CSE15 Discrete Mathematics
Midterm Review

Angelo Kyrilov

October 16, 2017

The midterm will cover everything we have done in lectures, with more emphasis placed on Algorithms and Complexity Analysis (Sections 5, and 6 here). That is not to say that the other sections are not important, but the last two will carry more weight.

1 Logic

In this section you need to be able to generate truth tables for compound propositions. You need to be able to show that a formula is valid, satisfiable or unsatisfiable, and you need to be able to show that two formulae are equivalent. It will be important to know DeMorgan's laws ($\neg(p \wedge q) \equiv \neg p \vee \neg q$ and $\neg(p \vee q) \equiv \neg p \wedge \neg q$), and the implication elimination law ($p \rightarrow q \equiv \neg p \vee q$). Questions will be similar to Homework 1.

2 Proofs

This section will contain questions similar to those in Homework 2. These include direct proofs, proofs by contraposition, and proofs by contradiction.

3 Set Theory

Questions in this section will come from Homework 3. You need to know properties of sets such as repetition and order do not matter, determining set membership, computing Cartesian products of sets, and computing power sets.

4 Functions

This section will have questions on one-to-one functions and onto functions. You need to know the definitions of these, as well as that the inverse of a function only exists if the function is both one-to-one and onto.

5 Algorithms

There will be questions on the algorithms that we have covered in lectures. These are the divide and conquer algorithms for finding the maximum number in a list, adding up the list, searching for a specific value in a list, and sorting a list of numbers.

You need to know how each algorithm works, which means its base case and recursive step. Take a look at the labs exercises you have been doing, as well as the coding demos uploaded to CatCourses.

You may also be asked to design a divide and conquer algorithm to solve a problem that we have not talked about in class.

6 Complexity Analysis

There will be two types of questions here. The first type will test whether you are familiar with the complexity of existing algorithms. This will include best case and worst case analysis. That is to say, you will be asked to provide inputs that will force a particular algorithm to perform in its best case (as few comparisons as possible), or its worst case (as many comparisons as possible).

The second type of question in this section will test you ability to derive complexity functions and prove that they belong to a particular complexity class. You will be presented with a piece of code and will be asked to derive a complexity function for it. That is to express the number of comparisons the code is performing as a function of the input size. The algorithms presented here may be recursive, or iterative (with loops). In case of a recursive algorithm, deriving the complexity function involves figuring out how many times the function is going to call itself, and each time it is called, how many comparisons are performed. You then multiply the two numbers. For example, if a function calls itself $n-1$ times for an input of size n and each time it gets called it performs 2 comparisons, then the complexity function is $f(n) = 2n - 2$.

If the algorithm contains loops, we need to figure out how many times a lopp is repeated and how many comparisons are done in each loop. If loops are nested, we

multiply the number of times they are repeated. If they are in line with each other (the second one begins when the first one ends), then we add the number of times they repeat.

Once you have derived a complexity function, you need to prove what complexity class it belongs to. With the example above, we have that $f(n) = 2n - 2 \in O(n)$. To determine the complexity class a function belongs to, we look at the highest order term, in this case it is n^1, so the function is in $O(n)$. To prove this, we find c and n_0 such that $cn \geq 2n - 2$, for all $n > n_0$. Selecting $c = 2$ will work as long as $n > 0$, therefore $n_0 = 0$.