CS401- INTRO TO ADVANCED STUDIES-I PROJECT

**SORTING AND SEARCHING ALGORITHMS**

**AKSHITHA BEDRE SHIVAKUMAR**

**A20544641**

Submission date: 11/28/2023

**INDEX:**

**Software Development Life Cycle**

|  |  |  |
| --- | --- | --- |
|  | TOPIC | Page No. |
|  | Problem specification | 3 |
|  | Software specification | 3-4 |
|  | Design diagram | 5-6 |
|  | Operational document | 7-10 |
|  | Testing document | 10-11 |
|  | Future improvement | 11 |
|  | Project management | 12-13 |
|  | Complexity Analysis | 13 |

1. **Software Development Life Cycle**
2. **Problem specification**

The software development project described in the accompanying Java code addresses the issue of making it easier to sort and search across datasets. The main goals are to compare the complexity of various sorting algorithms, such as selection sort and heap sort, by counting the number of comparisons done during the sorting process. Furthermore, the software allows users to compare the effectiveness of various search algorithms, such as linear search, binary search, tree search, and hash function search, by allowing them to enter search components. The application allows for user customization by accepting input for dataset construction and supports a variety of data types such as integers, floats, and texts. It also serves an educational role by offering a clear user interface and allowing users to view the inner workings of sorting and searching algorithms, promoting comprehension and experimentation. The software prioritizes data integrity and correctness, as well as precise algorithmic implementation and handling of various scenarios. Overall, the project intends to be a useful educational tool as well as a practical examination of algorithmic efficiency in sorting and searching.

1. **Software specification**

**Sorting Operations:**

* Selection Sort: The implementation of the selection sort algorithm allows users to choose this option from the menu. The algorithm sorts the dataset, and the number of comparisons made during the sorting process is displayed with O(**n2**) complexity.
* Heap Sort: Heap sort can be chosen by the users to apply heap sort to the dataset, and, similar to selection sort, the number of comparisons made during sorting is presented with O (**n • log2n**).

**Searching Operations:**

* Linear Search: Users can perform a linear search on the dataset by selecting this option from the menu. The software prompts users to input an element for linear searching, and the index of the element (if found) and the number of comparisons is displayed with O(**n**) complexity.
* Binary Search Tree Search: This option enables users to conduct a binary search using a binary search tree. Users input an element, and the software indicates whether the element is found in the tree, along with the number of comparisons made with O(**log2n**) complexity.
* Hash Function Search: The software supports searching using a hash function. Users input an element, and the program utilizes a hash map to check for the presence of the element with O (**1**) complexity.
* Menu System: The program includes a menu system that helps users through numerous functions. Users can select sorting or searching procedures, enter search elements, and quit the software.
* Data Entry: The software allows users to input data by reading from a file. It includes a function, dataEntry(), that reads the dataset from a file and displays its contents in a well-organized format.
* Data Type Determination: The getDataType() function determines the data type of a string, categorizing it as an integer, float, or string.
* Comparison Function: The compare() function compares two strings based on their data types, handling integers, floats, and strings appropriately.

1. **Design diagram document:**

**UML Diagram
**

**UML DIAGRAM**

**A diagram of a program

Description automatically generated**

**FLOWCHART**

1. **Operational document   
   Program Execution Steps:**

* Start by downloading the jar file into your system named as project.jar and open the terminal app or command prompt on your desktop.
* We can run the program with the command line as **java -jar project.jar,** be very cautious about the name save on your computer.
* Once the command line has been run on your system it will ask for manual input from the user.
* Initially it will ask to enter the file name that needs to be accessed in order to get into menu options.
* Please enter the path of the file saved on your computer.
* Here I’m running the path file name as /**Users/akshithabedre/case1.txt.**
* Once the path has been typed, we will be prompted with the menu options to choose the required function to be performed.
* Once the required inputs are given the user will be prompted with the results.
* Once done you can choose to exit.

