Report for Project 1: Find source reliability by counting inversions during sorting

Group # 3

Students: Da-El Kim, Olufisayo Ayodele, Hang Phan, Alibek Zeinolla, Alikhan Adilbekov

**A. Interpretation of Project:**

In this project, we were given five sources and we had to modify three sorting algorithms to sort through the total of all given sources. Then we had to calculate the inversions to determine the source that was most reliable.

In the context of the Web, the main applications include building meta-search engines, combining ranking functions, selecting documents based on multiple criteria. When we use Google search engine to search information, Google always show us a long list links one by one. Suppose Google considers combining ranking results from different sources. At beginning they treat each source equally. In other words, they just sum all ranks from various sources for each web page, and use the summation to generate the combined rank. However, they want to investigate the reliability of each source and assign high weight to the most reliable one in the future rank combination. Here we simply define the reliability is inversely proportional to the number of inversions between the rank from a source with the combined rank. That is, the source is more reliable if it has fewer inversions.

**B. Methodology of the solution:**

All the reading file, sorting the data is accomplished in a class object. And after this object sorted the 5 sources according to the pages of the sum array(we used vector), the object is copied and used as a temporary value to calculate the inversion of 3 sorting algorithm for each sources.

The three sorting algorithms that were used in this project were quick sort, merge sort and insertion sort. The average running times for quick sort and merge sort are O() and insertion sort takes O(). We used insertion sort to sort the Sum vector (sum of all the values from 5 sources) and re-rank the five given sources based on the order after sorted Sum vector.

In quick sort, the inversion is incremented by the size of the middle and bigger array, because that is how much that data is moving over.

In merge sort, the inversion is incremented by the size of the left array whenever the value in the right array is smaller than the value compared in left array.

We used object oriented design for reusability and the major methods used were:

*vector<double> quickSort(vector<double> a)*: to perform quick sort.

input: unsorted vector

process: pick a pivot (first element in the vector), then divide the vector into 3 components, vector of all elements less than pivot, all equal to pivot, and greater than the pivot. Continue to divide until the vector has size of one or empty, then concur and sort vector.

output: sorted vector

*vector<double> mergeSort(vector<double> a)*: to perform merge sort.

input: unsorted vector

process: Divide the vector in hafts until the smallest haft is one element, then concur and sort vector bottom up.

output: sorted vector

*vector<double> insertionSort(vector<double> a)* : to perform insertion sort.

input: unsorted vector

process: pick an element of the vector to be an insertion value, start from the second element in the vector and compare to the first element, if the insert is less than the first, then replace the first by the insertion value, else choose the next element to be the insertion, and repeat the process.

output: sorted vector

*void insertionSortSum(vector<double> &a)*: while sorting the Sum vector rerank all of the sources according to the order of sum vector.

input: unsorted sum, and sorted sources

process: sort the sum using the insertion sort method

output: sorted sum with all sources followed the order of the sorted sum.

*int testInversion(vector<double> test)* : to get the number of inversion with a nested for loop

*int main()* : to call methods and perform the tasks and print output to console

We sort the Sum of all sources using the three sorting algorithms and get the number of inversions with each. The insertion sort (*insertionSortSum()*) is used to rerank all 5 given sources into the order of the sorted sum. By using each sorting algorithm, the number of inversions of all re-ranked 5 sources would be calculated.

The insertion sort has been picked to sort the sum and to rerank all 5 given sources into the order of the sorted sum. At first, we don’t know which source have more reliability, so the reliability starts at 1 for all 5 sources.

By now, three chosen sorting algorithms are used to calculate the inversion number for each reranked sources. Base on the result, and the formula reliability = , being used to calculate the reliability for each sources. After summing the 5 reliability, we compare it to the previous sum of reliability (which is 5, at start). If the sum of current reliability is not the same as the previous one, we go to the process of normalization. And it multiply the second reliability to the each data in the sources to have different weight for each sources. And then, the process to calculate reliability has been repeatedly performed its task until the reliability has not changed. In addition, the comparison of reliability of each source will be used to decide the most trustworthy source over all, the source has greater liability would be more trustworthy than the others.

