Please type (or neatly handwrite) your solutions on standard 8.5×11 paper, with your name at the top of each solution. Ensure that you submit you solutions in one file PDF file on Gradescope. **each problem sets solution should be on in its own individual page, Gradescope will help ensure you submit each solution under its correct problem number**

While a solution must be absolutely perfect to receive full marks, I will be generous in awarding partial marks for incomplete solutions that demonstrate progress.

This work must be submitted on Gradescope on or before the due date. The late due date can be used with no penalty. Note that the system wont allow for work to be submitted after the late due date. Any work, not submitted will be awarded zero points. The instructor does not accept emailed work.

So that there is no ambiguity, there are two non-negotiable rules. A violation of either rule constitutes plagiarism and will result in you receiving an F for this course.

(a) If you meet with a classmate to discuss any of the Individual Problems, your submission must be an individual activity, done in your own words, away from others. The process of finding a solution might take 3 - 5 iterations or even more BUT you learn from all these attempts and your confidence grows with each iteration.

(b) These problem sets might seem hard on a first look. They are designed to be so. We learn by attempting problems, struggling through them and coming on top. I encourage you to make this learning exercise worth your while. What do I mean? Open the problem sets as early as you get them, then do not look at hints or answers any where (including on the internet and consulting other students for direct answers), give it the best shot you can. If you get stuck come to Professor or TA's office hour and we shall be glad to listen to your rationale and work with you till you are able to tackle the problem sets.

Problem #1 (20 points)

This problem will enabled you to explore linked lists and arrays in the context of insertion efficiency, search and deletion operation and memory usages. Notice for this question we are considering unsorted array A of n integers and an unsorted singly linked list L containing n integers

Please note that pseudocode/source code is required for all sub-parts

(a) You are given an unsorted array A of n integers and an unsorted singly linked list L containing n integers. Describe an algorithm to insert a new element into A and L . Analyze the time complexity of insertion for both data structures in the best, average, and worst cases. Explain the circumstances under which one data structure might be preferred over the other for insertion operations.

(b) Describe an algorithm to search for an element in the array A and the linked list L. Compare the time complexity of the search operation in both data structures. Discuss how the time complexity changes if the array is sorted. How does this affect your choice of data structure if frequent search operations are required?

(c) Discuss the process of deleting an element from the array A and the linked list L . Analyze and compare the time complexity of deletion in both data structures. Consider how the position of the element to be deleted (e.g., beginning, middle, end) impacts the complexity of the operation for each data structure.

(d) Consider a scenario where memory usage is a critical factor. Discuss the memory allocation of arrays and linked lists when:

- data structure is initialized
- data structure is at half capacity
- data structure is dynamically resized (for arrays) or nodes are added/removed (for linked lists)

Considering this scenarios, what is your considerations/recommendations for choosing arrays and linked lists in memory constraint environment?

Problem 2:

A queue is a linear data structure that follows the First-In-First-Out (FIFO) principle. Implementing a queue using arrays and linked lists tests one's understanding of basic data structures and their operations. This problem set will explore designing, implementing, and analyzing queues using both arrays and linked lists in a language of your choice.

Please note that pseudocode/source code is required for all sub-parts

(a) Design and implement a queue using a static array. Your implementation should support the following operations: *enqueue, dequeue, and peek.*

Discuss the limitations regarding the fixed size of the queue. Analyze the time complexity of your operations.

(b) In part *a* above, notice that your queue does not allow for inserting new elements if the queue is full. In this section, enhance your array-based queue implementation to allow dynamic resizing without losing any existing data in the queue.

Describe the strategy used to resize the queue and how it impacts the enqueue and dequeue operations. Analyze the amortized time complexity of the operations with dynamic resizing.

(c) In question 1 of this problem set, we saw that linked list although implemented different from arrays, they accomplish the same thing. In this regard we can design queues using linked list. In this section, design and implement a queue using a singly linked list. The implementation should support the same operations: *enqueue, dequeue, and peek.*

Compare and contrast the linked list queue implementation with the dynamic array-based queue in terms of memory allocation and performance.

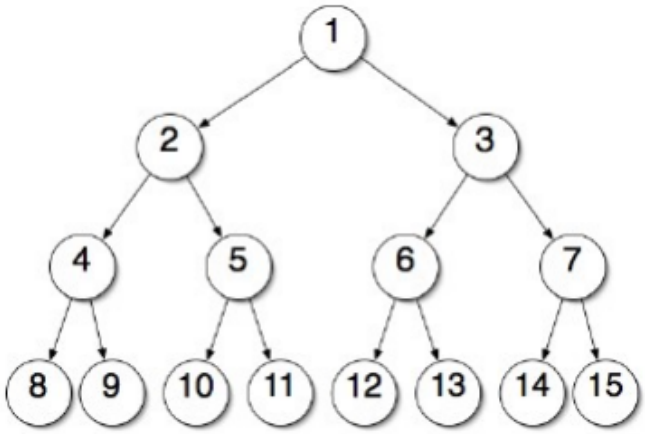
Analyze the time complexity of the operations with the linked list implementation.

(d) Queues are powerful data structures and are used in many applications. As such queue operation optimization strategies are important during implementations. In this section, Discuss optimization strategies for the enqueue and dequeue operations for both the dynamic array and linked list implementations. Consider scenarios with a high frequency of operations and the need to minimize the time spent on each operation.

- What optimization techniques can be applied to the array-based queue to improve performance?
- For the linked list queue, how would you optimize node allocation and deallocation to manage memory more effectively?
- Present any trade-offs involved in your optimizations and how they affect the overall performance of the queue operations.

Problem #3

Consider this binary tree, where each vertex is labelled with a positive integer. The root vertex is 1.



For all positive integers $k \geq 1$, vertex k has two children: $2k$ (Left) and $2k + 1$ (Right).

(a) In your own words, describe how Breadth-First Search (BFS) and Depth-First Search (DFS) work. Does one search algorithm *always* reach the destination faster than the other? Explain.

(b) Suppose we want to determine a path from vertex 1 (start vertex) to vertex 10 (end vertex).

Using BFS, determine the order in which the vertices will be visited. Using DFS, determine the order in which the vertices will be visited. Briefly explain your answers.

(c) Suppose that we extend this binary tree to infinitely many levels, so that each vertex k has two children: $2k$ (Left) and $2k + 1$ (Right).

The path from vertex 1 to vertex 10 can be described by a sequence of Left and Right moves, namely **Left, Right, Left**.

Consider the path from vertex 1 to vertex 2021. Determine the sequence of Left and Right moves for this path. Clearly justify your answer.