CSCE 411: Design and Analysis of Algorithms

Week 5, Exam 1 Review

Date: February 10, 2026 *Nate Veldt, updated by Samson Zhou*

1 Test format and instructions

- 12 multiple choice questions, 3 long answer questions
- You do not need to prove something unless you are explicitly asked to prove something
- You will be asked to explicitly prove something
- Scratch paper will be provided
- No calculators or electronic devices. You will not need them. If you have to do any computations, they will be basic enough to do by hand.
- You will put your name on the front page, and put your initials at the top of every other page.
- You have been provided with a "cheat sheet" with notes that you can use on the test without having to memorize it (including the Master Theorem among other things.)

2 Topics Covered on the Exam

2.1 Algorithm basics

For the whole test, you should be familiar with

- O notation, Θ notation, Ω notation, and the differences among them
- Basic matrix multiplication
- Basic proof strategies (i.e., induction, contradiction)

2.2 Divide-and Conquer

Conceptual questions

- What properties should hold in order to apply a divide-and-conquer approach?
- What are the three steps of a divide-and-conquer method?
- What strategies are there for proving a runtime from a runtime recurrence relation?
- What parts of the recurrence relation correspond to the three steps of divide and conquer?

Specific problems to know about

- Merge-sort
- Strassen's Algorithm for matrix multiplication

Possible types of sample questions (non-exhaustive!)

- Use different methods to prove runtime guarantees from recurrence relations (e.g., apply the Master Theorem to give a runtime from a recurrence)
- Find the recurrence related satisfied by an algorithm
- Analyze the runtime of the combine step of a divide-and-conquer method
- Provide pseudocode for the combine step of merge sort

2.3 Dynamic Programming

Conceptual Questions

- What properties should hold in order to apply a dynamic programming approach?
- How is dynamic programming similar and different from divide-and-conquer?
- What are two different approaches for writing code for a dynamic programming algorithm?

Specific problems to know about

- Fibonacci
- Rod-cutting
- Matrix Chain Multiplication
- Min cost path in a grid

Possible types of sample questions (non-exhaustive!)

- Answer some variant of the conceptual questions asked above
- Analyze a variation or application of one of the three problems we considered
- Provide pseudocode for a dynamic programming version of an algorithm (could be either version of dynamic programming)
- Identify when dynamic programming is right for a problem
- Give a recurrence relationship for a dynamic programming algorithm
- Analyze a runtime based on a recurrence relationship
- Prove/explain why overlapping subproblems holds for a certain problem

2.4 Greedy Algorithms

Conceptual Questions

- What properties should hold in order to apply a greedy algorithm?
- How is this similar to and different from other algorithmic paradigms?
- What is an overall strategy for applying greedy approach to a problem?

Problems to know about

- Activity subset selection problem

- The coin change problem
- Optimal prefix code problem
- Fractional knapsack problem

Possible types of sample questions (non-exhaustive!)

- Provide or analyze pseudocode for a greedy algorithm, e.g., coin change, activity subset selection
- Prove that a greedy strategy is optimal for a certain type of problem
- Prove that a greedy strategy is *not* optimal
- Identify when a greedy algorithm is right for a problem

2.5 Amortized Analysis

Conceptual Questions

- What is the definition of amortized analysis?
- How does it differ from other runtime analyses?
- What three strategies did we cover for amortized analysis, and how does each one work? What do you have to choose/set/show for each one?
- How are different strategies different?

Problems to know about

- Multipop stack
- Binary counter increment problem

Possible types of sample questions (non-exhaustive!)

- Prove that a certain choice of amortized costs is bounded above
- Prove that a certain choice of amortized costs bounds the actual cost
- Identify the right potential function using the potential method
- Perform an aggregate analysis of an algorithm

3 Tips on How to do a Runtime Analysis

- **Simple Iterative Algorithms**
  - Bound number of iterations
  - Bound runtime for each iteration, multiply by number of iterations
- **Divide and conquer**
  - Obtain a recurrence relationship
  - Confirm that subproblems do not overlap
  - Apply Master Theorem, or another approach
- **Dynamic programming**
  - Prove recurrence relationship between subproblems
  - Confirm problems overlap
  - Design an implementation that only solves each subproblem once. Given that each problem is only solved once, what is the overall runtime cost?
- **Amortized Analysis, for iterative method**
  - Aggregate analysis: bound total cost, divide by number of iterations (rather than multiplying number of iterations by maximum iteration cost)
  - There are two other methods...see practice problems and lecture notes for overview.

4 Practice Problems

1. What are the three steps of a divide-and-conquer method? Given examples.

2. Describe what steps are needed to prove an amortized runtime analysis using the potential method and the accounting method.

3. What two properties should be satisfied by a problem in order to use a greedy method?

4. What is the strategy we have seen for proving that a greedy method is optimal?

5. Consider the following recurrence:

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and SPECIAL CONDITION holds} \\ \max\{c[i, j-1], c[i-1, j]\} & \text{if } i, j > 0 \text{ and SPECIAL CONDITION does not hold.} \end{cases}$$

   where $c[i,j]$ is a subproblem of a problem that can be solved via dynamic programming, and is defined for every i and j such that $1 \leq i \leq g$ and $1 \leq j \leq p$. What is the runtime for computing $c[g,p]$ if it takes $O(1)$ time to check SPECIAL CONDITION?

6. Consider the following recurrence relations

$$T(n) = 4T(n/4) + n$$
$$T(n) = 7T(n/2) + n^2$$
$$T(n) = 3T(n/2) + n^5 \log n$$
$$T(n) = 3T(n/3) + n \log n$$

   What runtime for this method does the Master Theorem give in each case? Make sure you are aware of situations where the Master Theorem does not apply. You can find other versions of the Master Theorem from other sources, but we will restrict to using the version in our textbook (as presented in class).

7. Consider a data structure that maintains a stack of elements together with a counter. The data structure supports the following operations:

   - PUSH(x): pushes element x onto the stack.
   - POP(): removes the top element of the stack.
   - FLUSH(): removes *all* elements currently in the stack.

   The costs of operations are defined as follows:

- PUSH(x) takes $O(1)$ time.
- POP() takes $O(1)$ time.
- FLUSH() takes time proportional to the number of elements currently in the stack.

The stack is initially empty. Let $T(m)$ denote the total running time of any sequence of m operations.

1. Give the worst-case running time of each operation.
2. Explain why the worst-case bound on FLUSH() does not imply that a sequence of m operations takes $\Theta(m^2)$ time.
3. Perform an amortized analysis to show that for any sequence of m operations, the total running time $T(m)$ is $O(m)$.
4. Clearly define a potential function and use it to justify your analysis.
5. State the amortized cost of each operation.

Reviewing other homework problems is the next best way to prepare!