**Screenshots of the expected results: TestCase1**

* **Menu Options:**

A number on a white surface

Description automatically generated

* **Selection Sort:**

**A screenshot of a computer

Description automatically generated**

* **Heap Sort:**

A screenshot of a computer

Description automatically generated

* **Linear Search**

**A screenshot of a computer

Description automatically generated**

* **Binary Search**

**A screenshot of a computer

Description automatically generated**

* **Hash function Search**

**A screenshot of a computer program

Description automatically generated**

1. **Testing:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | CASE | INPUT | EXPECTED OUTPUT | FINAL OUTPUT |
| 1. | Read File path | /Users/akshithabedre/case1.txt | Start reading the data from the file | Successfully reads the data from the file |
| 2. | Prints Main Menu | Once we give the path it reads the data from the file and prints the main menu. | Print the Menu with options. | Successfully prints the menu options. |
| 3. | Select selection Sorting and Heap sort | Click on the option number 1 for selection sort and option number 2 for Heap sort. | Sorts the data and prints comparisons | Successfully sorts the data from the file and prints the comparison |
| 4. | Select Linear search and Binary Search | Click on the option number 3 for linear search and option number 4 for binary search. User will be asked to enter the element that needs to be searched | If present it says found at index number for linear along with the number of comparisons made. | Successfully found the number along with the number of comparisons made. |
| 5 | Select Linear search and Binary Search | Click on the option number 3 for linear search and option number 4 for binary search. If the number you are searching is not present in the data file. | It prompts not found | Successfully prints not found |
| 6. | Hash function search | Select the option number 5 for hash function search | If the number is found it prints the number along with the comparisons | Successfully prints found |
| 7. | Hash function search | Select the option number 5 for hash function search and the number you are trying to find is not present | It prompts not found | Successfully prints not found |
| 8. | Exit option | Once you click option number 6 it exits from the main menu | It prints Exit on the output | Successfully prints output. |

1. **Future:**

The project code has all the requirements expected and it satisfies most of the details requested. Whereas, it does have scope to improve at few areas such as,

* Modify the program to handle larger datasets efficiently and enhance memory management.
* Improve the user interface to make it more interactive and visually appealing.
* Implement a mechanism to re-prompt the user to enter the file name if the file is not found.
* Implement user-friendly error messages for better user experience.

1. **Project management:**

Developer: Student (AKSHITHA)

Application: Eclipse IDE

Language: Java

**Day 1-2: Project Setup and Requirements Understanding (2 hrs)**

Configure your programming environment (IDE, project structure), read, and comprehend project specifications. Design and plan the software architecture also identify and generate the classes required for sorting and searching algorithms.

**Day 3-6: Sorting Algorithm Implementations (6 hrs)**

Use the Selection Sort algorithm and create test cases to validate the implementation's correctness. Use the Heap Sort algorithm and implementation should be tested and debugged.

**Day 6-8: Searching Algorithm Implementations (5 hrs)**

* Linear Search should be used and create Linear Search test cases. We are also using Binary Search Tree Search.
* Use Hash Function Search, debug, and test search algorithms.

**Day 9-10: User Interface and Menu System (3 hrs)**

**Day 11: Documentation and Finalization ( 4 hrs)**

1. **Complexity analysis based on your results with the theory you learn.**

* The sorting and searching algorithms implemented in this Java code vary in complexity. Selection Sort, a comparison-based sorting algorithm, has a worst-case time complexity of O(n2) since it iterates across the entire array, doing n comparisons every pass.
* Despite its simplicity, it suffers from quadratic time complexity. Because it uses a binary heap data structure, Heap Sort has a more efficient average-case time complexity of O(n log n).
* The code includes searching techniques such as Linear Search, Binary Search Tree (BST) Search, and Hash Function Search. Linear Search has an O(n) time complexity because it compares elements sequentially until it finds the target element or reaches the end of the array.
* When the tree is balanced, BST Search has an average-case time complexity of O(log n), but it can degrade to O(n) in the presence of an imbalanced tree. When implemented with a well-designed hash function and few collisions, Hash Function Search achieves constant time complexity O(1) on average. However, the space complexity of the hash table may be O(n) in the worst case, depending on the number of elements and potential collisions.