CSC 411 Final Review Guide

Design and Analysis of Algorithms

Spring 2026

1 Asymptotic Analysis Foundations

What you should know well

- The formal definition of $O(g(n))$. You should be able to explain conceptually how Ω and Θ differ from O.
- Dominant-term reasoning: constants and lower-order terms do not change asymptotic class.
- How to choose an input size parameter.
- How to identify a basic operation and count it systematically.

Skills to practice

- Turn nested loops into summations and simplify.
- Combine algorithm phases correctly (e.g., $\Theta(n \log n) + \Theta(n) = \Theta(n \log n)$).

Common mistakes

- mixing up O and "worst-case" as concepts.
- assuming that O only applies to run-time, not to other measures (e.g., space, comparisons) or general functions.
- Reporting only big-O when the question asks for tight bounds (lower and upper).

2 Recursive Analysis and Recurrences

Master process

1. Define input size and basic operation.
2. Write recurrence + base case.

3. Solve by algebra or Master Theorem.

4. State asymptotic result with brief justification.

Master Theorem

For recurrences of the form

$$T(n) = aT(n/b) + cn^d,$$

compare a with b^d:

- $a < b^d$: $T(n) = \Theta(n^d)$,
- $a = b^d$: $T(n) = \Theta(n^d \log n)$,
- $a > b^d$: $T(n) = \Theta\left(n^{\log_b a}\right)$.

3 LLM Inference

Components and Complexities of a Language Model

- You should know the computations performed in the transformer **attention** module.
- You should be able to explain why attention is a quadratic operation.

KV Caching

- You should be able to identify the redundant computations performed during transformer decoding.
- You should be able to explain how caching K and V improves the asymptotic complexity of attention during decoding.

4 Dynamic Programming and Greedy Algorithms

Shortest Paths in DAGs

- You should be able to explain the overlapping subproblems that make shortest path (to a target) solvable by dynamic programming.
- You should understand the recurrence used to define the optimal value table:

$$v[i] = \min_{i \to j}(w_{i,j} + v[j])$$

- You should be able to explain why a greedy approach starting at a source in the DAG usually won't find shortest paths.

Edit Distance

- You should be able to explain the overlapping subproblems that make edit distance solvable by dynamic programming.
- You should understand the recurrence used to define the optimal value table:

$$D[i,j] = \begin{cases} j & \text{if } i = 0, \\ i & \text{if } j = 0, \\ \min \begin{cases} D[i-1,j]+1 & \text{(deletion)}, \\ D[i,j-1]+1 & \text{(insertion)}, \\ D[i-1,j-1] + \begin{cases} 0 & \text{if } x_i = y_j, \\ 1 & \text{if } x_i \neq y_j \end{cases} & \text{(match/substitution)} \end{cases} & \text{otherwise.} \end{cases}$$

- You should be able to find the edit distance between two strings using the tabular approach we covered in class.

Knapsack Problems

- You should be able to explain the difference between fractional knapsack and 0-1 knapsack problems.
- You should be able to explain the overlapping subproblems that make 0-1 knapsack problems solvable by dynamic programming.
- You should understand the recurrence used to define the optimal value table:

$$V[i,c] = \begin{cases} 0 & \text{if } i = 0 \text{ or } c = 0, \\ V[i-1,c] & \text{(exclude item } i \text{ if } w_i > c), \\ \max \left\{ \underbrace{V[i-1,c]}_{\text{exclude } i}, \underbrace{V[i-1,c-w_i]+v_i}_{\text{include } i} \right\} & \text{if } w_i \leq c. \end{cases}$$

- You should be able to find the edit distance between two strings using the tabular approach we covered in class.

Huffman Coding

- Given an alphabet $\mathcal{A}$ with frequencies, you should be able to apply Huffman's algorithm to find an optimal code for $\mathcal{A}$.

5 Formal Reductions, Decision Problems, and Complexity Classes

Decision-problem viewpoint

- A decision problem has YES/NO answers.
- Encode each input as a binary string.
- A problem corresponds to a language $L \subseteq \{0,1\}^*$ containing exactly the YES instances.

Reduction Relation

- You should be able to explain what $Y \leq_P X$ means.
- You should know that $\leq_P$ is transitive.

Classes and completeness

- You should be able to define the class P.
- You should be able to define *efficient verifier*.
- You should be able to define the class NP in terms of efficient verifiers.
- You should be able to define NP-completeness.
- You should be able to prove that if an NP-complete problem X is solvable in polynomial time, then $P = NP$.

Boolean Circuits

- You should be able to evaluate a boolean circuit K.

Reductions for NP-Completeness

- You should be able to reduce a 3-SAT instance to a SubsetSum instance.
- You should be able to reduce a SubsetSum instance to a 0-1 Knapsack instance.