Spring 2026 MSML606 HW1 (deadline: 2/11, Wednesday)

Instructions:

- Do not directly copy content from textbooks, other individuals' work, AI-generated outputs, online sources, or any material that isn't your own. While collaboration and consulting various resources are encouraged, it's essential that you express your understanding in your own words, reflecting your personal learning and insights.
- Add justifications/explanations to all answers, as there will be points awarded for that.

Important!

Before submitting your work, review the external-source policy in the syllabus. You must either
(1) cite any external sources you used, including AI tools, or
(2) explicitly state that you did not use any.

Submissions without this AI usage statement will receive no points!!!

Rubric for Part I: (Maximum Score: 17 Marks)

Problem	Criteria	Marks
Problem 1: Big-O Analysis		4.5 Total
(a) Complexity Analysis	• Correct answer (1) • Justification (0.5)	1.5
(b) Complexity Analysis	• Correct answer (1) • Justification (0.5)	1.5
(c) Complexity Analysis	• Correct answer (1) • Justification (0.5)	1.5
Problem 2: Growth Rate Ordering		6 Total
Correct Order	• Fully correct order (5) • Missing same growth rate or out of order (-0.5 marks each)	5
Explanation	• Clear reasoning for ordering	1
Problem 3: O(1/n) Complex		3 Total
Answer	• Clearly states feasibility	1.5
Explanation	• Justification for why it is unrealistic	1.5
Problem 4: Complexity for ML algorithms		3.5 Total
Discussion	• Clearly states opinion providing reasonable arguments	3.5

HW1 – Part I

Problem 1

For each of the following code fragments, provide an analysis of the running time in O (Big-Oh) Notation.

(a)

```
for(i = 0; i < n; i++)
        for(j = 0; j < n; j+=3)
                k+=1;
```

(b)

```
cnt=0
for(i = 1; i < n; i*=2)
        if i%2!=0:
                cnt++;
```

(c)

```
if (n % 3 == 0) {
    printf("abc");
  } else {
    For (i=0; i<n; i++)
        printf("cba");
  }
```

Problem 2. Order the following functions by their growth rate from slowest to fastest. Indicate any functions that grow at the same rate:

$$n^n, n!, (n+1)!, \sqrt{n}, e^n, \log n\,, \log_2 n\,, n^{1.5}, \log(n!)\,, n, n\log n$$

Problem 3. Can an algorithm with O(1/n) time complexity exist? What would it imply about the algorithm's behavior as the input size increases?

Problem 4. Discuss whether traditional time and space complexity measures are still meaningful for evaluating modern machine learning algorithms, particularly neural networks.

HW1 –Part II

Rubric for Part II: (Maximum Score: 21 Marks)

Problem	Criteria	Marks
1: Algorithm Description		4 Total
Clarity & Completeness	• Clear with all steps explicitly stated; anyone could follow it	2
Precision	• Every action is unambiguous (what to pick, where to place, how to compare)	2
2: Pseudocode		4 Total
Correctness	• Algorithm is logically sound and would produce sorted output	2
Structure & Syntax	• Proper use of loops/conditionals/indentation • Follows standard conventions	1
Input/Output	• Clear labels for input (e.g., "Array A[1...n]") and output	1
3: Step-by-Step Trace		3 Total
Completeness	• Shows every major iteration from unsorted to sorted (0.5) • List of 10 elements used (0.5)	1
Accuracy	• Each step correctly reflects the algorithm's logic	1
Presentation	• Well-organized, easy to read (e.g., clearly numbered steps)	1
4: ADT Specification		3 Total
ADT Identification	• Correctly identifies the ADT used (List, Stack, Queue, etc.)	1.5
Operations Listed	• All relevant operations clearly listed (e.g., insert, remove, swap, access)	1
Justification	• Thoughtful explanation of why this ADT fits the algorithm's needs	0.5
5: Asymptotic Analysis		7 Total
Best Case	• Identifies what input causes best case (1) • Provides correct Big-O (1)	2
Worst Case	• Identifies what input causes worst case (1) • Provides correct Big-O (1)	2
Average Case	• Intuitive explanation expected (1.5) Rigorous proof <u>not</u> required	1.5
Space Complexity	• States whether in-place (0.5) • Justifies with Big-O space complexity (1)	1.5

Problem: Design Your Own Sorting Algorithm

In this problem, you will design an original (or your own variation of a) sorting algorithm. Your goal is not necessarily to be efficient (a brute-force $O(N^2)$ or even O(N!)) approach is acceptable), but to be mathematically precise in your description and analysis.

1. Algorithm Description (The "Steps")

Clearly describe your sorting method in plain English. Assume you are explaining the logic to an elementary school child using 10 cards, each with a single number written on it. What specific instructions would they need to arrange these cards in increasing order? Be explicit: how do they pick a card, where do they place it, and how do they compare it to the others?

2. Pseudocode

Provide a formal pseudocode representation of your algorithm.

- **Requirements:** Use standard control structures (Loops, If/Else). Ensure you clearly label your input (e.g., List A of size n) and your output (the sorted list).

3. Step-by-Step Trace

Demonstrate how your algorithm processes an unsorted list. Create an example list of 10 elements. Show the state of the list after each major step or iteration of your algorithm until the list is fully sorted.

4. ADT Specification (The "Tools")

Describe the **Abstract Data Type (ADT)** and the specific operations you are using to build your algorithm.

- **Instructions for students:** "Which ADT are you using (e.g., List, Stack, Queue, or Priority Queue)? List the specific operations/methods of that ADT your algorithm relies on (e.g., insert, remove, get_value_at, swap). Briefly explain why this ADT is appropriate for your logic."

5. Asymptotic Analysis

Perform a formal Big-O analysis of your algorithm.

- **Best Case:** Describe the input state that results in the fewest operations (e.g., already sorted) and provide the Big-O.
- **Worst Case:** Describe the input state that results in the most operations (e.g., reverse sorted) and provide the Big-O.
- **Average Case:** Provide an intuition for the average case.
- **Space Complexity:** Analyze whether your algorithm is "in-place" or requires additional memory/structures.