CSE 531C: Algorithm Analysis and Design (Fall 2023)

Introduction and Syllabus

Lecturer: Kelin Luo

Department of Computer Science and Engineering
University at Buffalo
Outline

1. Syllabus

2. Introduction
   - What is an Algorithm?
   - Example: Insertion Sort
   - Analysis of Insertion Sort

3. Asymptotic Notations

4. Common Running times
Course Webpage (contains schedule, policies, and slides): https://cse.buffalo.edu/~kelinluo/teaching/cse531C-fall23/index.html

Please sign up course on Piazza via link https://piazza.com/buffalo/fall2023/cse531c on course webpage
- homeworks, solutions, announcements, polls, asking/answering questions

Acknowledgement: The course design and information primarily draw inspiration from Prof. Shi Li’s Algorithm Analysis and Design course in Fall 2022.
CSE 531C: Algorithm Analysis and Design

- **Time & Location**: Mon-Wed-Fri, 4:00pm - 4:50pm, Nsc 201
- **Instructor**: Kelin Luo, kelinluo@buffalo.edu
- **TAs**:
  - Davoud Moradi, davoudmo@buffalo.edu
  - Xiaoyu Zhang, zhang376@buffalo.edu
  - Xuelu Feng, xuelufen@buffalo.edu
  - Ibrahim Bahadir Altun, ialtun@buffalo.edu
- **Office hour**
You **should** already have/know:
You should already have/know:

- **Mathematical Background**
  - basic reasoning skills, inductive proofs
You should already have/know:

- Mathematical Background
  - basic reasoning skills, inductive proofs

- Basic data Structures
  - linked lists, arrays
  - stacks, queues
You should already have/know:

- Mathematical Background
  - basic reasoning skills, inductive proofs

- Basic data Structures
  - linked lists, arrays
  - stacks, queues

- Some Programming Experience
  - e.g. Python, C, C++ or Java
You Will Learn

- Classic algorithms for classic problems
  - Sorting, shortest paths, minimum spanning tree, ...
You Will Learn

- Classic algorithms for classic problems
  - Sorting, shortest paths, minimum spanning tree, ···
- How to analyze algorithms
  - Correctness
  - Running time (efficiency)
You Will Learn

- Classic algorithms for classic problems
  - Sorting, shortest paths, minimum spanning tree, · · ·

- How to analyze algorithms
  - Correctness
  - Running time (efficiency)

- Meta techniques to design algorithms
  - Greedy algorithms
  - Divide and conquer
  - Dynamic programming
  - · · ·
You Will Learn

- Classic algorithms for classic problems
  - Sorting, shortest paths, minimum spanning tree, ⋯

- How to analyze algorithms
  - Correctness
  - Running time (efficiency)

- Meta techniques to design algorithms
  - Greedy algorithms
  - Divide and conquer
  - Dynamic programming
  - ⋯

- NP-completeness
# Tentative Schedule

- 50 Minutes/Lecture × 41 Lectures

<table>
<thead>
<tr>
<th>Course</th>
<th>Lectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>3 lectures</td>
</tr>
<tr>
<td>Graph Basics</td>
<td>4 lectures</td>
</tr>
<tr>
<td>Greedy Algorithms</td>
<td>6 lectures</td>
</tr>
<tr>
<td>Divide and Conquer</td>
<td>6 lectures</td>
</tr>
<tr>
<td>Dynamic Programming</td>
<td>8 lectures</td>
</tr>
<tr>
<td>Graph Algorithms</td>
<td>7 lectures</td>
</tr>
<tr>
<td>NP-Completeness</td>
<td>4 lectures</td>
</tr>
<tr>
<td>Final Review</td>
<td>3 lectures</td>
</tr>
</tbody>
</table>
Textbook (Highly Recommended):

- Algorithm Design, 1st Edition, by Jon Kleinberg and Eva Tardos

Other Reference Books

Highly recommended: read the correspondent sections from the textbook (or reference book) before classes.

Sections for each lecture can be found on the course webpage.

Slides are posted on course webpage. They may get updated after the classes.

In last lecture of a major topic (Greedy Algorithms, Divide and Conquer, Dynamic Programming, Graph Algorithms), I will discuss exercise problems, which will be posted on the course webpage before class.
Grading

- **5% for participation**
  - In-class discussions or quizzes will be given randomly. (We choose the best 5 scores out of 8-10 quizzes.)

- **40% for theory homeworks**
  - 8 points × 5 theory homeworks (We choose the best 5 scores out of 6 homeworks.) (Recommendation: typed submissions, e.g. latex.)

