Deliverable-1: Project Proposal & Conceptual Design

Course: Data Structures and Algorithms (CS221)

Project Title: SentinelAV – A DSA-Based Antivirus

## 1. Problem Description

Computer viruses can infect systems by hiding inside files.  
Professional antivirus software detect these by matching known virus “**signatures**.”

Our goal is to build a basic antivirus tool that focuses on how **data structures and algorithms** can make virus scanning fast and efficient.  
It will scan files, match virus signatures, and report infections — emphasizing core DSA applications in a real cybersecurity problem.

## 2. Key Data Structures

| **Data Structure** | **Purpose** |
| --- | --- |
| **Trie (Prefix Tree)** | To store virus signatures efficiently and allow quick pattern lookup. |
| **Aho–Corasick Automaton** | Built from the trie; enables fast multiple-pattern matching during scanning. |
| **Hash Table** | For storing and retrieving virus signatures using hashing techniques. |
| **Bloom Filter** | For quick “probably safe” checks before full scanning. |
| **Linked List / Queue** | Used internally while constructing and traversing the automaton. |
| **Priority Queue (Heap)** | To rank files based on the number of detected signatures. |
| **Dynamic Programming Array (optional)** | For approximate or fuzzy matching of modified virus patterns. |

## 3. Main Algorithms

| **Algorithm** | **Function** |
| --- | --- |
| **Aho–Corasick Algorithm** | Detects multiple virus patterns efficiently in one scan pass. |
| **Rabin–Karp Rolling Hash** | Helps detect patterns and their variations quickly. |
| **Bloom Filter Operations** | Rejects clean files early to save time. |
| **Heap Operations** | Maintains a ranked list of suspicious files. |
| **Levenshtein Distance (optional)** | Enables fuzzy matching for slightly altered viruses. |

## 4. Data Flow

1. **Input:** User selects files or folders to scan.
2. **Pre-Check:** Bloom filter skips files unlikely to contain viruses.
3. **Scanning:** Aho–Corasick automaton scans remaining files for known signatures.
4. **Matching:** Detected patterns are logged in a hash table.
5. **Ranking:** Priority queue ranks files by number of matches.
6. **Output:** Displays file name, infection status, and detected signatures.

## 5. Integration with DSA Course Concepts

| **Course Topic** | **Application in Project** |
| --- | --- |
| **Pointers & Arrays** | Used for file buffer handling and automaton structures. |
| **Lists & Queues** | Used while constructing and traversing the automaton. |
| **Sorting** | For result ordering or ranking. |
| **Trees (Tries)** | Core structure for virus signature storage. |
| **Graphs** | Automaton modeled as graph transitions. |
| **Hashing** | Used in signature indexing and Bloom filters. |
| **Priority Queues / Heaps** | To manage the most suspicious files. |
| **Complexity Analysis** | Time and space analysis of implemented algorithms. |

## 6. Expected Outcome

By the end of the project, we will have:

* A working file-scanner antivirus prototype in C++.
* Efficient detection using tries, hashing, and priority queues.
* A performance comparison between naive and optimized scanning methods.
* A report showing how DSA improves real-world system performance.

## 7. Future Enhancements (If Time Allows)

To extend functionality, **heuristic-based detection** may be added later for smarter virus detection.

**Heuristic Detection Overview**

* **Static Analysis:**  
  Analyzes file content without execution using arrays, hashing, and pattern scoring.
* **Dynamic Analysis:**  
  Observes file behavior in a test environment using graphs, queues, and scoring algorithms.

These will only be implemented **if time permits**, after completing the core DSA-based modules.

## 8. Team Members

1. **Usman Azhar**
2. **Reyan Kashif**
3. **Malik Abdullah**

**Submitted to: Mr. Zubair Ahmad**

**Submission Date: 07 / 10 / 2025**