St. Francis Xavier University
Department of Computer Science

CSCI 550: Advanced Algorithms
Individual Lecture and Report
Fall 2026

The individual lecture and report are major assessments of this course. These assessments consist of three components: a lesson plan, the lecture itself, and the report.

The lesson plan is worth a total of 10% of your final grade. The lecture component is worth a total of 20% of your final grade. The report is worth a total of 10% of your final grade. You must complete all of these components in order to pass the course.

Do not leave these assessments until the last minute! Start as soon as possible!

Lesson Plan

Due October 30, 2026 in lecture

The lesson plan is a short document meant to establish your chosen lecture topic.

The lesson plan should include, at a minimum, the following information:

- Your name
- One or two paragraphs giving a high-level introduction to your lecture topic
- A broad outline of the structure of your lecture (e.g., X minutes for introduction, Y minutes for background, etc.)
- A list of preliminary references that you will use to create your lecture materials

A title page is not required.

Existing survey articles or books are great preliminary references, as they often include a large amount of background information presented at an accessible level. If you need pointers to such preliminary references, let me know and I can try to help.

All references must be cited in a consistent style (e.g., IEEE citation style). Citations to articles and books are preferred over citations to lecture notes, Wikipedia, or websites.

Your lesson plan should be a 1–2 page double-spaced document written in 12-point text. This is a strict page limit, but references do not count toward the limit. The lesson plan will be marked in terms of completeness, organization, and adherence to guidelines.

Lecture

Due date varies (last ∼3 weeks of lectures)

The lecture is a ∼20 minute presentation on your chosen topic. You will not be presenting for the entire length of time; you should budget ∼5 minutes for questions and teardown at the end. Therefore, your actual

lecture time will be closer to ∼15 minutes.

Your lecture should be presented at a level similar to other lectures in this course: targeted at a graduate student audience that is familiar with theoretical computer science. You can refer to past lecture slides as background material, but you should ultimately develop your own notes or slides for use in the lecture itself.

You must submit your lecture materials (e.g., notes, slides, etc.) *at least one day before* the lecture is scheduled. Your lecture will be marked in terms of three broad categories: quality (e.g., focused on topic, targeted at an appropriate level), accuracy (e.g., correct summary of main results, material all related to topic), and organization (e.g., notes/slides are clearly written, lecture is well-paced).

All lecture material must be appropriately referenced, and citations must be included wherever necessary.

Lastly, remember that *quality is better than quantity.* A very well-produced lecture that takes ∼12 minutes is better to experience than a poorly-planned lecture that takes ∼20 minutes.

Report

Due December 7, 2026

The report is a submission meant to complement your lecture. In this report, you will write a brief survey of your lecture topic *in your own words* and give a short reflection on your lecture experience.

On the first page of your report, you should include, at a minimum, the following information:

- Your name and the topic of your lecture
- Approximately 1-2 paragraphs discussing aspects of your lecture that you feel went well and aspects of your lecture that you feel could have been improved.

In the rest of your report, you should include the following information:

- 4–5 pages summarizing your lecture topic, written in the style of a survey article, short lecture notes, or Wikipedia article
- A complete list of references used in your report (not in your lecture in general)

You must additionally include a title page that lists your name, report topic, and course details.

All references must be cited in a consistent style (e.g., IEEE citation style). Citations to articles and books are preferred over citations to lecture notes, Wikipedia, or websites.

Your report should be a 5–6 page double-spaced document written in 12-point text. This is a strict page limit, but references and the title page do not count toward the limit. The report will be marked primarily in terms of completeness, organization, and adherence to guidelines.

Since the report is due at the end of the term, the report deadline is firm. Late reports will not be accepted.

Suggested Topics

Below, I offer a selection of topics that you can choose for your lecture. If you wish to study and present a topic related to this course that is not on this list, you must check with me first to approve the proposed topic.

1. **Dynamic graph algorithms.**

   Many classical graph algorithms assume that the input graph remains fixed throughout the computation. However, some applications involve graphs whose vertices or edges change over time. The goal of this lecture is to introduce dynamic graph algorithms, discuss algorithms for problems such as dynamic connectivity or shortest paths, and examine how their computational complexity compares with recomputing solutions from scratch.

2. **Streaming algorithms.**

   Traditional algorithms often assume that the entire input is available in memory. Streaming algorithms instead process data as it arrives, typically using only a small amount of memory. The goal of this lecture is to introduce streaming algorithms, discuss techniques for problems such as counting distinct elements or identifying frequent elements, and examine the tradeoffs between memory usage, accuracy, and computational efficiency.

3. **Parallel algorithms.**

   Parallel algorithms attempt to solve computational problems by distributing work across multiple processors. However, the efficiency of a parallel algorithm depends on more than just the number of available processors. The goal of this lecture is to introduce models of parallel computation, discuss parallel algorithms for problems such as sorting or searching, and examine measures of parallel efficiency such as work, depth, and speedup.

