COMP3121/9101

Algorithm Design and Analysis

Recitation 1

Introduction

Announcements

- The first set of lab problems is out. You will be able to find them on formatif.
- A general rule when tackling the lab problems: firstly, go through the material from the lectures and understand the **definitions**, as well as all of the **lecture examples**. Secondly, go through all of the recitation problems and understand how to do the recitation problems.

  Hint. *A lab problem may be a slight variation of a problem seen in either the lecture or the recitation.*
- For this term, the **tortoise** tutorials are M14A, M18A, T09A, T12A; the **hare** tutorials are M16A, T10A, T13A, T18A. You are welcome to attend any of the tortoise tutorials, hare tutorials, or both.

1 Definitions

- **The big picture**: For a given problem, there are obvious algorithms that work by simply exhausting all possibilities. For example, deciding if a given sorted list has the integer x can be done easily by checking every element of the list.
  - However, these solutions are not meaningful: they do not provide interesting insights into the problem.
  - These solutions are also quite slow, both theoretically and in practice. As the input size becomes large, the number of possibilities to check can grow very quickly.
  - Therefore, our goal is to come up with algorithms that are *provably* faster than these brute-force solutions.
- An *algorithm* is a finite sequence of instructions that can be executed without much forethought.
  - You can think of it as following a simple recipe – the instruction sequence is finite and you just need to follow what the recipe says.
- A *computational problem* is any problem that can be solved with an algorithm.
  - It is important to be specific about what you give the algorithm as *input* – this will often affect how you analyse the complexity of the algorithm!
- There are two major components when describing an algorithm: the *algorithm* and the *analysis*.
  - **Designing** an algorithm means to unambiguously describe the steps that an algorithm takes to return the output. A well-written algorithm will be clear and succinct.
  - **Analysing** an algorithm means to show that the algorithm reports the correct solution (or a correct solution, if there are several). A correct algorithm should produce the solution on *every* input of the problem. For example, if a problem asks for the *number* of items, then a correct algorithm will not return the set of items.

    Additionally, analysing an algorithm also means to analyse the running time of the algorithm. An algorithm is typically measured by the number of *primitive operations* that it takes to finish executing. Of course, we will need to be explicit about what we consider as a primitive operation. As usual, we will always assume worst-case analysis (unless we explicitly state otherwise!).

    > In this course, we will assume that all basic arithmetic operations are constant-time operations, unless otherwise specified.

2 Asymptotics

- A function f is in the class $O(g)$ if f is *bounded above* by a multiple of g eventually, i.e. for all n past some point.
  - Formally, $f(n) = O(g(n))$ if and only if there exist *constants* $n_0, C > 0$ such that $f(n) \leq C\, g(n)$ for all $n \geq n_0$.

> **A comment on notation.** $O(g(n))$, or $O(g)$, is technically speaking a *set of functions*, so it is more correct to actually say $f \in O(g)$ or $f(n) \in O(g(n))$. However, for all intents and purposes, we will use "$=$" to mean set membership.
>
> - This means that "$=$" is not symmetric in this regard; i.e. if $f(n) = O(g(n))$, this does not imply that $O(g(n)) = f(n)$. The latter has no meaning.
>
> The same applies to $\Omega(g)$ and $\Theta(g)$.

- A function f is in the class $\Omega(g)$ if f is eventually *bounded below* by a multiple of g.
  - Formally, $f(n) = \Omega(g(n))$ if and only if there exist *constants* $n_0, c > 0$ such that $f(n) \geq c\,g(n)$ for all $n \geq n_0$.
- A function f is in the class $\Theta(g)$ if f is in the class $O(g)$ *and* in the class $\Omega(g)$, i.e. it is eventually bounded above *and* below by (different) multiples of g.
  - Formally, $f(n) = \Theta(g(n))$ if and only if there exist *constants* $n_0, c, C > 0$ such that

    $$c\,g(n) \leq f(n) \leq C\,g(n)$$

    for all $n \geq n_0$.

    Exercise. *Prove this*.

For each of the following pairs of functions, decide whether $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, $f(n) = \Theta(g(n))$, or neither.

(a) $f(n) = n$; $g(n) = 100n + 100$.

(b) $f(n) = n^2 + 2n + 1$; $g(n) = n^3$.

(c) $f(n) = n^3$; $g(n) = n^2 \log_2 n$.

3 Writing Proofs

Much of the emphasis in this course is placed on communicating your algorithm and demonstrating that your algorithm is correct beyond just crunching test cases. In this exercise, we will critique some attempts at a proof – some are more wrong than others.