**C. Experimental results:**

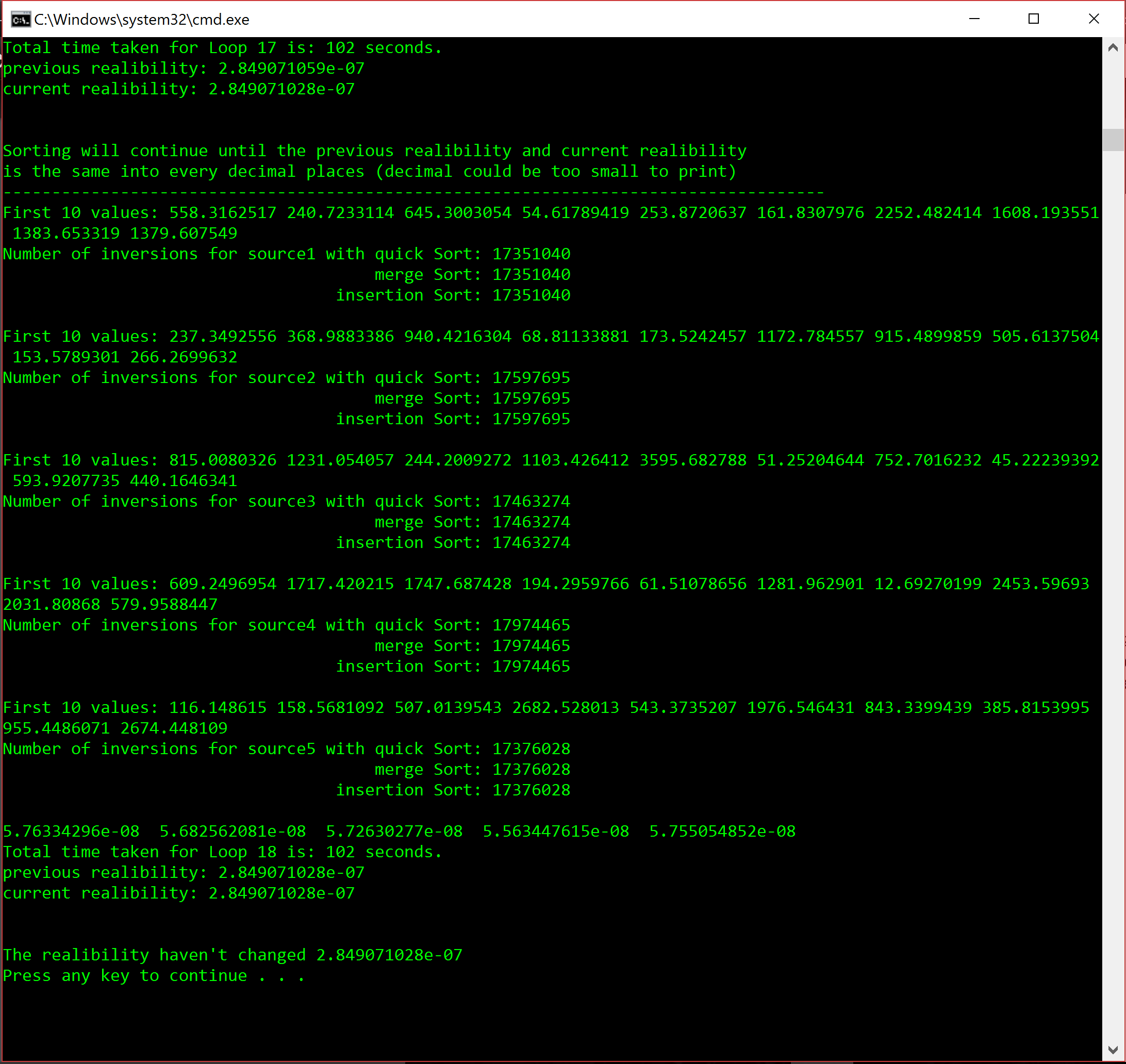
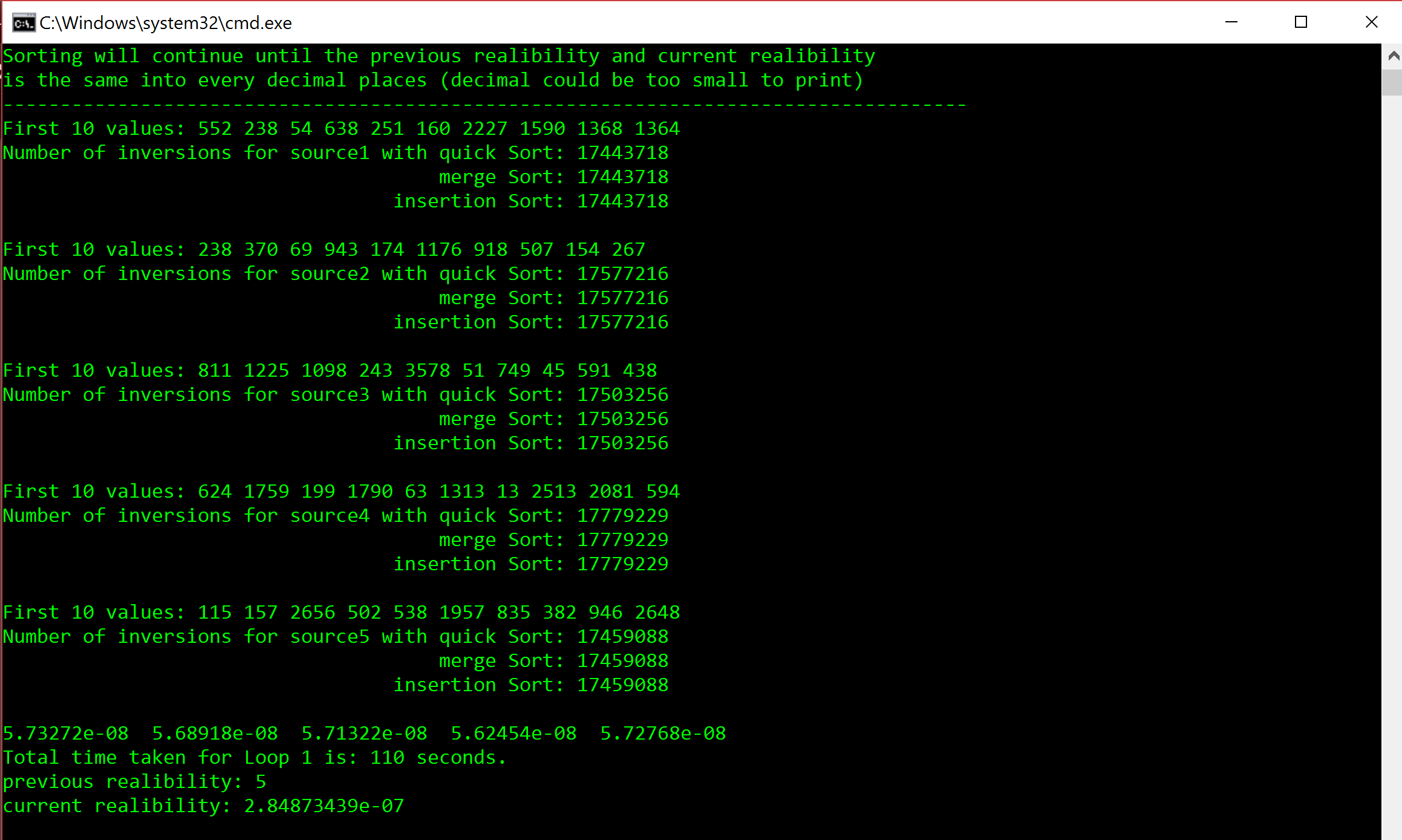
Code: C++ has been used to perform coding tasks for project.

For the initial case (Pre-reliability =1 for all five sources)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |



**D. Conclusions:**

Base on the result, we can conclude that for whichever the sorting algorithms has been used, it still gives the same number of inversion for the source.

For the value of 10,000 elements, the quick sort perform its task fastest, then is merge sort, and last one is insertion sort.

By comparing the reliability for all 5 sources, the most trustful source is source 2 because its reliability is highest comparing to others at the end.

**E. Discussion & Conclusion**

When we were coding we had to think of a way to sort the 5 sources according to the sorted order of Sum vector. And later on, we found a solution by, swapping the data in 5 sources whenever the Sum vector sorts. Also, it came to a surprise that calculating the inversion when sorting with different algorithm was not very hard task. If you know how the algorithm works all you have to do is increase the inversion value with certain values.