Project Documentation

Contents

1. Problem Specification: .....

2. Software Specification: .....

3. Design Diagram: .....

4. Operational Details: .....

5. Testing: .....

1. PROBLEM SPECIFICATION-

Objective:

The objective of this project is to create a Java application that implements various sorting and searching algorithms, along with additional functionalities for managing and analyzing data. The project aims to compare the complexities of different algorithms, provide a user-friendly interface for interacting with data, and demonstrate proficiency in object-oriented programming concepts.

Scope:

1. Creation of a list with data from a file or user input.
2. Sorting operations using one simple sorting algorithm (selection sort, insertion sort, bubble sort) and one O(Nlog2N) sort (Quick sort, Merge Sort, Heap sort).
3. Searching operations including linear search, binary search tree, and hash function search.
4. Additional user-defined functions for managing data quantity, adding, deleting, updating, and restoring data.
5. Analysis reporting feature to provide insights into the data and operations performed.

Requirements:

1. Develop a main application program (`CS401prj.java`) and at least one user-defined class without a main method.
2. Implement sorting algorithms without using Java's sorting library.
3. Implement searching algorithms for linear search, binary search tree, and hash function search.
4. Provide a menu-driven interface for selecting algorithms and other functions.
5. Support different data types (integer, float, string).
6. Print the sorted list and total count of comparisons for sorting algorithms.
7. Display data in a well-organized format for each menu.
8. Use a data set size of at least 100.
9. Use any implementation structure (array or linked list) except Java's ArrayList.
10. Use inheritance and/or interfaces to define abstract methods.
11. Ensure the project includes at least one user-defined class for sorting, searching, and other functionalities (OOP requirement).
12. Implement additional features for extra credit opportunities.

2. SOFTWARE SPECIFICATION-

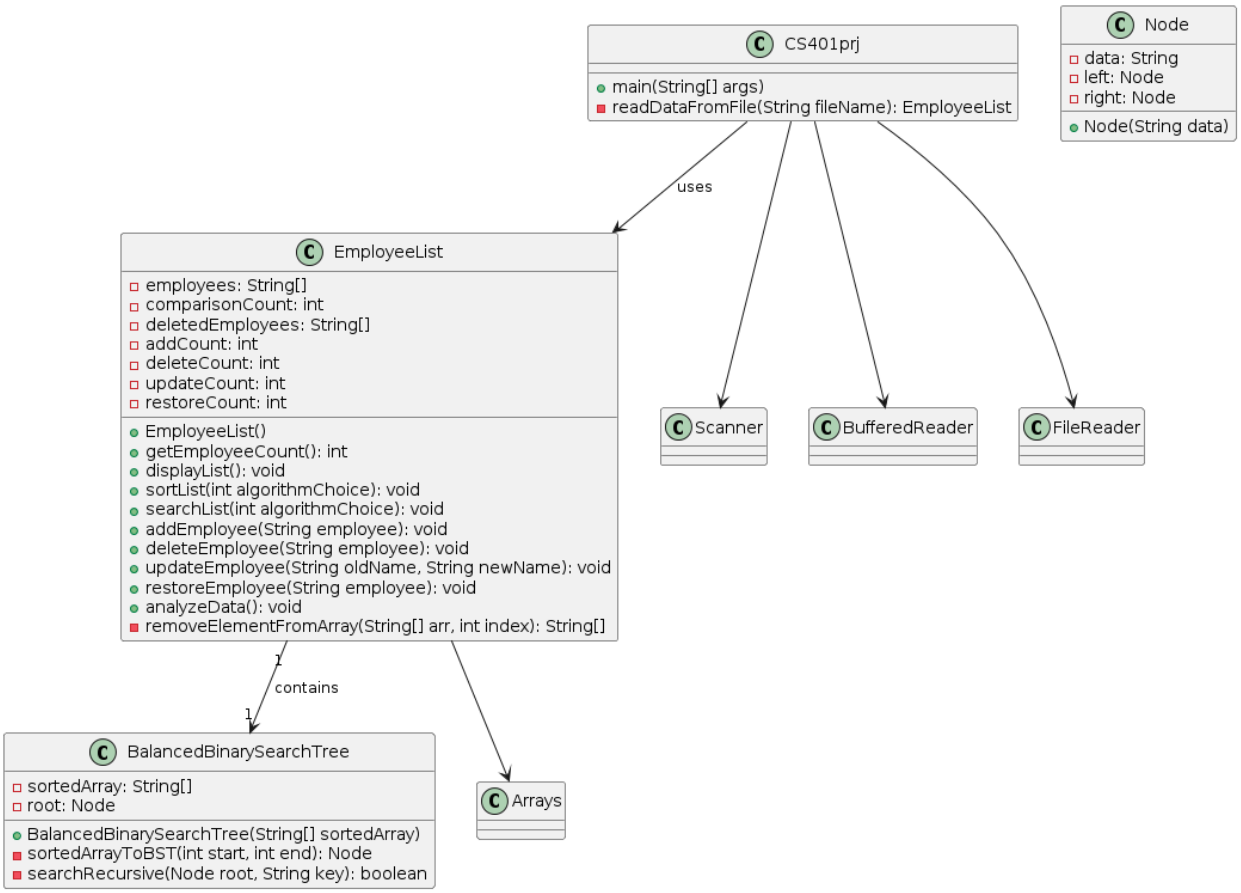
1. Functional Components:
   - BalancedBinarySearchTree:
     - Represents a balanced binary search tree (BST) data structure.
     - Provides methods for creating a balanced BST from a sorted array, searching for an element in the BST, and other related operations.
   - EmployeeList:
     - Manages a list of employee data, including adding, deleting, updating, and restoring employees.
     - Provides methods for sorting the employee list using different sorting algorithms (insertion sort and quick sort).
     - Provides methods for searching the employee list using different search algorithms (linear search, binary search, and hash function search).
     - Provides a method for analyzing the data operations performed on the employee list.

