Tutorial 2: Analysis of Algorithms

CAB301 - Algorithms and Complexity

School of Computer Science, Faculty of Science

Agenda

1. Recap:
   i. Theoretical Analysis of Algorithms
   ii. Big-$\mathcal{O}$ Notation
   iii. Empirical Analysis
2. Tutorial Questions
   i. Part A: Big-$\mathcal{O}$ Notation - mostly following definitions
   ii. Part B: Theoretical Analysis - efficiency class with summation
   iii. Part C: Empirical Analysis - efficiency class with experiments

Steps to Analyse an Algorithm, Theoretically

For example, let's consider an algorithm that sorts a list of numbers.

<small>

1. **Decide the parameter(s) characterising the input size.** E.g., the number of elements in the list, n.
2. **Identify the algorithm's basic operation(s)**. E.g., a swap, or a comparison of two numbers.
3. **Identify the case scenario(s) that will determine the algorithm's running time**. E.g., best, worst, or average (sorted, reversed-sorted, or random list).
4. **Set up a summation formula** for the number of times the basic operation is performed in the worst-case scenario.

Big-$\mathcal{O}$ Notation

The upper bound of an algorithm's running time, i.e., the worst-case scenario.

<small>

Consider the following nested for-loop in pseudocode:

for $i \leftarrow 0$ **to** $n-1$ **do**
  for $j \leftarrow 0$ **to** $n-1$ **do**
    $x \leftarrow x + 1$ c_1

Here, if c_1 is the time taken to execute the basic operation $x \leftarrow x + 1$, then:

$$C_{\text{worst}}(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} c_1 = c_1 \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 1 = c_1 \sum_{i=0}^{n-1} n = c_1 n^2$$

Empirical Analysis

Set up either a **counter**, or a **timer** to measure the algorithm's running time.

```
int x = 0;
int counter = 0;
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        x = x + 1;
        counter++; // Increment the counter after each basic operation
    }
}
```

Then try different values of n and find how the running time grows.