4. **Distributed algorithms.**

   In distributed computing, multiple computers communicate to solve computational problems, often without shared memory or global information. The goal of this lecture is to introduce distributed algorithms, discuss algorithms for problems such as leader election or spanning tree construction, and examine the importance of communication complexity and synchronization in algorithm design.

5. **External memory algorithms.**

   Traditional algorithm analysis generally assumes that accessing any location in memory takes constant time. However, accessing data stored on external devices can be substantially more expensive than accessing main memory. The goal of this lecture is to introduce external memory algorithms, discuss techniques such as external memory sorting, and examine how minimizing data transfers can improve algorithmic efficiency.

6. **Sublinear-time algorithms.**

   Most familiar algorithms require at least linear time to examine their entire input. However, certain computational problems can be solved, either exactly or approximately, without examining every input element. The goal of this lecture is to introduce sublinear-time algorithms, discuss examples involving property testing or approximate computation, and examine the assumptions and limitations associated with sublinear running times.

7. **Computational geometry.**

   Computational geometry concerns the design and analysis of algorithms involving geometric objects such as points, lines, and polygons. These problems have applications in computer graphics, robotics, and geographic information systems. The goal of this lecture is to introduce algorithmic techniques in computational geometry, such as sweep line algorithms and geometric divide-and-conquer, and discuss algorithms for problems such as convex hull construction and finding closest pairs of points.

8. **Advanced string algorithms.**

   Efficient string processing is essential in applications such as text searching, bioinformatics, and data compression. While basic string matching algorithms can locate occurrences of a pattern, more advanced algorithms and data structures can support considerably more sophisticated operations. The goal of this lecture is to introduce advanced string algorithms, discuss data structures such as suffix trees or suffix arrays, and examine their applications and computational complexity.

9. **Algebraic algorithms.**

   Many computational problems can be expressed using algebraic operations, allowing mathematical techniques to improve the efficiency of their solutions. For example, polynomial multiplication and matrix multiplication can be performed more efficiently than their straightforward algorithms might suggest. The goal of this lecture is to introduce algebraic algorithm design and discuss an important example, such as the fast Fourier transform or fast matrix multiplication, together with its analysis.

10. **Spectral graph algorithms.**

    Many properties of graphs can be studied using matrices, such as adjacency matrices and graph Laplacians. The eigenvalues and eigenvectors of these matrices can be used to develop algorithms for various graph problems. The goal of this lecture is to introduce the basic ideas underlying spectral graph algorithms, discuss an application such as graph partitioning or clustering, and examine the computational complexity of the techniques involved. Some familiarity with linear algebra is recommended.

11. **Succinct data structures.**

    Traditional data structures are generally designed to support efficient operations, but their memory requirements can be substantial. Succinct data structures attempt to represent information using space close to the theoretical minimum while maintaining efficient operations. The goal of this lecture is to introduce succinct data structures, discuss examples involving bit vectors, trees, or strings, and examine tradeoffs between storage requirements and computational efficiency.

12. **Algorithmic game theory.**

    Traditional algorithm design often assumes that a computational problem has a single objective to optimize. However, many applications involve multiple participants whose individual objectives may conflict. Algorithmic game theory examines the design and analysis of algorithms in such settings. The goal of this lecture is to introduce algorithmic game theory, discuss computational problems involving strategic behaviour, and examine concepts such as Nash equilibria, selfish routing, or the price of anarchy.

13. **Convex optimization algorithms.**

    Many computational problems involve minimizing or maximizing an objective function subject to various constraints. While linear programming addresses an important class of optimization problems, more general problems may involve nonlinear functions. The goal of this lecture is to introduce convex optimization, discuss algorithms such as gradient descent or Newton's method, and examine their convergence properties and computational complexity.

14. **Fine-grained complexity.**

    Traditional complexity theory distinguishes between problems that admit polynomial-time algorithms and those believed to require superpolynomial time. However, this distinction does not necessarily tell us whether an existing polynomial-time algorithm can be significantly improved. The goal of this lecture is to introduce fine-grained complexity, discuss hypotheses such as the Strong Exponential Time Hypothesis, and examine how conditional lower bounds can be established for familiar computational problems.

15. **Cache-oblivious algorithms.**

    The efficiency of an algorithm can depend heavily on how effectively it uses the memory hierarchy of a computer. While external memory algorithms explicitly account for the sizes of memory blocks and caches, cache-oblivious algorithms attempt to achieve efficient memory access patterns without requiring knowledge of these parameters. The goal of this lecture is to introduce cache-oblivious algorithms, discuss techniques such as recursive matrix multiplication or cache-oblivious sorting, and examine how their performance compares with traditional algorithms.

16. **Branch-and-bound algorithms.**

    Many optimization problems are computationally intractable, requiring exponential time in the worst case. Branch-and-bound algorithms attempt to solve such problems exactly by systematically exploring possible solutions while eliminating portions of the search space that cannot contain an optimal solution. The goal of this lecture is to introduce branch-and-bound algorithms, discuss their application to problems such as the travelling salesperson problem or integer programming, and examine how bounding techniques and branching strategies influence computational efficiency.