2. Data Types:
   - Node: A nested static class within the `BalancedBinarySearchTree` class, representing a node in the binary search tree. It has fields for the data, left child, and right child.
   - tree: A custom data type representing a binary search tree, defined using an inductive type in the `BalancedBinarySearchTree` class.
   - String: Used to represent employee names.
   - int: Used to represent various counters and indices.

3. Algorithms:
   - Balanced Binary Search Tree:
     - sortedArrayToBST: A recursive algorithm that converts a sorted array to a balanced binary search tree.
     - searchRecursive: A recursive algorithm that searches for an element in the binary search tree.
   - Sorting Algorithms:
     - insertionSort: An implementation of the insertion sort algorithm.
     - quickSort: An implementation of the quick sort algorithm, including the `partition` and `getMedianOfThree` helper methods.
   - Searching Algorithms:
     - linearSearch: An implementation of the linear search algorithm.
     - binarySearchTreeSearch: An implementation of the binary search algorithm, using the `BalancedBinarySearchTree` class.
     - hashFunctionSearch: An implementation of a hash function search algorithm, using a simple hash function.
   - Employee List Management:
     - addEmployee: An algorithm to add a new employee to the employee list.
     - deleteEmployee: An algorithm to delete an employee from the employee list and store the deleted employee in a separate array.
     - updateEmployee: An algorithm to update the name of an existing employee in the employee list.
     - restoreEmployee: An algorithm to restore a deleted employee from the separate array back to the employee list.
     - analyzeData: An algorithm to print out the counts of various operations performed on the employee list.
     - removeElementFromArray: A helper algorithm to create a new array without the element at the specified index.

3. Design Diagram:

UML Diagram: Below is the UML diagram that depicts a Java program structure for employee data management (`EmployeeList` class) and a balanced binary search tree (`BalancedBinarySearchTree` class) for efficient searching. The `EmployeeList` class manages employee data with methods for adding, deleting, updating, and analyzing data, while the `BalancedBinarySearchTree` class supports efficient searching operations.



PSEUDO CODE:

- **Main Class (CS401prj):**

```
class CS401prj
  main():
    readDataFromFile(fileName): EmployeeList
    employeeList: EmployeeList = readDataFromFile("employeeNames.txt")
    displayMenu(employeeList): void

  readDataFromFile(fileName): EmployeeList
    create new EmployeeList
    open file named fileName
    read each line from file
      add line to EmployeeList
    return EmployeeList
```

```
displayMenu(employeeList): void
  loop until exit option is chosen
    display menu options
    read user choice
    switch choice
      case 1: sortList(employeeList): void
      case 2: searchList(employeeList): void
      case 3: addEmployee(employeeList): void
      case 4: deleteEmployee(employeeList): void
      case 5: restoreEmployee(employeeList): void
      case 6: updateEmployee(employeeList): void
      case 7: analyzeData(employeeList): void
      case 8: exit program
    end switch
  end loop
```

