Ideas for Midterm-questions

This is *not* a sample-midterm. It is a list of the types of questions that are typically asked, but it is not necessarily including all types of questions, and not all these questions are necessarily asked. It is also not tailored to any particular term; for term-specific details such as the date, location and coverage of topics, please consult the web page and/or piazza.

In the actual midterm, the top of the front page is filled with automatically-generated information such as your name, location, seat, and the crowdmark stamps. The bottom will have some instructions much like the ones below.

Instructions

- Additional Materials Allowed: None.
- No calculators allowed.
- Proctors will not answer questions. In case of a perceived ambiguity, state your assumptions clearly.
- You may use any results from lectures without proof.
- Students may not leave the examination room during the first 60 or final 10 minutes.
- Write your answers in the space provided. You should not require extra space.
- If you do need more space, you may use the extra page at the back of the exam, and indicate clearly on the question's page that you have done so.
- Signature (pen only):

Problem	Out of
X	Y
X	Y
X	Y
X	Y
X	Y
X	Y
X	Y
X	Y
X	Y
Total	Y

1. Short-answer questions:

   (a) Which of the following statements is true? (We may or may not ask you to justify your answer).

      i. ...

   (b) The following statements are false. Say why they are false.

      i. ...

   (c) Answer the following questions in the space provided. No justification is needed.

      i. ...

   Some of these questions are very straightforward, but others might actually require deep thinking. Do *not* guess for True/False questions; in some grading schemes points will be *deducted* for incorrect answers.

2. Asymptotic Notation

   (a) What is the run-time (in terms of n and using Θ-notation) of the following code-fragments? Simplify as much as possible. (We may or may not ask you to justify your answer.)

      i. `fragment1(int n) { ... }`

   (b) For each of the following function pair, $f(n)$ and $g(n)$, indicate indicate whether or not $f(n)$ is in o, O, Ω, or ω of $g(n)$. (We may or may not ask you to justify your answer).

      i. $f(n) = XXX$, $g(n) = YYY$

3. Some Data Structure Applied

   Here is an example of Data Structure X. Apply operation Y to it.

   Data structures in cs240R typically include ordered and unordered arrays/lists, heaps, BST, AVL-trees, and Skip lists. Maybe also include Tries, Compressed Tries, and Hashing if the midterm is late. In cs240E there could also be meldable heaps, binomial heaps, Treaps, Scapegoat-trees and Splay-trees.
   This list is not tailored to any individual term; always consult the web page or piazza for the exact coverage of topics.

4. Some Algorithm Applied

   Here is an instance of Problem X. Apply algorithm Y to it.

   Algorithms in cs240 typically include MergeSort, HeapSort, QuickSelect, partition, QuickSort, BucketSort, CountSort, LSD-RadixSort, MSD-RadixSort, BinarySearch, MTFheuristic and (if covered) InterpolationSearch.
   This list is not tailored to any individual term; always consult the web page or piazza for the exact coverage of topics.

5. Some Algorithm Analyzed

   Here is an algorithm.

$mystery(A, n)$ 1. $\ldots$

   (a) What does this algorithm do?

   (b) What is its best-case/worst-case/average-case/expected run-time?

   > We will not ask you to do an analysis as hard as the one of average-case QuickSelect, but you should be able to come up with such recursions for simpler algorithms and recognize recursions that we have seen in class.

   (c) How much space/auxiliary space does it use?

6. Design

   Here is a problem P. (Description of P, perhaps with an example.) Design an algorithm/a data structure that solves P. The run-time should be $O(XXX)$, and the auxiliary space should be $O(YYY)$.

   > By default, such algorithm designs should always include an argument why it works correctly and with an analysis of why the run-time/space is as desired.

7. Critique of design

   Prof. S. Tupid thinks that he can solve problem P better than we did in class by doing X.

   (a) Explain why Prof. Tupid's idea is not good.

   > Typically the idea should contradict some lower bound or claim that we had in class. Study these and know how to prove them.

   (b) What would you need to change so that this becomes a good idea?

