WEEK 3 STUDIO

ABOUT ME

- Chen Yuan Bo
- Year 2, Computer Science
- Love dancing
- Telegram: @yuan_bobo

ABOUT YOU

- Name, School, Hobbies
- Why did you choose Computer Science?

HOW TO DO WELL IN CS1101S

- Do your tutorials. All of them.
- Understand. Don't memorize.
- Practice, practice, practice. (Missions, quests etc)
- Ask questions

AGENDA

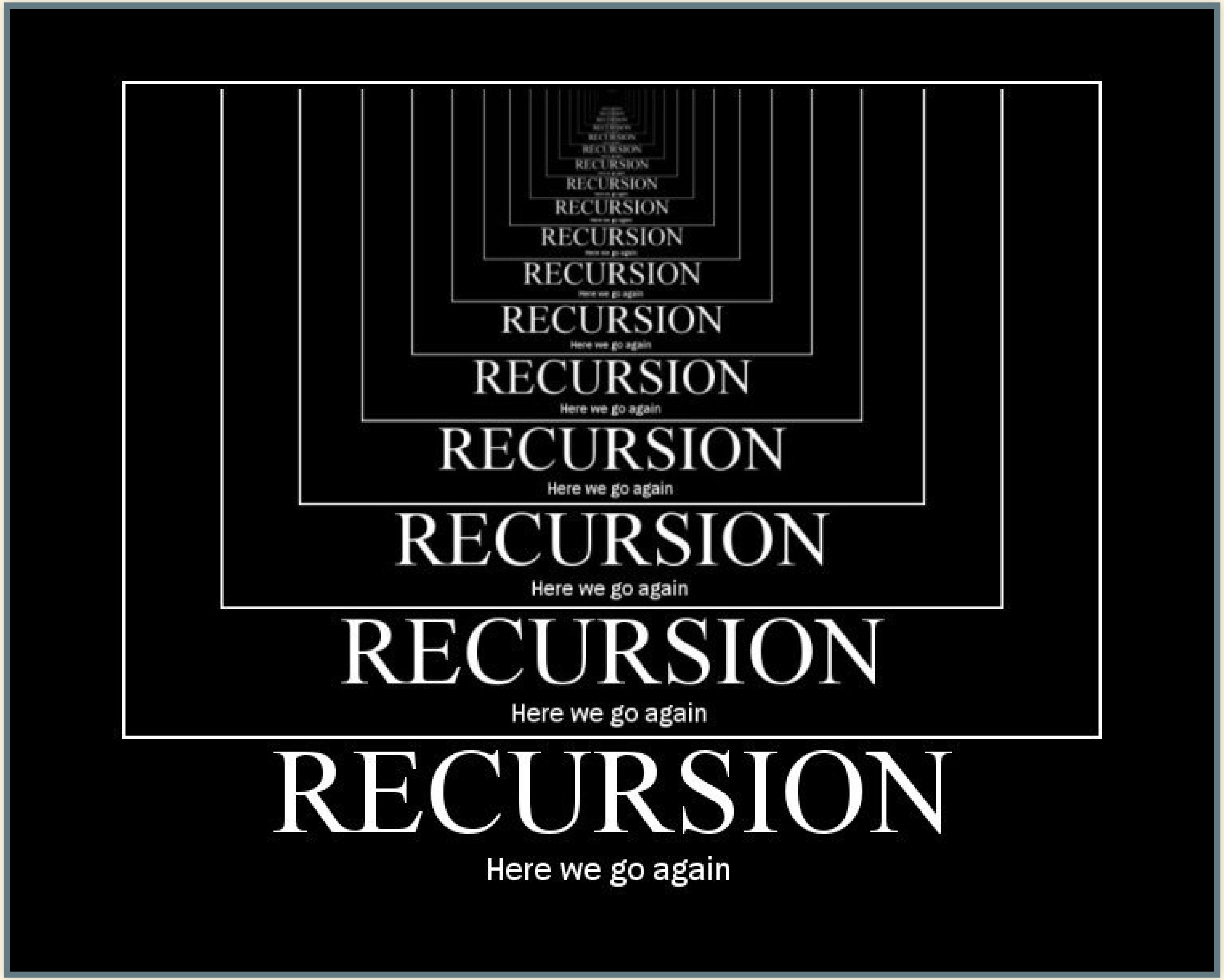
- Recursion
- Order of Growth
- Studio

RECURSION

WHAT IS RECURSION?

- Method of solving problems where solutions depends on solutions to smaller instances of the same problem

WHAT IS RECURSION?


RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
RECURSION
RECURSION
RECURSION

WHAT IS RECURSION?

WHAT IS RECURSION?

