Report: Examor

Overview

This report documents the project implementation of Examor (Exam Management System using a Singly Linked List) . The project includes exercises covering student management, list operations, and optimizations like splitting and merging lists. The complexity of each function is analyzed.

***Note:** function names have been modified to “snake_case” for better readibility and convinience.

Function Complexity Analysis

1. create_student() - O(1): Allocates memory and initializes a single student record.
2. display_list() - O(n): Traverses the linked list to print all student details.
3. add_student() - O(n): Appends a new student to the end of the list.
4. student_count() - O(n): Traverses the list to count the nodes.
5. find_student() - O(n): Searches for a student by their ID.
6. delete_last_student() - O(n): Traverses the list to find and remove the last node.
7. sort_list() - O(nlog(n)): Uses merge sort to sort students by grade.
8. average_exam() - O(n): Traverses the list to calculate the sum of grades and the count of students.
9. free_list() - O(n): Traverses the list and frees each node.
10. split_list() - O(n): Traverses the list to create two new sub-lists based on grades.
11. merge_lists() - O(n): Merges two sorted lists into a single one.

Additional Functions

12. print_intro(): Prints a colorful intro when the program starts.

**Main problem about this application is that, the exam list only exist on the random access memory and is lost when the application is closed. We implemented the following two functions to solve this issue by saving to a file or loading from a file.

13. save_list(): Saves the current list to a file.

14. load_list(): Loads a linked list from a file.

**We also implemented the following two functions to make the application more dynamic and actually usable.

15. update_student(): Updates details about a students.

16. delete_student(): Deletes a student in the list.

Additional Interface

As an additional interface we implemented a command-line interface (CLI) for speed and for those who likes CLI more :)

Complexity

The project implements 11 core functions, each with varying complexities based on their operations. Functions like `create_student()` execute in constant time **O(1)** as they allocate and initialize memory. Traversal-based functions like `display_list()`, `student_count()`, `find_student()`, `delete_last_student()`, `average_exam()`, `free_list()`, `split_list()`, and `merge_lists()` operate in linear time **O(n)**. The `add_student()` function, which appends nodes, also has a linear complexity of **O(n)**. Sorting is handled by `sort_list()` using merge sort, which has a time complexity of **O(nlog(n))**.