- **20% for programming projects**
  - 10 points × 2 programming assignments

- **35% for final exam (closed-book, closed-note)**
For Homeworks, You Are Allowed to

- Use course materials (textbook, reference books, lecture notes, etc)
- Post questions on Piazza
- Ask me or TAs for hints
- Collaborate with classmates
  - Think about each problem for enough time before discussions
  - **Must write down solutions on your own, in your own words**
  - Write down names of students you collaborated with
For Homeworks, You Are Not Allowed to

- Use external resources
  - Can’t Google or ask questions online for solutions
  - Can’t read posted solutions from other algorithm course webpages
- Copy solutions from other students
- Use of Artificial Intelligence Technologies like OpenAI’s ChatGPT, Google Bard, and AI models within search interfaces like Google or Bing, etc.
For Programming Projects

- Use Python3
- Need to implement the algorithms by yourself
- Can not copy codes from others or the Internet
- We use Moss (https://theory.stanford.edu/~aiken/moss/) to detect similarity of programs
Late Policy

- No late submissions will be accepted.
- 11:59PM EST. Please submit it before the deadline.
Academic Integrity (AI) Policy for the Course

- minor violation:
  - 0 score for the involved homework/prog. assignment, and
  - 1-letter grade down
- 2 minor violations = 1 major violation
  - failure for the course
  - case will be reported to the department and university
  - further sanctions may include a dishonesty mark on transcript or expulsion from university
Academic Integrity (AI) Policy for the Course

- minor violation:
  - 0 score for the involved homework/prog. assignment, and
  - 1-letter grade down
- 2 minor violations = 1 major violation
  - failure for the course
  - case will be reported to the department and university
  - further sanctions may include a dishonesty mark on transcript or expulsion from university
Course Sign up

- Last Day to Drop/Add a Course: September 05
- Resign Date: November 10

Questions, please go to Piazza!
Outline

1  Syllabus

2  Introduction
   - What is an Algorithm?
   - Example: Insertion Sort
   - Analysis of Insertion Sort

3  Asymptotic Notations

4  Common Running times
Outline

1 Syllabus

2 Introduction
   - What is an Algorithm?
   - Example: Insertion Sort
   - Analysis of Insertion Sort

3 Asymptotic Notations

4 Common Running times
Donald Knuth: An algorithm is a finite, definite effective procedure, with some input and some output.
What is an Algorithm?

- Donald Knuth: An algorithm is a finite, definite effective procedure, with some input and some output.
- Computational problem: specifies the input/output relationship.
- An algorithm solves a computational problem if it produces the correct output for any given input.
Examples

Greatest Common Divisor

**Input:** two integers $a, b > 0$

**Output:** the greatest common divisor of $a$ and $b$

Example:

Input: 210, 270
Output: 30

Algorithm: Euclidean algorithm

$gcd(270, 210) = gcd(210, 270 \mod 210) = gcd(210, 60)$

$gcd(210, 270) = gcd(210, 60) = gcd(60, 30) = gcd(30, 0)$
Examples

Greatest Common Divisor

**Input:** two integers \(a, b > 0\)

**Output:** the greatest common divisor of \(a\) and \(b\)

Example:

- Input: 210, 270
- Output: 30
Examples

**Greatest Common Divisor**

**Input:** two integers $a, b > 0$

**Output:** the greatest common divisor of $a$ and $b$

**Example:**

- **Input:** 210, 270
- **Output:** 30

- **Algorithm:** Euclidean algorithm
Examples

Greatest Common Divisor

**Input:** two integers $a, b > 0$

**Output:** the greatest common divisor of $a$ and $b$

Example:

- Input: 210, 270
- Output: 30

Algorithm: Euclidean algorithm

$\text{gcd}(270, 210) = \text{gcd}(210, 270 \mod 210) = \text{gcd}(210, 60)$
Examples

Greatest Common Divisor

**Input:** two integers $a, b > 0$

**Output:** the greatest common divisor of $a$ and $b$

Example:

- Input: 210, 270
- Output: 30

Algorithm: Euclidean algorithm

$$\text{gcd}(270, 210) = \text{gcd}(210, 270 \mod 210) = \text{gcd}(210, 60)$$

$$(270, 210) \rightarrow (210, 60) \rightarrow (60, 30) \rightarrow (30, 0)$$
Examples

Sorting

**Input:** sequence of $n$ numbers $(a_1, a_2, \ldots, a_n)$

**Output:** a permutation $(a'_1, a'_2, \ldots, a'_n)$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$
Examples

**Sorting**

**Input:** sequence of \( n \) numbers \((a_1, a_2, \cdots, a_n)\)

**Output:** a permutation \((a'_1, a'_2, \cdots, a'_n)\) of the input sequence such that \(a'_1 \leq a'_2 \leq \cdots \leq a'_n\)

