

National University of Singapore  
School of Computing  
CS1010X: Programming Methodology  
Semester II, 2022/2023

**Recitation 2**  
**Recursion, Iteration & Orders of Growth**

**Definitions**

Theta ( $\Theta$ ) notation:

$$f(n) = \Theta(g(n)) \Leftrightarrow \exists k_1, k_2, n_0 . k_1 \cdot g(n) \leq f(n) \leq k_2 \cdot g(n), \text{ for } n > n_0$$

Big-O notation:

$$f(n) = O(g(n)) \Leftrightarrow \exists k, n_0 . f(n) \leq k \cdot g(n), \text{ for } n > n_0$$

Adversarial approach: For you to show that  $f(n) = \Theta(g(n))$ , you pick  $k_1$ ,  $k_2$ , and  $n_0$ , then I (the adversary) try to pick an  $n$  which doesn't satisfy  $k_1 \cdot g(n) \leq f(n) \leq k_2 \cdot g(n)$ .

Terminology:

- $\exists$ : There exists.
- $\Leftrightarrow$ : If and only if (either both statements are true, or both are false).

**Implications**

Ignore constants. Ignore lower order terms. For a sum, take the larger term. For a product, multiply the two terms. Orders of growth are concerned with how the effort scales up as the size of the problem increases, rather than an exact measure of the cost.

**Typical Orders of Growth**

- $\Theta(1)$  - Constant growth. A fixed number of simple, non-decomposable operations have constant growth.
- $\Theta(\log n)$  - Logarithmic growth. At each iteration, the problem size is scaled down by a constant amount.
- $\Theta(n)$  - Linear growth. At each iteration, the problem size is decremented by a constant amount.
- $\Theta(n \log n)$  - Nifty growth. Nice recursive solution to normally  $\Theta(n^2)$  problem.
- $\Theta(n^2)$  - Quadratic growth. Computing correspondence between a set of  $n$  things, or doing something of cost  $n$  to all  $n$  things both result in quadratic growth.
- $\Theta(2^n)$  - Exponential growth. Really bad. Searching all possibilities usually results in exponential growth.

## What's $n$ ?

Order of growth is *always* in terms of the size of the problem. Without stating what the problem is, and what is considered primitive (what is being counted as a “unit of work” or “unit of space”), the order of growth doesn't have any meaning.

## Problems

1. Remember our point-of-sale and order-tracking system from last week? Recall that the joint only sells 4 options for combos: Classic Single Combo (hamburger with one patty), Classic Double With Cheese Combo (2 patties), and Classic Triple with Cheese Combo (3 patties), Avant-Garde Quadruple with Guacamole Combo (4 patties). We shall encode these combos as 1, 2, 3, and 4 respectively. Each meal can be *biggie-sized* to acquire a larger box of fries and drink. A *biggie-sized* combo is represented by 5, 6, 7, and 8 respectively, for combos 1, 2, 3, and 4 respectively.

In addition, an order is a collection of combos. We'll encode an order as each digit representing a combo. For example, the order 237 represents a Double, Triple, and *biggie-sized* Triple.

Assume that you have the following functions available:

- `biggie_size` which when given a regular combo returns a *biggie-sized* version.
  - `unbiggie_size` which when given a *biggie-sized* combo returns a non-*biggie-sized* version.
  - `is_biggie_size` which when given a combo, returns True if the combo has been *biggie-sized* and False otherwise.
  - `combo_price` which takes a combo and returns the price of the combo.
  - `empty_order` which takes no arguments and returns an empty order which is represented by 0.
  - `add_to_order` which takes an order and a combo and returns a new order which contains the contents of the old order and the new combo. For example, `add_to_order(1,2) -> 12`.
- (a) Write a recursive function called `order_size` which takes an order and returns the number of combos in the order. For example, `order_size(237) -> 3`.

- (b) Write an iterative version of `order_size`.

(c) Write a recursive function called `order_cost` which takes an order and returns the total cost of all the combos.

(d) Write an iterative version of `order_cost`.

(e) **Homework:** Write a function called `add_orders` which takes two orders and returns a new order that is the combination of the two. For example, `add_orders(123,234)` -> 123234. Note that the order of the combos in the new order is not important as long as the new order contains the correct combos. `add_orders(123,234)` -> 122334 would also be acceptable.

2. Give order notation for the following:

(a)  $5n^2 + n$

(b)  $\sqrt{n} + n$

(c)  $3^n n^2$

```
3. def fact(n):  
    if n == 0:  
        return 1  
    else:  
        return n * fact(n - 1)
```

Running time?

Space?

4. Write an iterative version of `fact`.

```
5. def find_e(n):  
    if n == 0:  
        return 1  
    else:  
        return 1/fact(n) + find_e(n - 1)
```

Running time?

Space?

(Assume iterative fact)

6. Assume you have a function `is_divisible(n, x)` which returns True if `n` is divisible by `x`. It runs in  $O(n)$  time and  $O(1)$  space. Write a function `is_prime` which takes a number and returns True if it's prime and False otherwise.

Running time?

Space?

7. **Homework:** Write an iterative version of `find_e`.