# Deliverable-1: Project Proposal & Conceptual Design

**Course:** Data Structures and Algorithms (CS221)

**Project Title:** SentinelAV – A DSA-Based Antivirus

**1. Problem Description**

Computer viruses often hide inside legitimate files, and professional antivirus tools rely on matching known virus “signatures” to detect them.

For this project, we aim to build a fundamental antivirus scanner that demonstrates how data structures and algorithms can be applied to make this process both fast and efficient. Our tool will scan files, identify signatures, and report potential infections, providing a practical application of core DSA concepts to a real-world cybersecurity challenge.

**2. Integration with DSA Course Concepts**

This table shows how the project directly applies concepts from your Data Structures and Algorithms course.

|  |  |
| --- | --- |
| **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. |

**3. Core Data Structures and Algorithms**

The following data structures and algorithms form the technical core of the SentinelAV scanner.

**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. |

**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:** The user will select a file or folder to scan.
2. **Pre-Check:** A Bloom filter will quickly check and pass over files that are very unlikely to contain viruses.
3. **Scanning:** For the remaining files, our Aho–Corasick automaton will perform the main scan to find known signatures.
4. **Matching:** Any matches found are logged using a hash table.
5. **Ranking:** A priority queue will be used to rank files based on how many matches were found.
6. **Output:** The tool will display a list of scanned files, their infection status, and the specific signatures detected.

**5. Expected Outcome**

By the end of this project, we aim to deliver:

* A functional antivirus prototype developed in C++.
* An efficient scanning engine built on tries, hashing, and priority queues.
* A performance analysis comparing our optimized approach with a more basic, naive scanning method.
* A final report that documents our findings on how specific data structures impact performance in this application.

**6. Future Enhancements (If Time Allows)**

If we have time after completing the core requirements, we may explore adding heuristic-based detection for smarter virus identification.

**Heuristic Detection Overview**

* **Static Analysis:** Analyzing file content without running it, using arrays, hashing, and pattern scoring.
* **Dynamic Analysis:** Observing file behavior in a sandboxed environment using graphs, queues, and scoring algorithms.

**7. Team Members**

* Usman Azhar
* Reyan Kashif
* Malik Abdullah

**Submitted to:** Mr. Zubair Ahmad

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