- **EmployeeList Class:**

```
class EmployeeList
  properties:
    employees: array of strings
    deletedEmployees: array of strings
    addCount: integer
    deleteCount: integer
    updateCount: integer
    restoreCount: integer
    comparisonCount: integer

  constructor:
    employees = empty array
    deletedEmployees = empty array
    addCount = 0
    deleteCount = 0
    updateCount = 0
    restoreCount = 0
    comparisonCount = 0

  methods:
    getEmployeeCount(): integer
    displayList(): void
    sortList(algorithmChoice): void
      insertionSort(): void
      quickSort(low, high): void
        partition(low, high): integer
        swap(i, j): void
        getMedianOfThree(i, j, k): integer
    searchList(algorithmChoice): void
```

```
        linearSearch(key): boolean
        binarySearchTreeSearch(key): boolean
            sortedArrayToBST(start, end): Node
            searchRecursive(node, key): boolean
        hashFunctionSearch(key): boolean
    addEmployee(employee): void
    deleteEmployee(employee): void
    updateEmployee(oldName, newName): void
    restoreEmployee(employee): void
    analyzeData(): void

nested class BalancedBinarySearchTree
    properties:
        sortedArray: array of strings
        root: Node

    constructor(sortedArray): void
        sortedArrayToBST(start, end): Node
        search(key): boolean

class Node
    properties:
        data: string
        left: Node
        right: Node
    constructor(data): void
```

- **EmployeeList Class: (Other functions: add, del, update, restore, analyze)**

```
properties:
    employees: array of strings
    deletedEmployees: array of strings
    addCount: integer
    deleteCount: integer
    updateCount: integer
    restoreCount: integer

constructor:
    employees = empty array
    deletedEmployees = empty array
    addCount = 0
    deleteCount = 0
    updateCount = 0
    restoreCount = 0

addEmployee(employee):
    add employee to employees array
    increment addCount

deleteEmployee(employee):
    find index of employee in employees array
```

```
        if employee found
            add employee to deletedEmployees array
            remove employee from employees array
            increment deleteCount
        else
            print "Employee not found in the list."

    removeElementFromArray(arr, index):
        create new array newArr with length arr.length - 1
        copy elements from arr to newArr, excluding element at index
        return newArr

    updateEmployee(oldName, newName):
        find index of employee with oldName in employees array
        if employee found
            update employee name to newName
            increment updateCount
        else
            print "Employee not found in the list."

    restoreEmployee(employee):
        find index of employee in deletedEmployees array
        if employee found
            add employee back to employees array
            remove employee from deletedEmployees array
            increment restoreCount
            print "Employee has been restored."
        else
            print "Employee not found in the deleted list."

    analyzeData():
        print "Data Analysis Report:"
        print "Number of employees added: " + addCount
        print "Number of employees deleted: " + deleteCount
        print "Number of employees updated: " + updateCount
        print "Number of employees restored: " + restoreCount
```

- **Sorting Algorithms (Insertion Sort and Quick Sort):**

```
insertionSort(array):
    for i = 1 to length(array) - 1
        key = array[i]
        j = i - 1
        while j >= 0 and array[j] > key
            array[j + 1] = array[j]
            j = j - 1
        array[j + 1] = key

quickSort(array, low, high):
    if low < high
        pi = partition(array, low, high)
```

```
        quickSort(array, low, pi - 1)
        quickSort(array, pi + 1, high)

partition(array, low, high):
    pivot = array[high]
    i = low - 1
    for j = low to high - 1
        if array[j] < pivot
            i = i + 1
            swap array[i] with array[j]
    swap array[i + 1] with array[high]
    return i + 1
```

- **Searching Algorithms (Linear Search, Binary Search Tree Search, and Hash Function Search):**

```
linearSearch(array, key):
    for each item in array
        if item equals key
            return true
    return false

binarySearchTreeSearch(sortedArray, key):
    convert sortedArray to balanced binary search tree
    search key in binary search tree
    return result

hashFunctionSearch(array, key):
    hash key to get index in hash table
    search key in hash table at that index
    return result
```

4. Operational Details:
Employee Management System User Manual

- **Introduction**

The Employee Management System is a Java program that allows users to manage a list of employee names. It provides functionalities such as adding, deleting, updating, and restoring employees, as well as sorting and searching the list of employees using different algorithms.