8. Which one is best?

   Your want to solve problem P, using one of the algorithms X_1, X_2, X_3. For each of the following situations, which algorithm would be best to use? Justify your answer (e.g. by arguing what the run-times of the algorithms would be).

   (a) Situation 1: XXX

   (b) Situation 2: YYY

There frequently is not a unique answer to these, the answer is "it depends" and you should expand on what it depends on.
Know for each algorithm and data structure what its run-time and space requirements are.
Study tradeoffs (e.g. can you make search fast if insert is slow?) and under what restrictions you can use it (e.g. small numbers for countsort).

9. Expand-that-questions

These are rarely questions on their own, but usually mixed in with some of the other questions. So after you're been asked to do something on an example, you may be asked how to do this in general. Or whether this is always possible. Or to provide an example where it can't be done. Or to analyze how long it would take for arbitrary size.
There isn't really a good way to study for these questions, other than generally understanding what we did in class and why we did it the way we did it.

How much of each kind? For cs240R, we usually aim for the following split:

- About 40% of marks are for "do" questions where you execute something we had done before on new input/code. These are questions of type 2,3,4,5.

  Note that you can study *really* well for such questions (do lots and lots of examples, and exchange answers with your friends to check).

- About 30% are questions that should be easy if you understood why we did what class. These are questions of type 1,7,8.

  To study for these, go again over your notes from class and in each step, ask 'why'. Do not just believe a time bound, convince yourself that it holds.

- About 30% are questions that require you to come up with new thoughts. These are questions of type 6 and 9.

  You have seen many questions of these types on the assignments (but the ones on the exam are usually simpler). Study old assignments (maybe also from previous terms) to get better at problem solving.

For cs240E, the percentages are closer to 20/40/40, i.e., a lot less routine-work and a lot more problem-solving.

Here is a type of question that we usually don't ask: "Regurgitate some result that we had in class." For example, we would not ask you to write down exactly the proof that AVL-trees have height $O(\log n)$.
Having said that, you should know the main idea of such proofs, because we might ask you for a similar (simpler) proof where knowing the one from class probably helps.

(This page is intentionally left blank. You can use it if you need more space for some other question, but clearly indicate on that question's page that you have done so.)

The exam will have at least one such page, for overflow-work.

A few general exam-tips

Most of the following are not cs240-specific, and should be familiar to you. But, just in case...

- Don't get stuck on a question! The questions are usually ordered by topic, not by difficulty. In a first pass through the exam, answer all the easy question first (especially the "do" questions).

  Read over the "think" question (including the bonus question, if any), but don't spend more than a minute on it unless you're quite sure what to do. Quite possibly an inspiration might form later.

  Once done with the "do" question, attack the remaining ones in the order in which you feel comfortable with them. Leave the bonus question (if any) to the end.

- It is nearly impossible to write too much. If in doubt, *always* write justifications and show how you obtained the answer. For many questions just showing the final correct answer is enough. But if your answer is incorrect, it helps to have shown your work—partial marks may be awarded if your work shows that you understood the concept. Even just writing down what needs to be done (even if you don't know exactly how to do it) may sometimes give you a pity-mark. No promises though.

- In a similar spirit, do not erase what you have written, unless you really really need the space (and you are quite sure that you have a better solution). Too often we see erased solutions that might have given you some partial marks, but since they were erased they cannot be counted.

  With the exception of True/False questions, the worst that can happen if you don't erase a wrong solution is that you get 0 marks. If you erase it (and don't write anything else) you'll definitely get 0 marks. So don't erase.

- Write your assumptions. Often we'll have seen different variants of a data structure / algorithm in class. If you're not sure which one is the "standard" one, then write what you're assuming, and as long as the question doesn't get simpler in the process there won't be a deduction for making the assumption.

  Even if the question gets simpler, writing your assumption is better than not writing it. At least then we understand why you wrote your answer, which makes it more likely that you'll get part marks.

- When drawing data structures, aim to use the same drawing conventions as we used in class. However, as a general rule, other ways of drawing will be accepted as long as all relevant information is displayed. (If in doubt, write what denotes what, i.e., write your assumptions.)