USING RECURSION TO SOLVE PROBLEMS

- Break the problem into smaller sub-problems
    - Use wishful thinking!
- Base case
    - Smallest sub-problem possible, where the function call just evaluates to a result directly

RECURSIVE FUNCTION

Any function that calls itself is a recursive function.

EXAMPLE RECURSIVE FUNCTION

```
function factorial(n) {
    return n === 1
        ? 1
        : n * factorial(n - 1);
}
```

TOWER OF HANOI

Tower of Hanoi consists of three pegs or towers with n disks placed one over the other.
The objective of the puzzle is to move the stack to another peg following these simple rules.

1. *Only one disk can be moved at a time.*
2. *No disk can be placed on top of the smaller disk.*

TOWER OF HANOI

```
function hanoi(num, current, target, other) {
    if (num === 1) {
        display("move " + current + " to " + target);
    } else {
        hanoi(num - 1, current, other, target);
        display("move " + current + " to " + target);
        hanoi(num - 1, other, target, current);
    }
}

hanoi(3, "1", "2", "3");
```

RECURSIVE PROCESS VS ITERATIVE PROCESS

- Deferred operations
- Why?

RECURSIVE PROCESS VS ITERATIVE PROCESS

```
function factorial(n) {
    return n === 1
        ? 1
        : n * factorial(n - 1);
}

function fact_iter(n) {
    return iter(1, 1, n);
}

function iter(counter, product, max_count) {
    return counter > max_count
        ? product
        : iter(counter + 1, counter * product, max_count);
}
```

desmos

WHAT IS ORDER OF GROWTH?

- **Purpose**: Find a rough measure of the resources (time and/or space) used by a computational process (a program).
- **Approach**: Use a mathematical function to describe how the amount of resources consumed grows along with the scale of the problem.

TYPES OF COMPLEXITY (FOR CS1101S)

- Time: how long does the program run
  - Number of steps/function calls
- Space: how much memory do we need to run the program
  - Number of deferred operations

TYPES OF ORDER OF GROWTH

Order (Time)	Description	Example
$\mathcal{O}(1)$	Constant	Add, subtract, multiply, square...
$\mathcal{O}(\log n)$	Logarithmic	Binary search
$\mathcal{O}(n)$	Linear	Factorial
$\mathcal{O}(n \log n)$	Nifty	Merge Sort
$\mathcal{O}(n^2)$	Quadratic	Bubble Sort, Selection Sort
$\mathcal{O}(2^n)$	Exponential	Fibonacci

EXAMPLE 1

```
function foo(n) {
    return n < 0
        ? 0
        : foo(n - 1);
}
```

- Time complexity: $\mathcal{O}(n)$
- Space complexity: $\mathcal{O}(1)$

EXAMPLE 2

```
function foo(n) {
    return n < 0
        ? 0
        : foo(n - 1) + 2;
}
```

- Time complexity: $\mathcal{O}(n)$
- Space complexity: $\mathcal{O}(n)$

EXAMPLE 3

```
function foo(n) {
    return n < 1
        ? 0
        : foo(n / 2);
}
```

- Time complexity: $\mathcal{O}(\log n)$
- Space complexity: $\mathcal{O}(1)$

EXAMPLE 4

```
function foo(n) {
    return n < 1
        ? 0
        : foo(n / 3) + 3;
}
```

- Time complexity: $\mathcal{O}(\log n)$
- Space complexity: $\mathcal{O}(\log n)$

EXAMPLE 5

```
function foo(n) {
    const k = n / 3;

    function iter(n) {
        return n < 0
            ? 0
            : iter(n - k);
    }

    return iter(n);
}
```

- Time complexity: $\mathcal{O}(1)$
- Space complexity: $\mathcal{O}(1)$

EXAMPLE 6

```
function foo(n) {
    return n < 0
        ? 0
        : foo(n - 1) + foo(n - 1);
}
```

- Time complexity: $\mathcal{O}(2^n)$
- Space complexity: $\mathcal{O}(n)$

EXAMPLE 7

```
function foo(n) {
    return n < 1
        ? 0
        : foo(n / 2) + foo(n / 2);
}
```

- Time complexity: $\mathcal{O}(n)$
- Space complexity: $\mathcal{O}(\log n)$

EXAMPLE 8

```
function foo(n) {
    const k = math_sqrt(n);
    function iter(n) {
        return n < 0
            ? 0
            : iter(n - k);
    }
    return iter(n);
}
```

- Time complexity: $\mathcal{O}(\sqrt{n})$
- Space complexity: $\mathcal{O}(1)$

ATTENDANCE TAKING

1. Open telegram, search for @CS1101SBot
2. Start the bot, and type in

    ```
    /setup <Your matric number>
    ```

3. Once you have successfully registered, type

    ```
    /attend <token>
    ```

4. Receive reply that you have successfully marked your attendance.

STUDIO SHEET