- **System Requirements:**

- Java Development Kit (JDK) installed on your computer.
- Text file containing a list of employee names (e.g., employeeNames.txt)

- **Running the Program:**

1. Ensure that the JDK is properly installed on your computer.
2. Download the Employee Management System Java files (CS401prj.java and EmployeeList.java) to your local machine.
3. Open a command prompt or terminal window.
4. Navigate to the directory where you saved the Java files.
5. Compile the Java files by entering the following command: javac CS401prj.java EmployeeList.java
6. Run the compiled program by entering the following command: java CS401prj

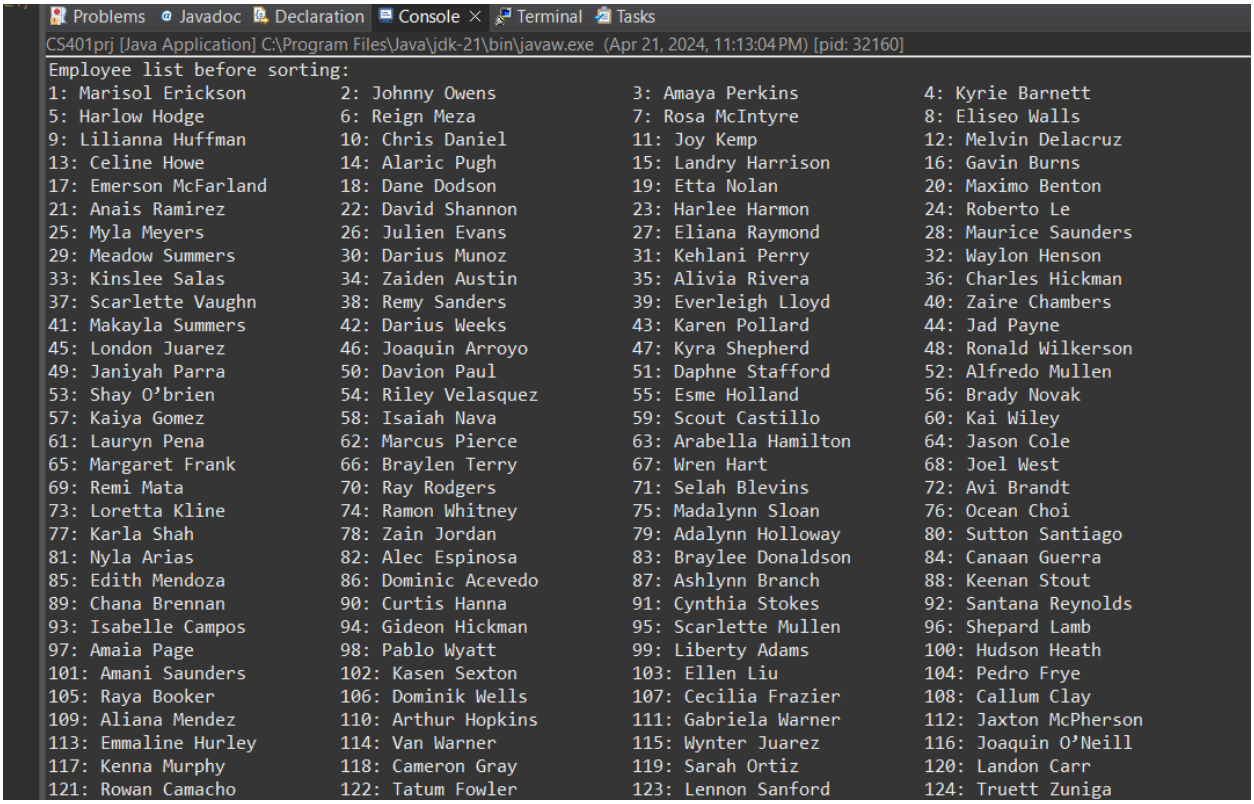
- **Using the Program:**

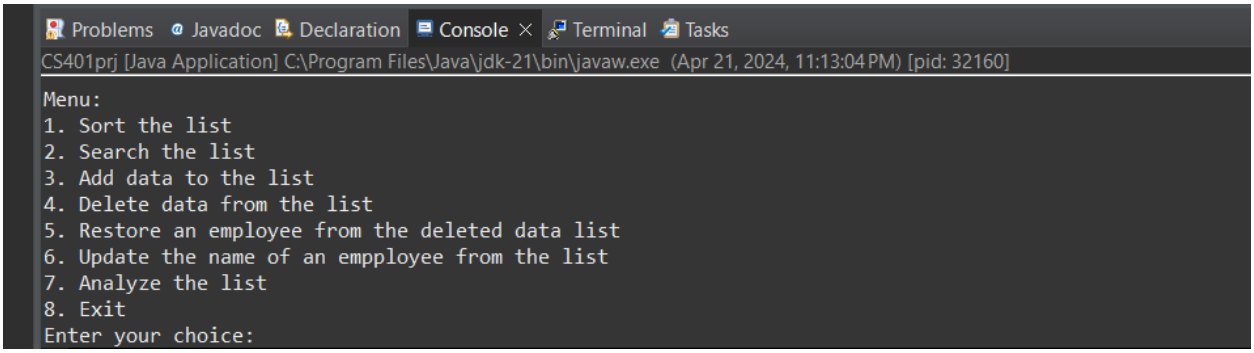
- Upon running the program, you will see a menu with different options:
  Sort the list
  Search the list
  Add data to the list
  Delete data from the list
  Restore an employee from the deleted data list
  Update the name of an employee from the list
  Analyze the list
  Exit
- Choose an option by entering the corresponding number.
