CS 106 MIDTERM 2 REVIEW PACKET

Spring 2020

NAME: _____

The instructions you'll also see on the midterm are shown below. For a list of *topics* that will be covered on the midterm, see the Midterm Study Guide. While example topics are given on this practice sheet, this is not an exhaustive list of all possible problems or even types of problems that may be on the exam.

1. You have **2 hours** to complete this exam. You may take the exam between the time it is posted (by Wednesday at 11:59pm EDT) and Friday at 11:59pm in any time zone (including time zones you are not currently in).
2. There are a few options (see list below) for preparing your exam work (note that it may be necessary to draw some diagrams, which could be done using a program like PowerPoint, or could be done by hand).
   (a) Print out the exam and write on the actual pages, then take pictures (I recommend CamScanner) and create a *single* PDF file to submit.
   (b) Write your answers in a word processor (NOT Eclipse or another tool for developing Java code).
   (c) Have the exam on-screen and write your answers on paper, then take pictures like the first option.
   (d) You are welcome to do a combination of approaches (e.g. write out text using a computer and then take pictures of supplementary diagrams), but make sure that you still can submit a *single* PDF file.
3. Regardless of the method you use, submit your exam as a *single* PDF file on Moodle. Unless there are extenuating circumstances with uploading your work, complete your upload within **30 minutes** of finishing your exam and marking your end time below.
4. You may use a one-page (front and back) "cheat-sheet". It must be created by you. **No other notes or resources are allowed. Even if you are using a computer to prepare some of your work, you should not have any other applications open, including web browsers, chat/video applications, Eclipse or other coding software, etc. There will be questions that call for Java code/pseudocode, but you should not be actually running/testing this code on a computer.**
5. In order to be eligible for as much partial credit as possible, show all of your work for each problem, and **clearly indicate your answers**. Credit cannot be given for illegible answers.
6. Java code that you write should be as close to correct (runnable) as possible. Small syntax errors will not cost you points, but code that is unclear *will*.
7. Choose good variable names and comment anything that might be unclear (focus on correctness first though).
8. Since this exam will be asynchronous, unfortunately I will not be able to answer questions. If you feel an exam question assumes something that is not written, write it down on your exam sheet.
9. Everything you write on these pages must be your own work, collaboration of any kind is a violation of the Honor Code.
10. This examination is **NOT** to be shared with students who take this class in subsequent years, nor circulated in any manner. Thank you for abiding by the Honor Code in respecting this restriction.

11. Sign below (or print your name on your submission) to indicate that you will **abide by the Honor Code** in taking this examination:

_____

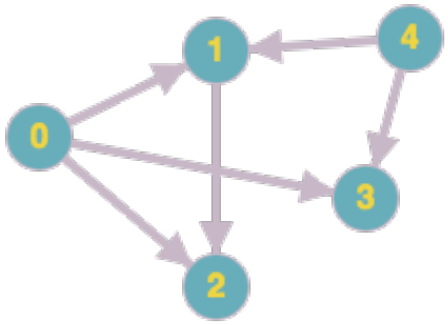
Regardless of how you submit your work, include the following information (include timezone too):

START TIME: _____

END TIME: _____

1 Running Algorithms and Data Structures

One good way to study for computer science exams is to run an algorithm and draw any associated data structures using a variety of inputs. Here are some example questions to get you started — you can make your own additional practice problems by simply changing the input examples.



a) Draw the adjacency list *and* adjacency matrix for the graph above. For each node also indicate the number of neighbors it has, and if the graph is directed. Also indicate the in-degree and out-degree of all of the nodes.

b) Insert the following items into a Binary Search Tree (left child less than parent, right child greater than parent) *in the given order*. Items to insert:

3, 1, 4, 5, 8, 2, 6, 9, 7

c) Perform pre-order, in-order, and post-order traversals on the above tree and clearly write down the results.

d) Delete the following items from the binary search tree you created above *in the given order*. Draw the tree after each item is deleted. Items to delete:

3, 8, 2, 6, 7

e) Insert the following items into a *min*-heap *in the given order*. Draw the heap after each insertion.

9, 3, 2, 5, 7, 1, 13

f) Call `removeMin` three times on the min-heap you created above. Draw the heap after each item is removed.

g) Using the hash function $h(x) = x\%7$ and a maximum hash table size of $N = 7$, create three hash tables using i) linear probing, ii) quadratic probing, and then iii) double hashing with the function $d(x) = 3 - x\%3$ and insert the following hash keys into those three tables:

35, -10, 21, 7, 0, 2

Calculate and write down the *load factor* of the hash tables after each insertion.

h) Showing your work by writing down each step of the algorithm and each new sorted list or sublist, sort the following list of numbers using i) insertion sort, ii) heap sort, iii) merge sort, and iv) quick sort:

3, 1, 6, 9, 7, 3, 8, 4

i) Make a union-find forest structure. After each operation, draw the new structure. Perform the operations below:

create(A)
create(B)
create(C)
create(D)
union(A, B)
union(C, D)
find(D)
union(A, C)
find(C)

2 Complexity

In addition to being able to run algorithms and simulate the resulting data structures, it's important to know the computational complexity of these operations and express them using big-O notation. You should be able to do this both for algorithms and data structures that you've seen before, and for those you haven't.

a) Determine the complexity of each operation or algorithm you performed when running the examples in the previous section. Since complexity depends on a specific implementation, you may need to make some assumptions about how, e.g., the trees are represented as a data structure in order to determine this. On the exam, such assumptions would be made explicitly for you.

b) Suppose that you had access to a priority queue data structure that could perform insertions and removal operations in $O(1)$ time. What would the complexity of heap-sort be if it used this new data structure instead of a heap?

3 Code

The problems below should give you an idea of what types of problems to expect, but are not an exhaustive list of possible coding questions. The below problems also expect you to write more code than would be expected on an exam (where some of the code would likely be provided for you, and you might be writing a single function). Be sure to try writing this code on paper! (It's of course fine to test your answers later by running them on the computer.)

a) Assume that you are writing an `UndirectedGraph` class that uses an underlying adjacency matrix representation for the graph. Given the index i for a vertex, write a method to determine the degree of vertex i.

b) Implement merge sort in two ways. First implement it "out-of-place", using a new data structure to merge two lists. Then implement it "in-place" using an array.

c) Implement radix sort of decimal integers, where each integer is represented as a `String` of `Characters`.