Preparing for Prelim 2.

Tuesday, 22 April 2014

5:30-7:00PM. Mallot 228, if your student ID is even
7:30-9:00PM. Kennedy, Call Aud. if student ID is odd
Review session: Sat, 19 Apr, 1-3pm, Kimball B11

If you have a conflict with the assigned time, contact
Maria: mpr13@cornell.edu. With her permission, you can
come to the other one. If you cannot make either
time, contact Maria immediately.

Five people have replied to our email about the
Passover conflict; we will contact them about taking the
prelim.

This handout explains what may be tested on prelim 2.
The course website contains several prelims from past
semesters. To prepare for the prelim, (1) practice
writing Java programs in Eclipse, (2) read the text, (3)
memorize definitions/principles, (4) study lecture slides,
and (5) study past prelims.

The overall length and balance of the exam will be
similar to past prelim 2s, but the exam will cover only
topics covered through Tuesday, 15 April. Ignore past
prelim 2 questions that touch on topics not listed below.

You are expected to know everything that was required
for Prelim1. Look at the review handout for Prelim1,
which can be found on the course website.

Here are some topics that might be covered:

1. Proofs — weak and strong induction. We may ask
you to prove things using induction (e.g. that some
simple closed form equation is the correct solution to an
iterative/recursive equation).

2. Loops and recursion. Use of invariants to develop
loops and argue about their correctness. Use of
induction to prove recursive methods correct. We used
these on searchingsorting algorithms and graph
algorithms.

3. Algorithmic complexity. Big-O complexity notation
and the associated definitions. You should understand
how to derive a big-O complexity formula for an
algorithm, best-case/worst-case/average complexity, the
notion that what this counts is some sort of "operation
we care about" and not every line of code, etc.

4. Abstract data types (ADTs) and how they can be
defined in Java (using interfaces).

5. Searching and sorting. We have covered a number
of sorting algorithms and you need to know them!
Linear versus binary search, mergesort, quicksort,
heapsort. How they work, complexity, data structures
they use. You should be able to write mergesort and
quicksort, using high-level statements for the parts that
actually massage the array (e.g. “merge sorted partitions
b[h..k] and b[k+1..n]”). Understand the min-heap data
structure, how it can be used to implement a priority
queue, and how it is used in heapsort.

6. Hashing. Hashing as presented in recitation.

7. Interfaces. Review the interface lecture materials and
make sure you understand the ideas. Be familiar with
the standard operations that are supported by common
data structures implementing Collection<T>, List<T>,
Set<T>, Map<T>, ArrayList<T>, etc.

8. Trees. Binary trees, data structure for binary and
non-binary trees, BSTs, the minimax tree of A4.

9. Graphs. Different types of graphs. Know about
depth-first and breadth-first search, topological
ordering, Dijkstra’s shortest path algorithm, spanning
trees, Kruskal’s and Prim’s algorithm. Expect questions
that involve graphs: be ready to tell us which algorithm
is the best choice for solving a problem, precisely what
that algorithm does, why it would solve a problem, and
how costly it might be.

10. GUIs. You will be not be asked to write GUI
programs, although we may ask you to read and
understand small ones. At this point you should know
about the three major classes that can contain
components (JFrame, JPanel, and Box), what their
layout managers are and how they lay out components
on the screen, and how one listens to events.

11. Keep in mind the following.

A. Being able to write correct Java code is critical.
We will continue to have a large number of points on
coding questions. We plan to grade them with a bit
more insistence on correct Java. We were relaxed about
giving partial credit for code in Matlab or Python on
prelim1. Don't expect a second "free pass" on that on
prelim2. Our graders will even catch errors like
confusion between instance and static methods, syntax
errors, unnecessarily complex code, etc. On prelim1,
you got full credit if your code was correct. On prelim2,
you might lose credit for code that is long, is inefficient,
or reveals a poor grasp of Java features.

B. We expect you to know Java —not just the bits and
pieces of Java used on slides in class. If there is some
aspect of Java that worries you, read about it in
Appendix A or look in the JavaSummary.ppt.

C. Use the powerful built-in Java tools. We give
maximum credit for concise, elegant code that doesn’t
reinvent the wheel. Know how to use standard Java
types like ArrayList, HashSet, HashMap, and know the
preexisting methods available for Collections, Arrays,
Strings, etc.