CPSC 335 – Lecture 2

Efficiency Analysis Part 1

(ADITA Ch 3.1-3.5)

Measuring Resource Use (3.2)

Resource Types:

- **Time**
- *Space*
- I/O bandwidth
- Cache misses
- Energy

Worst-Case Analysis (3.2.1)

Complexity: the amount of a **resource** consumed when an algorithm runs on a specific **instance** of a problem

Worst-case complexity: **maximum** amount of resource the algorithm may consume

- Related to **instance (= input) size**: running on bigger instances causes the algorithm to consume more resources

Worst-Case Analysis (3.2.1)

Why *worst* case?

- The algorithm *might* use up only this much resource, but maybe more (e.g. take *at least* X seconds to run)

vs

- The algorithm will *definitely* not use more than this much resource (e.g. take *at most* X seconds to run)

Which is more useful?

Complexity Functions (3.2.2)

Size measure: size of a problem's **instances** (e.g. number of elements in a list)

Complexity function: maps the **size measure** (= instance size, e.g. n, m) to the algorithm's **complexity** (= amt of resource consumed)

- Written in terms of the size measure, e.g. f(n)
- Can never be negative
- Time complexity function: T(n)

Asymptotic Notation (3.3)

O(f(n)) ≋ ***efficiency class***

$= \{g(n) \mid \exists c > 0, t \geq 0$ such that $g(n) \leq c * f(n)\ \forall n \geq t\}$

- **$g(n) \leq c * f(n)$**

  g(n) = any given instance (of size n) complexity

  f(n) = worst-case complexity

  c = **c**onstant factor, irrelevant to efficiency analysis (implementation code vs. algorithm pseudocode)

- **$\forall n \geq t$**

  t = **t**hreshold of n small enough (≥ 0) to not have same complexity behavior as all n larger than threshold

Definition of O (3.3.1)

$O(f(n)) = O(n)$?

- $O(n)$: set of all functions equivalent to $f(n) = n$
- $O(n^2)$: set of all functions equivalent to $f(n)= n^2$
- $O(f(n)) = O(n)$
  - $7n \in O(n)$
  - $2n + 3 \in O(n)$
- $O(f(n)) = O(n^2)$
  - $3n^2 \in O(n^2)$

Big Eight (3.4)

- O(1): constant
- O(log n): logarithmic
- O(n): linear
- O(n^2): quadratic
- O(n^3): cubic
- O(c^n): exponential
- O(n!): factorial

Experimental Analysis (3.5)

- **Experimental analysis:** deriving complexity functions by using scientific method to gather and analyze **observed data** (implementation running actual code)
- **Mathematical analysis:** deriving complexity functions via **models** (pseudocode) and **proofs**
- Projects = experimental analysis, homework = mathematical analysis

Experimental Analysis (3.5)

1. **Question:** What is the time efficiency class of algorithm A?
2. **Hypothesis:** Algorithm A runs in O(n) time.
3. **Prediction:** Multiple runs of A on various instance (= input) sizes n will have runtimes that follow the trend of a straight line (linear)
4. **Testing:** implement A and run the code on various input sizes, plot the runtime results
5. **Analysis:** analyze scatterplot for linear fit

Experimental Analysis (3.5)

<u>Practical considerations</u>

- **Instrumental error:** inaccuracies due to the method of measurement
  - is the instrument sensitive enough? (seconds vs milliseconds vs nanoseconds)
  - is the measurement being affected by outside circumstances? (other programs running)
- **Benchmarks**: specific inputs and measurement criteria for meaningful real-world results
  - Hard to choose good ones
- **Random instances**: easier to generate, but real-world use may have non-random instance distribution