**Example:**
- **Input:** 53, 12, 35, 21, 59, 15
- **Output:** 12, 15, 21, 35, 53, 59

Algorithms: insertion sort, merge sort, quicksort, \ldots
Examples

Sorting

**Input**: sequence of \( n \) numbers \((a_1, a_2, \cdots, a_n)\)

**Output**: a permutation \((a'_1, a'_2, \cdots, a'_n)\) of the input sequence such that \(a'_1 \leq a'_2 \leq \cdots \leq a'_n\)

Example:

- **Input**: 53, 12, 35, 21, 59, 15
- **Output**: 12, 15, 21, 35, 53, 59

- **Algorithms**: insertion sort, merge sort, quicksort, …
Shortest Path

**Input:** directed graph $G = (V, E)$, $s, t \in V$

**Output:** a shortest path from $s$ to $t$ in $G$
Examples

Shortest Path

**Input:** directed graph $G = (V, E)$, $s, t \in V$

**Output:** a shortest path from $s$ to $t$ in $G$
Examples

**Shortest Path**

**Input:** directed graph $G = (V, E)$, $s, t \in V$

**Output:** a shortest path from $s$ to $t$ in $G$

![Diagram of a directed graph with labeled edges and nodes. The path from $s$ to $t$ is highlighted.]
Examples

**Shortest Path**

**Input:** directed graph $G = (V, E)$, $s, t \in V$

**Output:** a shortest path from $s$ to $t$ in $G$

![Graph Illustration]

- **Algorithm:** Dijkstra’s algorithm
Algorithm = Computer Program?

- Algorithm: “abstract”, can be specified using computer program, English, pseudo-codes or flow charts.
- Computer program: “concrete”, implementation of algorithm, using a particular programming language
Pseudo-Code:

Euclidean \((a, b)\)

1: while \(b > 0\) do
2: \((a, b) \leftarrow (b, a \mod b)\)
3: return \(a\)

Python program:

```python
def euclidean(a: int, b: int):
    c = 0
    while b > 0:
        c = b
        b = a % b
        a = c
    return a
```
Theoretical Analysis of Algorithms

- Main focus: correctness, running time (efficiency)
Theoretical Analysis of Algorithms

- Main focus: correctness, running time (efficiency)
- Sometimes: memory usage

Why is it important to study the running time (efficiency) of an algorithm?

1. feasible vs. infeasible
2. efficient algorithms: less engineering tricks needed, can use languages aiming for easy programming (e.g., python)
3. fundamental
4. it is fun!
Main focus: correctness, running time (efficiency)

Sometimes: memory usage

Not covered in the course: engineering side

- extensibility
- modularity
- object-oriented model
- user-friendliness (e.g., GUI)
- ...
Main focus: correctness, running time (efficiency)

Sometimes: memory usage

Not covered in the course: engineering side
  - extensibility
  - modularity
  - object-oriented model
  - user-friendliness (e.g., GUI)
  - ...

Why is it important to study the running time (efficiency) of an algorithm?
Main focus: correctness, running time (efficiency)

Sometimes: memory usage

Not covered in the course: engineering side
- extensibility
- modularity
- object-oriented model
- user-friendliness (e.g., GUI)
- ...

Why is it important to study the running time (efficiency) of an algorithm?
- feasible vs. infeasible
Theoretical Analysis of Algorithms

- Main focus: correctness, running time (efficiency)
- Sometimes: memory usage
- Not covered in the course: engineering side
  - extensibility
  - modularity
  - object-oriented model
  - user-friendliness (e.g., GUI)
  - ...

Why is it important to study the running time (efficiency) of an algorithm?

1. feasible vs. infeasible
2. efficient algorithms: less engineering tricks needed, can use languages aiming for easy programming (e.g., python)
Theoretical Analysis of Algorithms

- Main focus: correctness, running time (efficiency)
- Sometimes: memory usage
- Not covered in the course: engineering side
  - extensibility
  - modularity
  - object-oriented model
  - user-friendliness (e.g, GUI)
  - ... 

Why is it important to study the running time (efficiency) of an algorithm?

1. feasible vs. infeasible
2. efficient algorithms: less engineering tricks needed, can use languages aiming for easy programming (e.g, python)
3. fundamental
Theoretical Analysis of Algorithms

- Main focus: correctness, running time (efficiency)
- Sometimes: memory usage
- Not covered in the course: engineering side
  - extensibility
  - modularity
  - object-oriented model
  - user-friendliness (e.g, GUI)
  - ...

Why is it important to study the running time (efficiency) of an algorithm?

1. feasible vs. infeasible
2. efficient algorithms: less engineering tricks needed, can use languages aiming for easy programming (e.g, python)
3. fundamental
4. it is fun!