  Follow the on-screen instructions to perform the selected operation.
  After each operation, the program will display the updated list of employees and any relevant information (e.g., total comparisons during sorting or searching, data analysis report).

- **Expected Results**

- Adding an employee: The program will add the employee to the list and display the updated list.
- Deleting an employee: The program will remove the employee from the list and display the updated list.
- Updating an employee: The program will update the employee's name and display the updated list.
- Restoring an employee: The program will restore the deleted employee and display the updated list.
- Sorting the list: The program will sort the list using the selected sorting algorithm and display the sorted list.
- Searching the list: The program will search for the specified employee using the selected searching algorithm and display the search result.
- Analyzing the list: The program will display a data analysis report showing the number of employees added, deleted, updated, and restored.

- **Screenshots:**





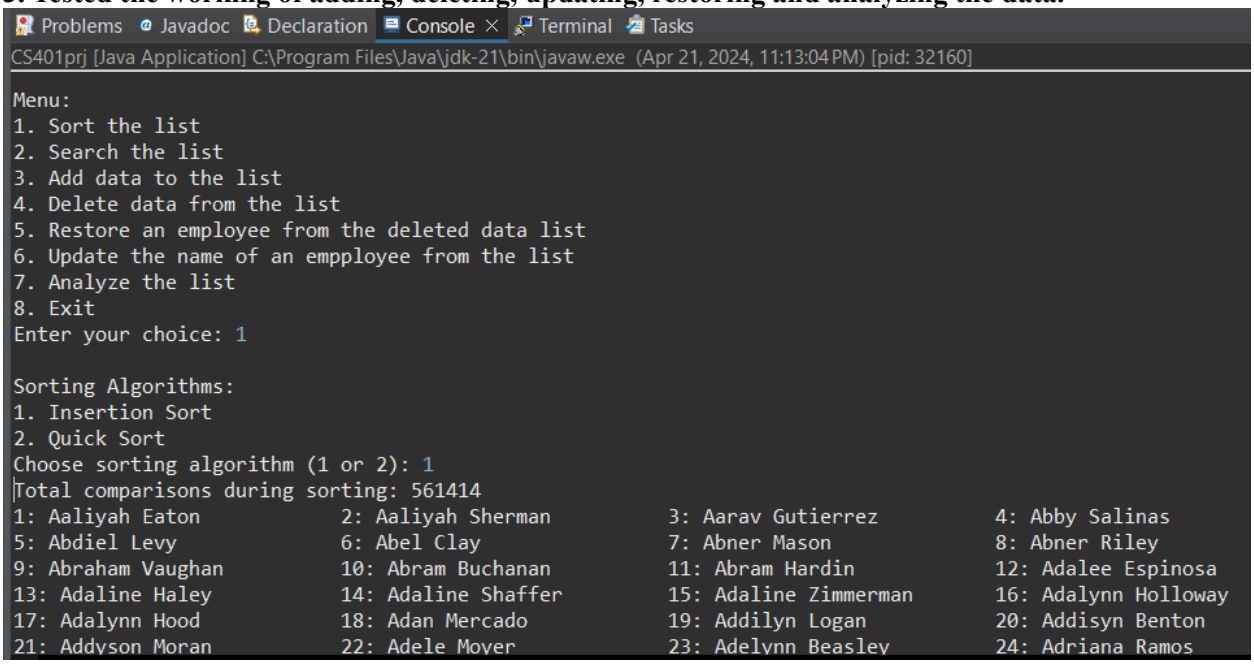
5. Testing:

