CSC 411 Midterm Review Guide

Design and Analysis of Algorithms

Spring 2026

How to Use This Review

This guide is organized by topic clusters. For each topic, study:

- core definitions and theorems,
- standard derivations and proof ideas,
- common algorithm templates,
- pitfalls that cause avoidable errors on timed exams.

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 (usually n, but justify it).
- How to identify a basic operation and count it systematically.
- How to compare growth rates: polynomial vs logarithmic vs exponential vs factorial.

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.
- Mixing up worst-case and average-case claims.
- Reporting only big-O when the question asks for tight bounds (lower and upper).
- Dropping non-dominant terms too early before algebra is done.

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 pattern

For recurrences of the form

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

compare a with b^d:

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

Must-know recurrences

$$T(n) = 2T(n/2) + cn \Rightarrow \Theta(n \log n)$$
$$T(n) = T(n-1) + cn \Rightarrow \Theta(n^2)$$
$$T(n) = 3T(n/2) + cn \Rightarrow \Theta\left(n^{\log_2 3}\right)$$
$$T(n) = T(n-1) + 1 \Rightarrow \Theta(n)$$

Pitfalls

- Using Master Theorem on recurrences not in proper form.
- Forgetting to include combine cost (the nonrecursive work).
- Giving only intuition without the recurrence itself.

3 Sorting and Divide-and-Conquer

Insertion Sort

- Best case (already sorted): $\Theta(n)$.
- Worst case (reverse sorted): $\Theta(n^2)$.
- In-place and low-overhead for small n.

Merge Sort

- Recurrence: $T(n) = 2T(n/2) + \Theta(n)$.
- All cases: $\Theta(n \log n)$.
- Understand practical overhead: allocation, copying, cache behavior.

Quick Sort

- Best/average: $\Theta(n \log n)$.
- Worst (poor pivot splits): $\Theta(n^2)$.
- Be able to execute one partition pass by hand.

Empirical behavior to explain

- Why two $O(n^2)$ algorithms can differ substantially in runtime.
- Why crossover points exist (constants vs asymptotics).
- Why two $\Theta(n \log n)$ implementations can have large timing gaps.

4 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.

Reductions

Problem A reduces to problem B if there is a function f such that

$$x \in A \iff f(x) \in B.$$

Intuition: solve A by transforming input to B and solving B.

Resource-bounded reductions

- **Polynomial-time reduction:** f computable in polynomial time.
- **Log-space reduction:** f computable using $O(\log n)$ memory.

The reduction type determines the hardness notion you are using.

Classes and completeness (high-level)

- **P**: decision problems solvable in polynomial time.
- **NP**: decision problems whose YES instances are polynomial-time verifiable.
- A problem is *complete* for class $\mathcal{C}$ if:
  - it is in $\mathcal{C}$, and
  - every problem in $\mathcal{C}$ reduces to it.

What you should be able to articulate

- Why optimization problems are often studied via decision versions.
- Why complete problems are interpreted as the hardest problems in a class.

5 Brute Force and Exhaustive Search

Brute-force string matching

- Worst-case comparisons: $(n - m + 1)m = \Theta(nm)$.

Geometric brute-force algorithms

- Closest pair brute force: compare all pairs, $\Theta(n^2)$.
- Convex hull brute force (same-side test on all pairs): $\Theta(n^3)$.

Exhaustive search counts

- TSP tours (up to rotation/reversal): $(n-1)!/2$.
- Knapsack subsets: 2^n.
- Assignment problem: $n!$ permutations.

6 Graph Traversal, DAGs, and Topological Sorting

DFS vs BFS operationally

- DFS uses a stack (explicit or recursion stack), explores depth-first.
- BFS uses a queue, explores in layers by distance from source.

What BFS guarantees

- In unweighted graphs, BFS finds shortest paths in number of edges.
- DFS does *not* guarantee shortest paths.

Why visited marking matters

Without visited checks, traversal on even small cyclic graphs can fail to terminate (re-enqueue/revisit forever).

Directed vs undirected BFS

Core BFS logic does not change; only neighbor interpretation changes (for directed graphs, follow outgoing edges).

Topological sorting knowledge checklist

- Source-removal method: repeatedly remove indegree-0 vertices.
- Runtime with adjacency lists: $\Theta(n+m)$; adjacency matrix: $\Theta(n^2)$.

7 Large Integer Multiplication

Grade-school vs Karatsuba

- Grade-school multiplication: $\Theta(n^2)$ single-digit multiplications.
- Karatsuba reduces 4 half-size products to 3:

$$T(n) = 3T(n/2) + \Theta(n) = \Theta\left(n^{\log_2 3}\right).$$

8 Checklist: What you should be able to do without notes

Ask yourself the following questions. If you can answer all of them confidently in the affirmative without looking at notes, you are in good shape for the midterm. If not, identify which topics you need to review and practice more.

- I can define and distinguish O, Ω, Θ.
- I can solve loop/summation complexity quickly and cleanly.
- I can write and solve standard recurrences using the Master Theorem.
- I can execute one partition pass of quicksort correctly.
- I can explain why BFS gives shortest paths in unweighted graphs.
- I can run DFS/BFS/topological sorting by hand on small graphs.
- I can define the classes **P** and **NP** and explain the difference between them in one sentence.