KUSH PATEL

27684203

Algorithms and Complexity Assessment 1 Report

Design

The assignment given to us was to create a Search and Sort application in C# that will be used to help with the analysis of Network Traffic data. We were given multiple text files that have a range of numbers inside. There were 2 different lengths of text files given to us, 256 and 2048, which indicated how many numbers were in that file. Our task was to read these files and store the values from the text files into arrays in the program so we can manipulate them. We had to select a specific text file and then output every 10th value if the file selected was a 256 file, and every 50th value if the file selected was a 2048 file. Additionally, we had to select which type of sorting algorithm to run to sort the array in order and select which searching algorithm to run. The user gets to decide which number they wish to find, if the number is not in the list, then the closest number inside the list is chosen. The index of the chosen number will be displayed in the terminal.

To tackle this problem, I used decomposition to manage the workload easier. I first started my code with a choice to select 1 of 3 files. Each file had 2 types, 256 and 2048, these numbers represented how many numbers were within the text file. I then gave them another option to select 256 or 2048.

A screen shot of a computer code

Description automatically generated

Depending on these 2 user choices, a nested if statement would correctly call a method inside a class that corresponds to the users answer. I created 3 separate classes each for the 3 files and inside was 2 methods one for the SmallFile and one for the LargeFile. The method would read the file and then use a for loop to iterate through each value, parse it from a string to an integer and add it to the array. The method would then return the complete finalArray These 3 classes were named: Net\_1, Net\_2 and Net\_3.

A computer screen shot of text

Description automatically generatedA computer screen shot of a program code

Description automatically generated

Algorithmic choices

A computer screen shot of a program code

Description automatically generatedAfter the values are stored inside of finalArray, the user is given a choice between 4 different sorting algorithms. I had chosen to create a Bubble sort, Insertion sort, Quick sort and Selection sort algorithm. I created another class here named SortingAlgorithms and added a constructor array and assigned it the name Values. When calling the method, I made sure that the finalArray was assigned as the constructor in the class. I created 4 different methods inside the class: Bubble, Insertion, Quick and Selection. I have chosen these 4 sorting algorithms because they were the most common algorithms, I also had a more secure understanding of how these algorithms function so implementing them into C# for this project was challenging yet clear.

A screen shot of a computer program

Description automatically generated

When the user selects a number that is not in the array of numbers from the text file, the closest number in the array is selected instead. I achieved this by using another method to search for the number and return it.

A computer screen shot of a black background with green text

Description automatically generated

For my searching algorithms, I used 2 different methods inside the same class called SeachingAlgorithms. One of my searching algorithms is the Binary search and the other is the Linear search. My reasoning for choosing these searching algorithms was similar to why I chose the Sorting Algorithms, I was comfortable and had the most knowledge of these 2 algorithms so implementing them would have been more understandable than trying to use heap sort which is new to me and more complex.

A computer screen shot of a program code

Description automatically generated

The Binary search algorithm is a very efficient searching algorithm that splits the data set in half depending on if the value wanted is in that side of the array. The best case time space complexity of Binary search is O(1) as it would only take one comparison to find the target element in this scenario. Furthermore, my Binary search algorithm implements iteration using a while loop which is more efficient than using a recursive binary search algorithm. The average time complexity would be O(log 256) if the user selected the smaller 256 file size option; and O(log 2048) if the user selected the larger 2048 file size option. The worst case scenario for my binary search would be O(log N) because the comparisons required to reach the first element in the array is log(N) which means that it is also the worst case. *In-text citation: (GeeksforGeeks, 2019).*

Linear search algorithm is straightforward algorithm that goes through every value in the array until it has found the target element. This way, linear search is a very inefficient algorithm when the size of the dataset increases. The best case scenario for the space time complexity is O(1) because the element wanting to be found could be the first comparison the algorithm makes if the element is at the start of the array. In addition the average case for my algorithm is O(256) if the smaller file size is chosen and O(2048) if the larger file size is chosen because it would go through each data value in the array until it reaches the end. Finally, the worse case scenario for Linear search would be O(256) for smaller file size and O(2048) for the larger file size again due to the fact that the code will iterate through each value in the array, the worst case would mean that the value is at the very end of the array so therefore, O(N) would be the worst case. *In-text citation: (GeeksforGeeks, 2024).*