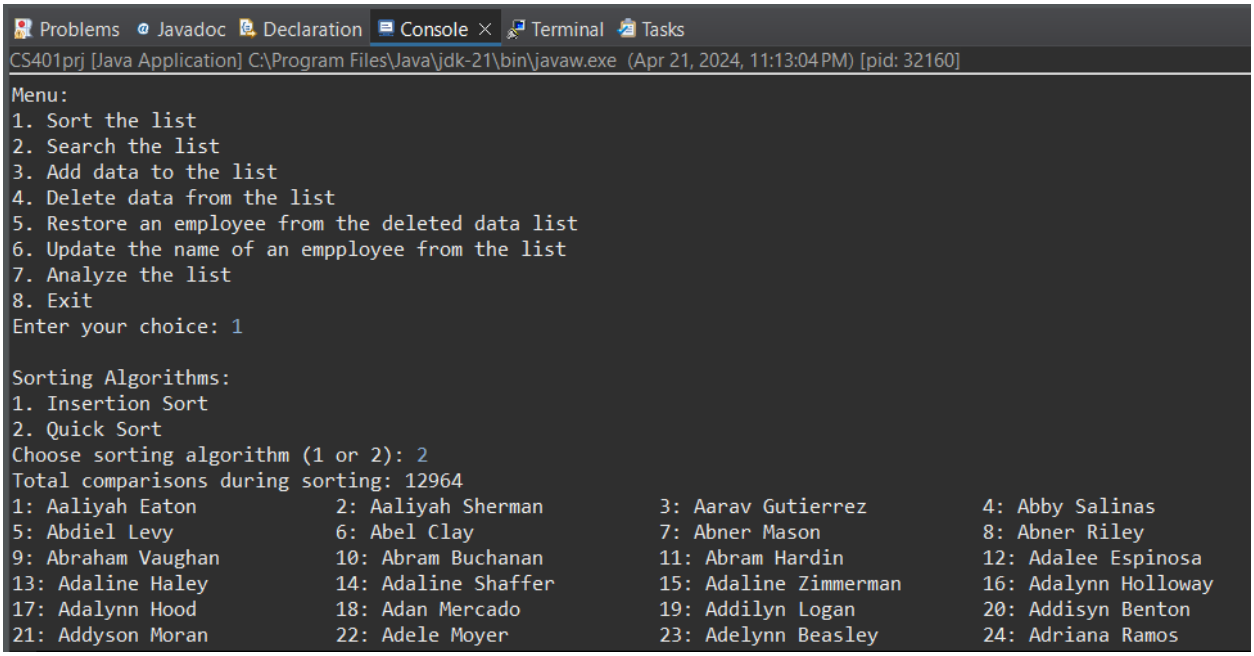
Test Cases:

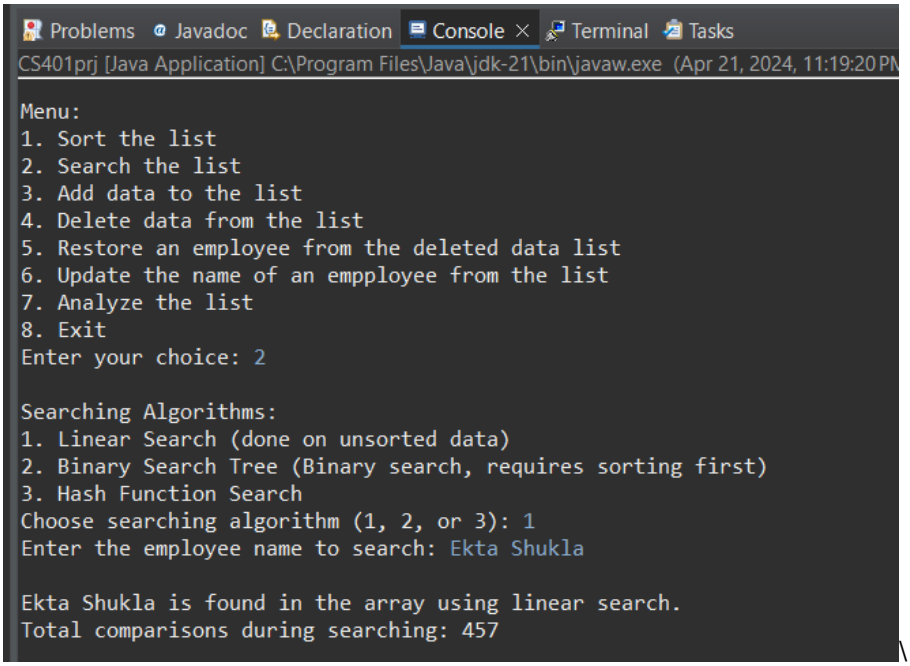
1. Tested sorting and searching with multiple inputs/elements.

