Project Title: Digital Forensic File Reconstruction System

1. Problem Description and Project Aims

What we're looking to solve here revolves around a classic computer science illusion: **File deletion**. When we 'delete' a file from a computer or a storage device it isn't 'wiped out' as per say. What happens is the file removes the ***pointer*** (or the reference) to the data, not the data itself. The OS maintains a **'file allocation table'**, this basically maps *file names* to *data blocks* on disk. So, when we delete a file, we delete only the entry in that table. The OS then says it's **'free' space**. Until other data is overwritten in the same fragments, we can still recover the prior data using forensic tools. The forensic software goes to scan the *raw disk*. It finds clues like matching file headers, footers, and metadata clues like timestamps. Then it reconstructs the deleted file.

This project will **simulate** this process:

1. We will take a folder with, say, 10 files inside it.
2. We will 'delete' some files and track their metadata.
3. We will then reconstruct it using the stored metadata. We won't actually do a full recovery, but rather simulate it in our program to show how it works in disk memory.

If we actually want data *truly erased*, then we use file shredder software, which overwrites the data with random bits a couple of times so data recovery is impossible.

2. Data Structures

We plan on applying a majority of the data structures and algorithms that we have studied or will study in the class. To manage the file metadata and recovery history, we will need these core structures, primarily:

Data Structure	Purpose in the Project	Justification
Linked List	**Metadata Storage**. This will initially store all the info like name, size, data modified before insertion into the tree.	This will be our file table holding data like: Name, Path, Size, and Status (Deleted/Active) . We use a linked list because it's the fastest way to collect entries with a time complexity

		of O(1). Once collected, we can shift them into a tree.
AVL Tree / BST	**The now-active file table**. Now, this will store the metadata for all files.	We use this to make searching, deletion and recovery faster. Sorting will be by some attribute of the metadata, like filename/timestamp. Our structure choice will depend on the comparative time complexity of each in action.
Stack	**Recovery History**. This will track metadata for files that have been logically 'deleted'.	We can move the 'deleted' files' metadata to a stack to keep a record of deletion history, in order. Why a Stack not a heap or queue? Typically in deletion, as is with the 'recently deleted' feature on several platforms, Stacks are used because they are LIFO models. More often than not, we want to see or acquire the most recently deleted record first, a stack helps perfectly there.

3. Main Algorithms

The system's functionality relies on standard algorithms adapted for our file simulation:

a. **Search and Traverse Tree:** We will use algorithms to search the linked list or BST to locate a specific file entry

   e.g. **File* deleted = SearchinTree("notes.txt")**

   This is important for both deletion and recovery.

- **Insertion and Deletion:** Standard tree algorithms will handle inserting new file entries. For deletion, we simply search for the file and update its **Status** attribute to "Deleted".
- **Recovery Simulation:** The program will search the linked list/BST from earlier for entries marked as 'deleted'. Once the metadata is acquired , it will check for or create a **Recovered** folder. Finally, it creates a fresh empty file **e.g. Recovered_notes.txt** and writes the recovery text e.g “This is the recovery file of the prior file xyz.txt”, and the deleted file’s information inside.
- **Status Update:** After the recovery process, the program will change the deleted file status to “recovered” in the linked list or BST.

4. Data Flow and Processing

The overall sequence of the project will look like this:

1. **Input:** Make a folder and have different files in it e.g text, pdf, images. Feed this folder path to the program.
2. **Collection and Storage:** The program will read all files inside the folder. All their info like name, size, data modified. It will store this metadata in a data structure.
3. **File Table Creation:** Insert this metadata into a Binary Search Tree/AVL Tree.
4. **Simulated Deletion:** Then 'delete' a file from the folder. The program will still have its metadata in the file table we created earlier. This is how an OS actually works, so we have mimicked that. The deleted file metadata is moved to a **Stack** for recovery history.
5. **Recovery:** We need to recover data. For this, we will only use the metadata we stored earlier in the program.
6. **Output:** We will create a “fake” recovered version of the file. We’ll create a new blank file in a “Recovered” folder. The program will write information inside the file, consisting of the line “This file was recovered using stored metadata” and the original file’s info.
7. **Front-End Display (Potential):** The program will display the list of functional/current files (from the BST) and deleted files (from the Stack).