CSC110 Lecture 22: Properties of Asymptotic Growth and Basic Algorithm Running Time Analysis

Print this handout

Exercise 1: Properties of asymptotic growth

1. Recall the following definition:

   Let $f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$. We can define the **sum of** f **and** g as the function $f + g : \mathbb{N} \to \mathbb{R}^{\geq 0}$ such that

   $$(f+g)(n) = f(n) + g(n), \qquad \text{for } n \in \mathbb{N}$$

   For example, if $f(n) = 2n$ and $g(n) = n^2 + 3$, then $(f+g)(n) = 2n + n^2 + 3$.

   Consider the following statement:[1]

   $$\forall f, g : \mathbb{N} \to \mathbb{R}^{\geq 0},\ g \in \mathcal{O}(f) \Rightarrow f + g \in \mathcal{O}(f)$$

   In other words, if g is Big-O of f, then $f + g$ is no bigger than just f itself, asymptotically speaking.

   a. Rewrite this statement by expanding the definition of Big-O (twice!). Use subscripts to help keep track of the variables. This is a good exercise in writing a complex statement in predicate logic, and will help with writing the proof in the next part.

   b. Prove this statement.

      Hint: This is an implication, so you're going to *assume* that $g \in \mathcal{O}(f)$, and you want to *prove* that $f + g \in \mathcal{O}(f)$.

Exercise 2: Analysing running time (for loops)

Analyse the running time of each of the following functions, in terms of their input length n. Keep in mind these three principles for doing each analysis:

- For each for loop, determine the *number of iterations* and the *number of steps per iteration*.
- When you see statements in sequence (one after the other), determine the number of steps for each statement separately, and then add them all up.
- When dealing with nested loops, start by analyzing the inner loop first (the total steps of the inner loop will influence the steps per iteration of the outer loop).

```python
def f1(numbers: list[int]) -> None:
    for number in numbers:
        print(number * 2)
```

```python
def f2(numbers: list[int]) -> int:
    sum_so_far = 0
    for number in numbers:
        sum_so_far = sum_so_far + number

    for i in range(0, 10):
        sum_so_far = sum_so_far + i * 2

    return sum_so_far
```

```python
def f3(numbers: list[int]) -> None:
    for i in range(0, len(numbers) ** 2 + 5):
        for number in numbers:
            print(number * i)
```

Additional exercises

Review the properties of Big-O/Omega/Theta we covered in lecture today, and try proving them! You should be able to prove all of them except (5) and (6) of the *Elementary function growth hierarchy theorem*.

---

1. This statement is a simpler form of the more general "Sum of Functions" Theorem we saw in lecture.↩