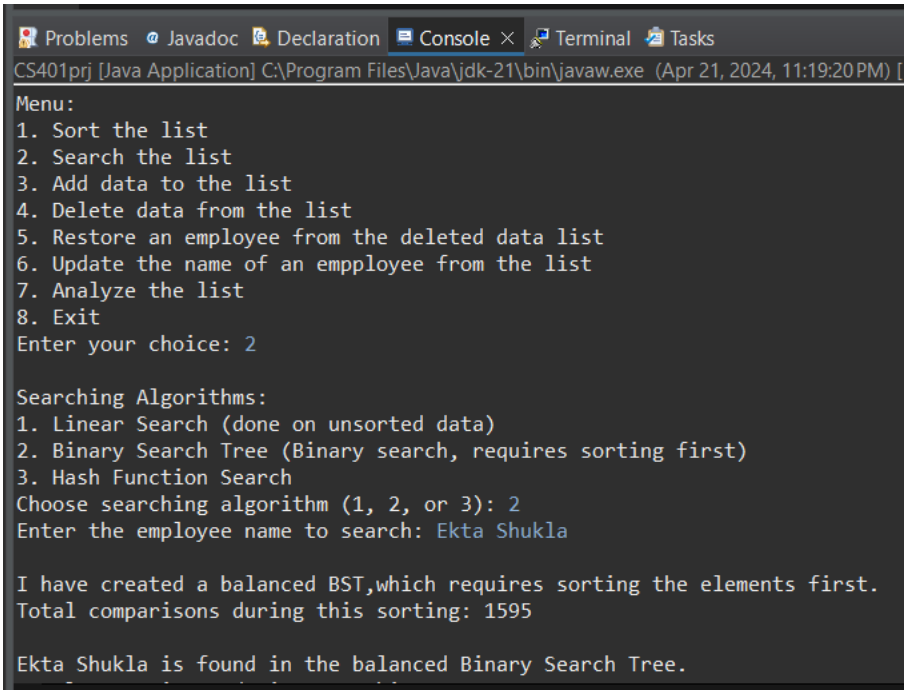
2. Tested the working of searching with different algorithms.

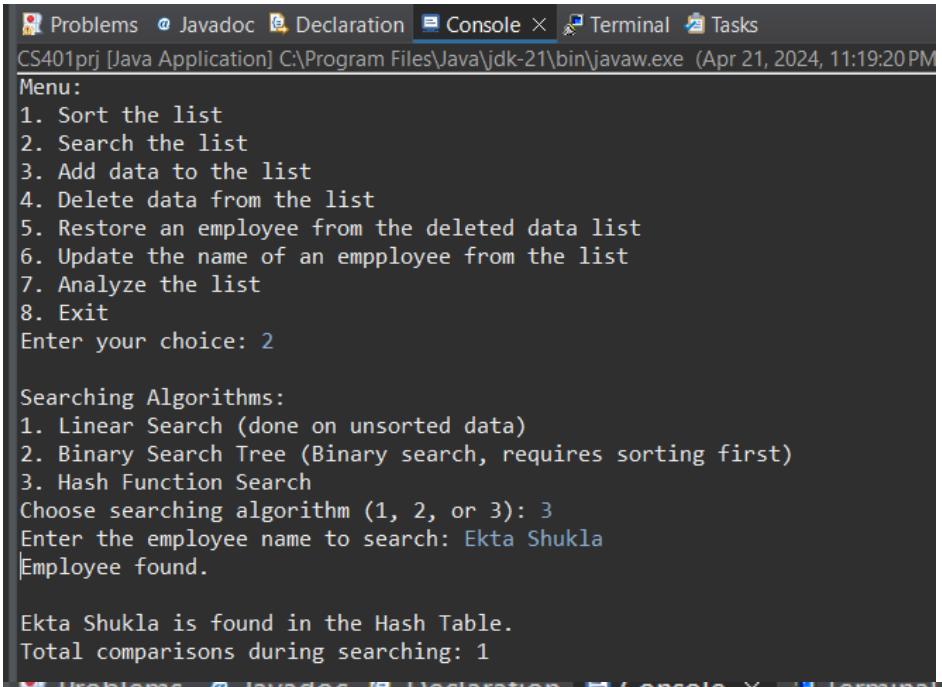
3. Tested the working of adding, deleting, updating, restoring and analyzing the data.