Bubble Sort is also a highly inefficient algorithm when the size of the data set increases. The algorithm works by comparing 2 adjacent values in an array from left to right and swapping if the left most value is larger than the right. Then it iterates this process until it has a fully sorted array. The best case time space complexity of Bubble sort is O(256) for the smaller file size and O(2048) for the larger file size. This is because the array may already be sorted and all it needs to do is check everything is in order by having no swaps in the first iteration. Therefore, the best case for Bubble sort is O(N). The average and worst case for bubble sort is O(256^2) for the smaller file and O(2048^2) for the larger file because this would be a scenario where the array is in descending order by default causing each iteration to run for as long as a swap has occurred. Therefore, the worst case would be O(N^2). *In-text citation: (GeeksforGeeks, 2023).*

Insertion sort is a simple sorting algorithm similar to linear search but instead it using more logic to make it more efficient. The algorithm starts from the second value on the left and compares if the number is smaller that it, if it is, then the number is placed behind the larger number. Because it goes through each element in the list, the best space time complexity is O(256) for my smaller files and O(2048) for the larger ones. This means that it is a linear complexity. The average and worst case is O(256^2) for smaller files and O(2048^2) for the larger ones because the array could be in reverse order, therefore making it a quadratic complexity. *In-text citation: (GeeksforGeeks, 2024).*

Quick sort is a divide and conquer algorithm that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array. Quick sort is done recursively which is why it may be inefficient which is why the space time complexity has a best case scenario of O(256 log 256) for the smaller files and O(2048 log 2048) for the larger files. The best case occurs when we select the pivot as the mean. The average case is O(N log N) too but the worst case is different. The worst case is O(256^2) for smaller files and O(2048^2) for larger files because the array is divided into 2 parts with one part consisting of N-1 elements and the other and so on. *In-text citation: (GeeksforGeeks, 2023)*

Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list. I have implemented the sorting algorithm using iteration rather than recursion which makes it easier to follow and understand. The best case scenario for this algorithm is O(256^2) for the smaller file size and O(2048^2) for the larger file size. The quadratic complexity is assigned due to the best case when the array is already sorted. The average case is also quadratic because when the array is in a random order, with no ascending or descending pattern. And the worst case for this algorithm is also quadratic because the array would initially be in descending order.

Space Complexity Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sorting Algorithm** | **Best Case** | **Worst Case** | **Average Case** | **Space Complexity** |
| Bubble Sort | O(N) | O(N^2) | O(N^2) | 1 |
| Insertion Sort | O(N) | O(N^2) | O(N^2) | 1 |
| Quick Sort | O(N log N) | O(N^2) | O(N log N) | log(N) |
| Selection Sort | O(N^2) | O(N^2) | O(N^2) | 1 |

Time Complexity Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Search Algorithm** | **Best Case** | **Worst Case** | **Average** |
| Binary Search | O(1) | O(log N) | O(log N) |
| Linear Search | O(1) | O(N) | O(N) |

GeeksforGeeks. (2019). Complexity Analysis of Binary Search. [online] Available at: <https://www.geeksforgeeks.org/complexity-analysis-of-binary-search/>. Accessed 19 Mar. 2024].

GeeksforGeeks. (2024). Time and Space Complexity of Linear Search Algorithm. [online] Available at: https://www.geeksforgeeks.org/time-and-space-complexity-of-linear-search-algorithm/. [Accessed 19 Mar. 2024].

GeeksforGeeks. (2023). Time and Space Complexity Analysis of Bubble Sort. [online] Available at: <https://www.geeksforgeeks.org/time-and-space-complexity-analysis-of-bubble-sort/>. [Accessed 19 Mar. 2024].

GeeksforGeeks. (2024). Time and Space Complexity of Insertion Sort. [online] Available at: <https://www.geeksforgeeks.org/time-and-space-complexity-of-insertion-sort-algorithm/>.

GeeksforGeeks. (2023). Time and Space Complexity Analysis of Quick Sort. [online] Available at: https://www.geeksforgeeks.org/time-and-space-complexity-analysis-of-quick-sort/?ref=lbp.

‌