Explanation of the Algorithms Used in the Bank Organization System

In this project, I implemented two core algorithms: Merge Sort for sorting applicants alphabetically, and Binary Search for efficiently locating specific employees by name. These algorithms were chosen deliberately based on their performance advantages and suitability for handling a growing list of employee records. The system also accepts new user input, allowing employees to be added manually or generated automatically, which means the algorithm must handle a dynamic list of data that may continue to increase over time.

The first major algorithm used is Merge Sort, a divide-and-conquer sorting method. I selected Merge Sort instead of simpler methods like Bubble Sort or Selection Sort because Merge Sort is significantly faster, especially when dealing with large data. Merge Sort works by repeatedly splitting the list into halves until each portion contains only one element. Single-element lists are already sorted, so the algorithm begins merging them back together in sorted order. This merging process continues until the entire list is rebuilt in alphabetical order. The key advantage of this algorithm is its O(n log n) time complexity, which makes it ideal for applications that handle many entries. Bubble Sort and Selection Sort, by comparison, run in $O(n^2)$ time, meaning they slow down drastically as the number of applicants increases. Because a bank organization can collect hundreds or even thousands of names over time, Merge Sort was the most appropriate and scalable choice.

Another benefit of Merge Sort is that it is stable and predictable. It always performs in O(n log n) time regardless of how sorted or unsorted the list is. Algorithms like Bubble Sort can perform slightly better on partially sorted lists, but they become extremely slow when the list is large or completely unsorted. Since applicant data can be random and unpredictable, it is safer to use a sorting algorithm that performs consistently in all situations. Merge Sort also works well with objects, such as the Employee class in this project, because it compares data fields, in this case, the employee name without modifying the overall structure of the object.

After sorting, the program uses Binary Search to quickly locate a specific employee. Binary Search requires the list to be sorted first, which is another reason Merge Sort was a suitable choice. Binary Search is much faster than Linear Search because it divides the list in half during each comparison. Instead of checking

every element one by one, the algorithm examines the middle element, decides whether the target name comes before or after it alphabetically, and then repeats the process only in the relevant half. This gives Binary Search a time complexity of O(log n), making it extremely efficient. With this algorithm, even a large list can be searched very quickly.

Finally, the system supports new user input. Users can manually add employees, and these new entries are inserted into the list before future sorting or searching. This interaction ensures that the algorithms continue to function correctly even as the list grows.

Overall, I chose Merge Sort and Binary Search because they are efficient, scalable, and ideal for systems that manage large datasets. Merge Sort guarantees fast and consistent sorting performance, while Binary Search provides rapid lookup times. Together, they create a reliable algorithmic foundation for the Bank Organization application.

GITHUB LINK- https://github.com/Reginald-Ghub/BankOrganization