Problems
Javadoc
Declaration
Console
Terminal
Tasks
CS401prj [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe (Apr 21, 2024, 11:19:20 PM
Menu:
1. Sort the list
2. Search the list
3. Add data to the list
4. Delete data from the list
5. Restore an employee from the deleted data list
6. Update the name of an empployee from the list
7. Analyze the list
8. Exit
Enter your choice: 2
Searching Algorithms:
1. Linear Search (done on unsorted data)
2. Binary Search Tree (Binary search, requires sorting first)
3. Hash Function Search
Choose searching algorithm (1, 2, or 3): 1
Enter the employee name to search: Ekta Shukla
Ekta Shukla is found in the array using linear search.
Total comparisons during searching: 457


Problems
Javadoc
Declaration
Console
Terminal
Tasks
CS401prj [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe (Apr 21, 2024, 11:19:20 PM)
Menu:
1. Sort the list
2. Search the list
3. Add data to the list
4. Delete data from the list
5. Restore an employee from the deleted data list
6. Update the name of an empployee from the list
7. Analyze the list
8. Exit
Enter your choice: 2
Searching Algorithms:
1. Linear Search (done on unsorted data)
2. Binary Search Tree (Binary search, requires sorting first)
3. Hash Function Search
Choose searching algorithm (1, 2, or 3): 2
Enter the employee name to search: Ekta Shukla
I have created a balanced BST,which requires sorting the elements first.
Total comparisons during this sorting: 1595
Ekta Shukla is found in the balanced Binary Search Tree.


Problems
Javadoc
Declaration
Console
Terminal
Tasks
CS401prj [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe (Apr 21, 2024, 11:19:20 PM
Menu:
1. Sort the list
2. Search the list
3. Add data to the list
4. Delete data from the list
5. Restore an employee from the deleted data list
6. Update the name of an empployee from the list
7. Analyze the list
8. Exit
Enter your choice: 2
Searching Algorithms:
1. Linear Search (done on unsorted data)
2. Binary Search Tree (Binary search, requires sorting first)
3. Hash Function Search
Choose searching algorithm (1, 2, or 3): 3
Enter the employee name to search: Ekta Shukla
Employee found.
Ekta Shukla is found in the Hash Table.
Total comparisons during searching: 1

Problems
Javadoc
Declaration
Console
Terminal
<terminated> CS401prj [Java Application] C:\Program Files\Java\jdk-21\
1. Linear Search (done on unsorted data)
2. Binary Search Tree (Binary search, requires sortin
3. Hash Function Search
Choose searching algorithm (1, 2, or 3): 3
Enter the employee name to search: Ekta Shukla
Employee found.
Ekta Shukla is found in the Hash Table.
Total comparisons during searching: 1
Menu:
1. Sort the list
2. Search the list
3. Add data to the list
4. Delete data from the list
5. Restore an employee from the deleted data list
6. Update the name of an empployee from the list
7. Analyze the list
8. Exit
Enter your choice: 8
Exiting the program. Goodbye!

- **Troubleshooting**

If you encounter any issues while running the program, ensure that you have followed the installation and running instructions correctly.
Check that the employeeNames.txt file is in the correct format and contains the list of employee names.

- **Conclusion**

The Employee Management System provides a simple and efficient way to manage a list of employee names. By following the user manual, you can easily run the program and perform various operations on the employee list.