to find the position of 74 with binary search?

- 💡 You should count = and < as two comparisons.

LINEAR TIME COMPLEXITY

A linear time algorithm has running time $O(n)$.

📌 Examples include searching for an element in an unsorted array.

```
1: Linear-Search( $A[1..n]$ ,  $key$ )
2: for  $i \leftarrow 1$  to  $n$  do
3:   if  $A[i] = key$  then
4:     return  $i$ 
5:   end if
6: return  $NULL$ 
```

QUADRATIC TIME COMPLEXITY

A quadratic time algorithm has running time $O(n^2)$.

📌 A simple example is the brute-force algorithm for finding the closest pair of points in the plane.

```
1: Closest-Pair( $x[1..n]$ ,  $y[1..n]$ )
2:  $d \leftarrow \infty$ 
3:  $cp \leftarrow (\text{NULL}, \text{NULL})$ 
4: for  $i \leftarrow 1$  to  $n - 1$  do
5:   for  $j \leftarrow i + 1$  to  $n$  do
6:      $dt \leftarrow \sqrt{(x[i] - x[j])^2 + (y[i] - y[j])^2}$ 
7:     if  $dt < d$  then
8:        $d \leftarrow dt$ 
9:        $cp \leftarrow ((x[i], y[i]), (x[j], y[j]))$ 
10:    end if
11: return  $cp$ 
```

EXPONENTIAL TIME COMPLEXITY

An exponential time algorithm has running time $O(c^n)$ for some constant $c > 1$.

📌 A classic example is the naïve recursive computation of Fibonacci numbers

$$\{F_n\}_{n=0}^{\infty} = \{0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \dots\}$$

which is defined recursively as

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

```
1: Fib(n)
2: if n ≤ 1 then
3:   return n
4: else
5:   return Fib(n - 1) + Fib(n - 2)
6: end if
```

WHAT ARE YOUR MAIN TAKEAWAYS TODAY? ANY QUESTIONS?

