Solving Recurrences
Advanced Algorithms & Data Structures

COMP4500/7500

July 30, 2019

Overview of today

- Admin
- Recap of last week
- Recursive algorithms
- Calculating computational complexity

Tutorials

First tutorial was last week, another one this week.

Solutions to revision exercises will become available the week after the tutorial.

Recap of last week: analysing algorithms

We use **functions of input size** to describe the:

- *best-case*,
- *worst-case* and
- *average-case*

efficiency (e.g. time, space) of algorithms.

Recap of last week: analysing algorithms

We describe and compare these functions using **asymptotic notation**:

- focus on the *rate of growth* of these functions for large inputs
- abstract away from constant factors, lower order terms to give a machine-independent comparison

We use:

- asymptotic upper bounds (O),
- lower bounds (Ω) and
- tight bounds (Θ)

Recap of last week: analysing algorithms

We use **summations** to describe the amount of work involved in loops, recursion etc.

- Need to remember some maths to help us solve these summations.

Overview of this week

- Recusive procedures are hard to analyse
- Express running time as a *recurrence* (instead of, for instance, a summation for a loop)
  Merge-sort:

$$\begin{aligned} T(n) &= \Theta(1) && \text{if } n = 1 \\ T(n) &= 2T(n/2) + \Theta(n) && \text{if } n > 1 \end{aligned}$$

- Three strategies:
  1. **Substitution**
     Use experience to guess; then prove by induction
  2. **Iteration (also called recursion-tree method)**
     Expand out the recurrence, obtaining a summation, and solve
  3. **Master method**
     Generalises the above work into 3 cases; but does not cover every case