# Weekly Assignment 1
# Total: 100

CS 2500: Algorithms

**Due Date:** September 3, 2024 at 11.59 PM

## Instructions

- Submit your solutions by the deadline specified above.
- Ensure that your work is your own.
- Write your answers clearly and show all your work.
- If you have any questions, ask during recitations or office hours.

## Problems

1. In this exercise we deal with the problem of string matching.
   (a) Explain how to use a brute-force algorithm to find the first occurrence of a given string of $m$ characters, called the target, in a string of $n$ characters, where $m \leq n$, called the text. **[5 points]**
   (b) Express your algorithm in pseudocode. **[5 points]**
   (c) Give a big-O estimate for the worst-case time complexity of the brute-force algorithm you described. **[5 points]**
2. Consider the following algorithm, which takes as input a sequence of $n$ integers $a_1, a_2, \ldots, a_n$ and produces as output a matrix $M = \{m_{ij}\}$ where $m_{ij}$ is the minimum term in the sequence of integers $a_i, a_{i+1}, \ldots, a_j$ for $j \geq i$ and $m_{ij} = 0$ otherwise.

```
m_ij ← a_i if j ≥ i, else m_ij ← 0;
for i ← 1 to n do
    for j ← i + 1 to n do
        for k ← i + 1 to j do
            m_ij ← min(m_ij, a_k);
return M;
```

   (a) Show that this algorithm uses $O(n^3)$ comparisons to compute the matrix $M$. **[10 points]**

(b) Show that this algorithm uses $\Omega(n^3)$ comparisons to compute the matrix $M$. Using this fact and part (a), conclude that the algorithms uses $\Theta(n^3)$ comparisons. [Hint: Only consider the cases where $i \leq \frac{n}{4}$ and $j \geq \frac{3n}{4}$ in the two outer loops in the algorithm.] **[10 points]**

3. Big-O notation compares the growth rates of functions using the inequality $f(n)$ is $O(g(n))$ iff $|f(n)| \leq C \cdot |g(n)|$ for sufficiently large $n$. In the context of analyzing the growth of complexity functions, why do we typically assume that $f(n)$ and $g(n)$ are always positive? **[5 points]**

4. Show that $n^2\log(n) \notin \Theta(n^2)$. **[10 points]**

# Finding Unique Triplets Summing to Zero

This is a popular problem that companies like Amazon, Meta, Microsoft, Oracle, etc. often ask in their interviews to test your problem-solving skills and ability to work with arrays.

- You must submit both the source code and a brief report that explains your approach, the algorithms used, and any challenges encountered.
- Ensure your code is well-documented, with comments explaining the purpose of key sections and functions.
- Your code will be evaluated on both correctness and efficiency, as well as clarity and organization.

## Part 1: Coding Exercise [20 points]

### Task

Write a function `tripletSum(nums)` that takes an array of integers `nums` and returns all unique triplets $[\text{nums}[i], \text{nums}[j], \text{nums}[k]]$ such that $i \neq j$, $i \neq k$, $j \neq k$, and $\text{nums}[i] + \text{nums}[j] + \text{nums}[k] = 0$.

### Constraints

The solution set must not contain duplicate triplets.

### Examples

- **Example 1:**
  - **Input:** `nums = [-1,0,1,2,-1,-4]`
  - **Output:** `[[-1,-1,2],[-1,0,1]]`
  - **Explanation:**
    $\text{nums}[0] + \text{nums}[1] + \text{nums}[2] = (-1) + 0 + 1 = 0$.
    $\text{nums}[1] + \text{nums}[2] + \text{nums}[4] = 0 + 1 + (-1) = 0$.
    $\text{nums}[0] + \text{nums}[3] + \text{nums}[4] = (-1) + 2 + (-1) = 0$.
    The distinct triplets are `[-1,0,1]` and `[-1,-1,2]`.
    Notice that the order of the output and the order of the triplets does not matter.
- **Example 2:**
  - **Input:** `nums = [0,1,1]`
  - **Output:** `[]`
  - **Explanation:** The only possible triplet does not sum up to 0.
- **Example 3:**
  - **Input:** `nums = [0,0,0]`
  - **Output:** `[[0,0,0]]`
  - **Explanation:** The only possible triplet sums up to 0.

## Part 2: Analysis [30 points]

1. Analyze the time complexity of your implementation. Is it possible to improve the time complexity? If yes, describe how you would do so. [**5 points**]
2. What is the space complexity of your solution? Discuss whether there are trade-offs between time and space complexity in your approach. [**5 points**]
3. How does your solution ensure that the returned triplets are unique? What specific techniques or steps do you use to avoid duplicates? [**5 points**]
4. Describe how your algorithm handles the following edge cases:
   (a) The array contains fewer than three elements. [**2 points**]
   (b) All elements in the array are the same (e.g., `[0, 0, 0]`). [**2 points**]
   (c) The array contains only positive or only negative numbers. [**4 points**]
5. How does your algorithm perform as the size of `nums` increases towards the upper limit (e.g., nums.length = 3000)? What are the potential bottlenecks, and how might you address them? [**4 points**]
6. If you were asked to find pairs of numbers that sum to zero instead of triplets, how would your approach change? How does the complexity of the problem scale as you move from pairs to triplets? [**3 points**]