**Project 1 Report**

**Algorithms CSCI 3330**

**Fall 2018**

Group Members:

Aung Kyaw Min

Hassan Khan

Chris Laxton

**Interpretation of Project**

We were given five different sources and we had to use three algorithms to sort all five sources. We also calculate the inversions by taking the sum of the rankings for the same documents but from different sources. The purpose of this project is to evaluate the source reliability by counting inversion through sorting five sources.

Most of the search engines such as Yahoo, Bing, Google, Baidu, DuckDuckGo, uses special algorithms to generate the search result. Google uses automate programs called spiders or crawlers. Google ranks the websites based on frequency and location of keywords of a web page, how long does it has existed and the number of other Web pages that link to the page in question. One of the methods to find the match results is to count the inversions of sorted list. An inversion can be defined as the number of swaps we need to try to get the sorted data. The number of inversion is the efficiency of the search algorithms. So, we can say it’s inversely proportional to the rank result of the search. The source is more reliable if it has fewer inversions.

**Methodology of the solution**

The three algorithms we used in this project were quick sort, merge sort and insertion sort. The average running time for both quick sort and merge sort is O(n log n) and insertion sort takes O(n²).

In the quick sort, we incremented the inversion by comparing the index of the pivot versus the index of the lesser or greater value.

In merge sort, we incremented the inversion if the value of the right-side element is bigger than the value on the left side.

In insertion sort, we incremented the inversion if the value we pick is higher than the previous value.

Methods we used:

public static ArrayList<Integer> MergeSortMain(ArrayList<Integer> a): to perform merge sort by dividing.

Input: unsorted arraylist

Process: Divide the arrayList into half until the smallest half becomes a single element.

Public static ArrayList<Integer> MergeSortMerge(ArrayList<Integer> a, ArrayList<Integer> b): to conquer the merge sort, bottom up.

Input: two unsorted small arraylists

Output: Sorted ArrayLists

Input: unsorted arrayList

Process: Pick the second element and compare with first element and if the second is less than the first, swap it and pick the third element, compare with first two elements and repeat the process. This process ends when the last element compares with all the previous elements.

Output: Sorted ArrayList

Product\_io.java: read the file

Product.java: create the row of the table

Table.java: contains all the methods for a table

Sort.java: contains all the sorting methods

Firstly, we inserted 5 sources into different ArrayLists. Then, we initiate and declare the product, and fill the product with data and we sorted the table according to the sum values of the products. After that, we created a new table to insert all the data which has been sorted according to the sum of the products. Later, we count the inversions of individual source by using three algorithms. We get the inversions by passing the data into the sorting functions.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sources** | **Inversions by quick\_sort** | **Inversions by merge\_sort** | **Inversions by insertion\_sort** | **Reliability** |
| **Source1** | 17443718 | 17443718 | 17443718 | 5.73272e-08 |
| **Source2** | 17577216 | 17577216 | 17577216 | 5.68918e-08 |
| **Source3** | 17503256 | 17503256 | 17503256 | 5.71322e-08 |
| **Source4** | 17779229 | 17779229 | 17779229 | 5.62454e-08 |
| **Source5** | 17459088 | 17459088 | 17459088 | 5.72768e-08 |

For the final case (reliability has not changed for all five sources)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sources** | **Inversions by quick\_sort** | **Inversions by merge\_sort** | **Inversions by insertion\_sort** | **Reliability** |
| **Source1** | 17351040 | 175351040 | 175351040 | 5.7633e-08 |
| **Source2** | 17597695 | 17597695 | 17597695 | 5.6826e-08 |
| **Source3** | 17463274 | 17463274 | 17463274 | 5.7263e-08 |
| **Source4** | 17974465 | 17974465 | 17974465 | 5.5635e-08 |
| **Source5** | 17376028 | 17376028 | 17376028 | 5.7551e-08 |

**Conclusion:**

Based on the results, no matter what algorithms we use, it gives us the same number of inversions.

The fastest sorting algorithm is the quick sort, the second is the merge sort and then the insertion sort.

**Discussion and Contributions by team members**

We divided the team into two groups. We had a programming team and report team. Hassan Khan is responsible for quick sort and merge sort, Chris Laxton is responsible for inversion sort and Aung Kyaw Min is responsible for the report.