Cosc 241
Programming and Problem Solving
Lecture 26 (30/5/2019)
Review

Lech Szymanski
lechszym@cs.otago.ac.nz

UNIVERSITY of OTAGO
SAPERE AUDE
Te Whare Wānanga o Otāgo
NEW ZEALAND

Exam format

- Eight questions, ten points per question.
- Each question has a title, indicating its theme.
- Write all answers in booklet, not on exam itself even if there seems to be space available/provided.
- Questions are typically divided into three or four parts, and points are specified for each part.
- Previous exams (particularly 2011 onwards) are a good study guide, but skip material on trees (except as related to heaps) and graphs. There will be questions on sorting and divide and conquer algorithms.

Words to the wise

A happy grader is a generous grader so:

- Start each question on a new page.
- If you don’t finish a question and plan to come back to it, leave an extra page or two.
- Be as neat and organized as possible.
- Avoid the ‘brain dump’ technique (almost all parts of questions can be answered fully in a short paragraph at most).

What do you need to know?

Everything!

- In principle, anything covered in lectures or in labs is fair game.
- In practice, most of the exam is devoted to material covered in the lectures.
- CS in general and programming in particular are cumulative subjects, so a certain amount of background is presupposed (e.g., arrays, references, methods, . . .).
- Exception to the everything rule: material from the Object Oriented Programming lectures is not on the exam.

Algorithms, recursion and algorithmic analysis

- What is an algorithm?
- How do we describe them?
- What (and why) is recursion? This links forward to recursive data structures.
- Who is the big-O? And what does it mean?
- Scales of efficiency (is $n \log n$ better than n^2?)
- Common efficiency analyses (nested loops, divide and conquer).
- Remember we aim for “best possible” O estimates. For instance a $O(n \log n)$ algorithm is also $O(n^2)$, but the former estimate is better because it imposes a more stringent upper bound.

Arrays, sorting and searching

- Using subarrays (particularly in recursive methods for array processing).
- Finding maximum values, doing swaps.
- Searching in unsorted (linear search) and sorted data (binary search).
- Sorting methods (selection, insertion, quick, merge, heap).

Random number generators and their uses

- Why are random numbers important in computing?
- What is the difference between truly random and pseudo-random numbers?
- How are pseudo-random numbers generated?
- How can we pick a winner? Shuffle a deck? Choose from a collection of known or unknown size? What are some of the efficiency issues?

ADT principles, data structures

- What is an ADT?
- What are the advantages of using ADTs?
- What is the relationship between ADT and data structure?
- Common ADTs: stack, list, queue – similarities and differences.
- What is the significance of generic types in Java data structures?

Stack and queue ADTs, linked data structures

- What is a stack? What is a queue? How are they similar? How are they different?
- How are linked data structures implemented in Java?
- One link good, two links bad? What are the issues with multiple linking?
- Room for some “hands on” material here (e.g., ‘What does this code do?’)

Divide and conquer algorithms

- What are the three phases of a divide and conquer algorithm?
- What is the complexity of a divide and conquer algorithm if the non-recursive parts require linear time? Why?
- Why might we “bail out” of the recursive part in divide and conquer on small sets of data?

Heaps, heap sort, and priority queues

- The heap data structure.
- Addition and removal algorithms.
- Using heaps for sorting (in place, $O(n \log n)$ – best possible for a comparison based sort).
- Priority queue ADT and why heap is an ideal data structure for it.