Assignment: Computer Science

Computer Science: Problem Solving Assignment ? Linked Lists

Due Date: April 5th, 2025

Word Limit: 1000 words (excluding code)

Topic: Linked Lists

This assignment focuses on your understanding and practical application of linked list data structures. You will be required to design, implement, and analyze various operations on singly linked lists. Your solution should be well-documented and demonstrate a strong grasp of algorithmic efficiency and data structure principles. Emphasis will be placed on code clarity, correctness, and efficiency.

Part 1: Implementation (70%)

Implement a singly linked list class in a language of your choice (e.g., Python, Java, C++, C#). Your implementation should include the following methods (at minimum):

* **`insert(data, position)`:** Inserts a new node with the given `data` at the specified `position` in the list (0 being the head). Handle edge cases such as inserting at the head, tail, or an invalid position.
* **`delete(position)`:** Deletes the node at the specified `position` in the list. Handle edge cases such as deleting the head, tail, or an invalid position. Return the deleted data.
* **`search(data)`:** Searches for a node containing the given `data` and returns `True` if found, `False` otherwise.

* **`reverse()`:** Reverses the order of nodes in the linked list.

* **`print_list()`:** Prints the data of all nodes in the linked list.

Part 2: Problem Solving and Analysis (30%)

Answer the following questions, providing clear explanations and justifications for your answers. Include relevant code snippets where necessary to support your reasoning. For questions involving algorithmic complexity, use Big O notation to express your analysis.

AI-Generated Questions:

1. **Merge Two Sorted Linked Lists:** Write a function that takes two sorted linked lists as input and merges them into a single sorted linked list. Analyze the time and space complexity of your solution. Consider the edge cases of empty input lists.

2. **Detect a Cycle:** Develop an algorithm to detect whether a given linked list contains a cycle (a loop where a node points back to a previous node). Analyze the time and space complexity of your algorithm. Discuss the efficiency of your approach compared to alternative methods.

3. **Nth Node from the End:** Write a function to find the Nth node from the end of a linked list. Handle the edge case where N is larger than the length of the list. Analyze the time and space complexity of your solution. Explore different approaches and discuss their trade-offs.

4. **Remove Duplicates:** Implement a function that removes duplicate nodes from a sorted linked list, ensuring that only unique values remain. Analyze the time and space complexity.

5. **Palindrome Check:** Design an algorithm to determine whether a linked list is a palindrome

(reads the same forwards and backwards). Analyze the time and space complexity of your algorithm. Consider memory usage and efficiency for very large lists.

Submission:

Submit a single document containing:

* Your well-documented source code implementing the linked list class and the solutions to the problem-solving questions.
* A written explanation for each problem-solving question, including detailed analysis of your chosen algorithm, its time and space complexity, and justification for your design choices.

This assignment aims to assess your understanding of linked lists, your ability to implement core functionalities, and your problem-solving skills in the context of data structures. Good luck!