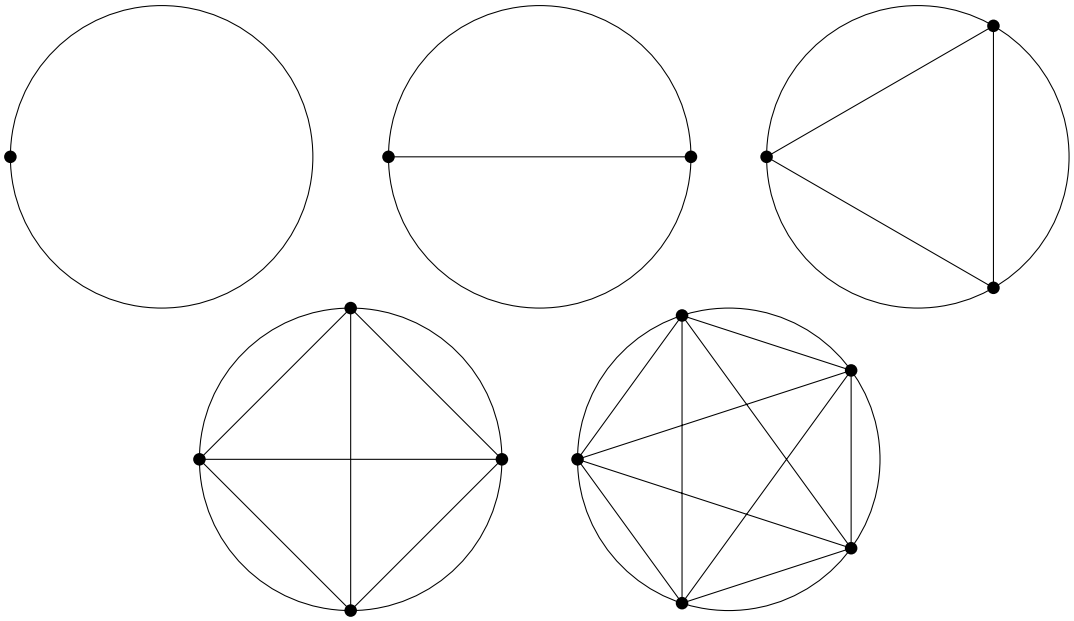
(a) **Problem.** Let $a, b \geq 0$ be non-negative integers. In the recitation, you know there are a identical students and b identical tutors. How many ways are there to arrange the a students and b tutors in a line so that all b tutors are next to each other?

Proposed solution. First, place down all of the a students in a line. There are exactly $a + 1$ possible positions to place all b tutors. Therefore, there are $a + 1$ total ways. □

(b) **Problem.** How many regions can be formed by placing n points on the circumference of a circle and drawing all the line segments between pairs of these points?

Proposed solution. Testing the first few cases, we have the following results.

Points (n)	1	2	3	4	5
Regions	1	2	4	8	16



Following this pattern, the number of regions is 2^{n-1}. □

(c) **Claim.** For $n \geq 2$, any n points are collinear (i.e. any set of n points lies on a single line).

Proposed proof. We prove this by induction on n. The base case is trivial, as any pair of points defines a unique line.

Assume now that the statement holds for all sets of $k-1$ points. We show that the statement holds for all sets of k points.

Consider k points $P_1, \ldots, P_k$. From the induction hypothesis, points $P_1, \ldots, P_{k-1}$ are collinear, as are $P_2, \ldots, P_k$. Therefore, all k of these points are collinear.

Therefore, any n points in the plane are collinear, by induction. □

Note. You may also know this as the "proof" that *all horses are the same colour*.

(d) **Claim.** All square numbers leave a remainder of 0 or 1 when divided by 3.

Proposed proof. We prove this by contradiction. Suppose the opposite, that is, all square numbers leave a remainder of 2 when divided by 3. However, 4 is a square number, and $4 \equiv 1 \mod 3$. This is a contradiction, so the claim holds true. □

4 Data Structures and Algorithms

Throughout this course, we will be using various data structures and algorithms from COMP2521 or COMP9020. It is worthwhile reviewing these; please refer to the *Review of Prerequisite Material* handout on Moodle.

In all circumstances, including this question, arrays are by default not guaranteed to be sorted.

(a) Suppose that you have an array of n integers. How can you find the largest integer using at most $n-1$ comparisons?

(b) Suppose that you have an array of $2n$ *distinct* integers. How can you find the smallest and largest

integers using at most $3n - 2$ comparisons?

Note. *Naively finding the smallest and largest integers will take* $(2n - 1) + (2n - 2) = 4n - 3$ *comparisons, which is too many. Try and eliminate some redundant comparisons.*

(c) Now, suppose you have an array of 2^n *distinct* integers. How can you find the largest two integers using at most $2^n + n - 2$ comparisons?

Hint. *Do something similar with a different data structure.*