**Project report on prison management system**

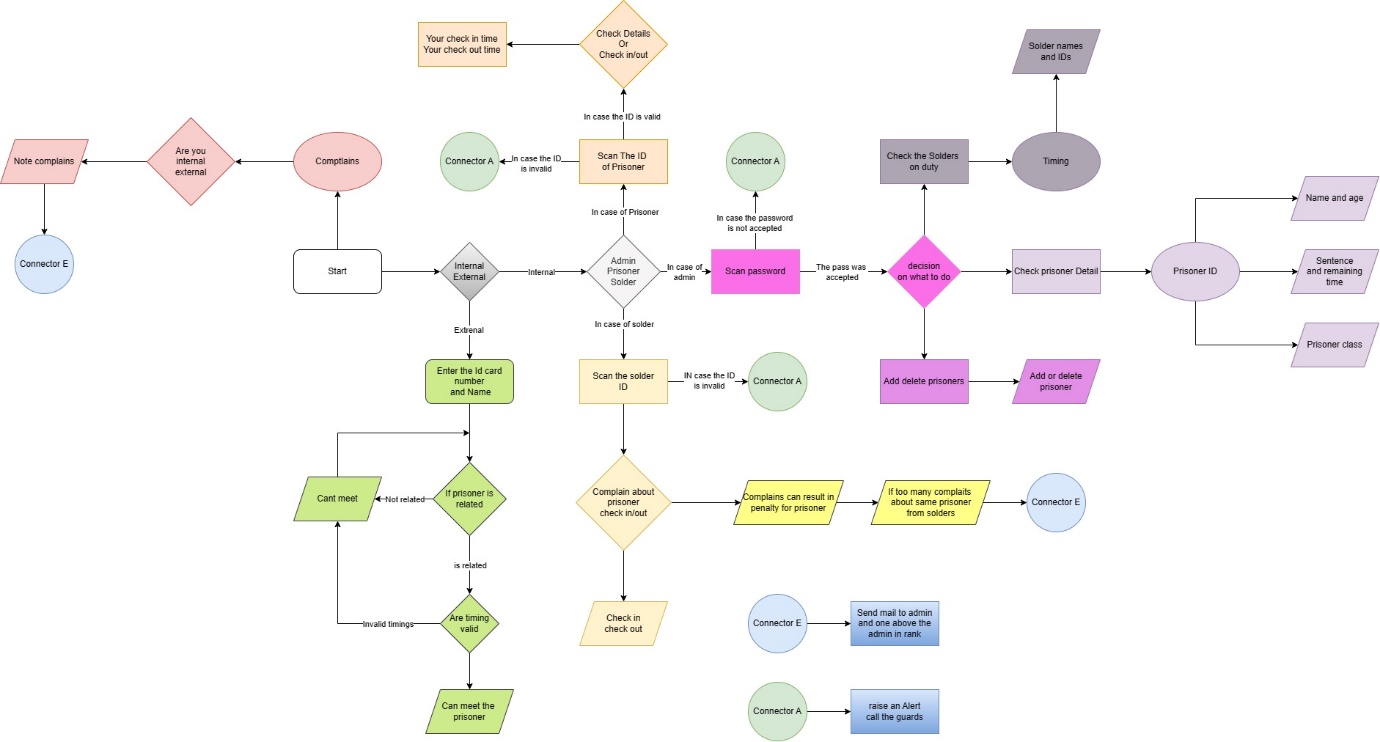
**Introduction:**

A project called "Police Management System" intends to make managing prison records easier. This system will offer a user-friendly interface for Admin, prisoner, and normal users while effectively storing and retrieving prison records using a binary tree data structure. Admin will have more authority over the system than a normal user. The system will also take complaints from all types of users and store them in a database.

**Objectives:**

1. Creating a user-friendly interface for users, prisoners, and Admin.
2. Efficiently managing data through binary trees.
3. Efficiently updating, adding, and removing data.
4. Efficiently allowing users to see their relative's prison records and request a meetup.
5. Using QR codes for prisoner's identity and easy login for Admins.
6. Let the prisoner only see his record.
7. Using a graphical user interface for Admin and the user to interact with the system.

**Basic data flow:**



**Data structures:**

The data structure we have used in our code is the binary search tree. We store the data of the prisoners in a file and then we store the names and IDs of the prisoners in BSTs. This helps us reduce the search time for any prisoner.

**Time complexity:**

To store data, we could have used linked lists but then the search would have been O(n). In the case of the BST its log(n). In the case of an unbalanced search tree, the complexity is O(n) but we have a function that always converts the BST into a balanced tree so that the complexity is always O(log n). The complexity of the balancing function is O(n).

**Work done:**

We have created the binary trees, prepared the files, and coded functions to read and write from the files. We have also created the function to add the prisoners to the file. We have also overloaded different operators for the classes.

**Quick Sort:**

We have written a function to sort the arrays when we add or remove a prisoner. For this, we have used quick sort. The reason for using quick sort is that it is more efficient than the merge sort in case of arrays that are already partially sorted. Merge sort does not adapt and will perform its functionality even if the array is partially sorted. So, it has the same best, worst, and average complexities. But it’s different for quick sort. It is more adaptable and has different complexities if the arrays are partially sorted.

**1. Time Complexity:**

**Merge Sort:**

- Worst Case: O(n log n)

- Average Case: O(n log n)

- Best Case: O(n log n)

**Quicksort:**

- Worst Case: O(n^2) - can occur if the pivot selection consistently results in unbalanced partitions.

- Average Case: O(n log n)

- Best Case: O(n log n) - when the pivot selection consistently produces balanced partitions.

In general, quick sort has better average-case time complexity than merge sort, but its worst-case time complexity can be worse.

**2. Space Complexity:**

**Merge Sort:**

- Requires additional space proportional to the size of the input array. This makes it less memory-efficient, especially for large datasets.

- The space complexity is O(n).

**Quicksort:**

- Quick sort version has a space complexity of O(log n) due to the recursive nature of the algorithm.

- Quicksort is often more memory-efficient than merge sort because it can be implemented in an in-place manner.

In summary, while both merge sort and quicksort have average-case time complexities of O(n log n), merge sort is more consistent in its performance and is a stable sorting algorithm, but it requires additional space. Quicksort, on the other hand, is usually faster in practice and can be more memory-efficient when implemented in-place, but its worst-case time complexity is higher. The choice between them depends on the specific requirements and characteristics of the data being sorted.

We have used the quick sort algorithm because we want to design a system that will hold the data of a lot of prisoners. So, memory usage will be a problem. The quick sort will never achieve the worst complexity because all our files are partially sorted, and the only sorting needed is when we add or remove the data of a prisoner.