National University of Singapore
School of Computing
CS1010S: Programming Methodology
Semester I, 2022/2023

Side Quest 2.4
Magic Efficiency

Release date: 26th August 2022
Due: 2nd September 2022, 23:59

Required Files

- sidequest02.4-template.py

This side quest consists of **three** tasks. For each question that asks for orders of growth, simply replace the `O(...)` in the template file with your answer.

Task 1: Simplification (8 marks)

Task 1a: Simplified Big-O (3 marks)

Give the **simplified big-O** notations for all twelve expressions below.

(i) $4^n n^2$
(ii) $n3^n$
(iii) $1000000000n^2$
(iv) $2^n/1000000000$
(v) $n^n + n^2 + 1$
(vi) $4^n + 2^n$
(vii) 1^n
(viii) n^1
(ix) $(n+3)^4$
(x) $(n+4)^3$
(xi) $e^n + n^2$
(xii) $n^7 + 3^n$

Express x^y in the format `x**y` instead of `x^y`.

Task 1b: Comparing Orders (3 marks)

Determine, in each group, the **simplified big-O notation** of the expression that has the **larger** order-of-growth.

(i) $4^n n^2$ vs $n3^n$
(ii) $1000000000n^2$ vs $2^n/1000000000$
(iii) $n^n + n^2 + 1$ vs $4^n + 2^n$
(iv) $(n+3)^2$ vs $(n+2)^3$
(v) $e^n + n^2$ vs $n^7 + 2^n$
(vi) $1^n + 2^n + 3^n$ vs $n^3 + n^2 + n^1$

Task 1c: Magical Arrangement (2 marks)

Russell feels that comparing only two order of growths is boring, so he decided to compare the twelve obtained in **Task 1a** and arrange them in ascending order instead. In other words, put the smallest order of growth at #1 and the largest at #12.
However, he wants you to do it, so he doesn't have to. Can you help him just this time?

Task 2: Analysis (2 marks)

Consider the following function `foo`:

```
def foo(n):
    def bar(n):
        if n == 0:
            return 0
        else:
            return 1 + bar(n - 1)
    return n * bar(n)
```

What is the time complexity for the running time of `foo` in terms of its input n?
What about space complexity?

Task 3: Improvisation (6 marks)

Consider the following two functions:

```
def bar(n):
    if n == 0:
        return 0
    else:
        return n + bar(n - 1)

def foo(n):
    if n == 0:
        return 0
    else:
        return bar(n) + foo(n - 1)
```

(i) What is the time complexity of `bar`? What about `foo`?

(ii) What is the space complexity of `bar`? What about `foo`?

(iii) Implement the function `improved_foo` **using any method** such that it computes the same value as `foo`, but with improved efficiency.
To get full credit, your new function must have improved (smaller) order-of-growth in **both time and space**. Be sure that your function returns an `int`!

(iv) State both order-of-growths for your `improved_foo` clearly in big-O notation.