VOL. 15, NO. 24, DECEMBER 2020 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

THE DEVELOPMENT OF SYSTEM FOR ALGORITHMS VISUALIZATION USING SIMJAVA

Jamil Abedalrahim Jamil Alsayaydeh[1,2], Maslan Zainon[1,3], A. Oliinyk[6], Azwan Aziz[1,4], A. I. A. Rahman[1,2] and Zikri Abadi Baharudin[1,5]

[1]Fakulti Teknologi Kejuruteraan Elektrik dan Elektronik, Universiti Teknikal Malaysia Melaka, Hang Tuah Jaya, Durian Tunggal, Melaka, Malaysia

[2]Center for Advanced Computing Technology, Universiti Teknikal Malaysia Melaka, Hang Tuah Jaya, Durian Tunggal, Melaka, Malaysia

[3]Centre of Smart System and Innovative Design, Universiti Teknikal Malaysia Melaka, Hang Tuah Jaya, Durian Tunggal, Melaka, Malaysia

[4]Centre for Advanced Research on Energy, Universiti Teknikal Malaysia Melaka, Hang Tuah Jaya, Durian Tunggal, Melaka, Malaysia

[5]Centre for Robotics and Industrial Automation, Universiti Teknikal Malaysia Melaka, Hang Tuah Jaya, Durian Tunggal, Melaka, Malaysia

[6]Department of Software Tools, National University Zaporizhzhia Polytechnic, Zhukovskoho str., Zaporizhzhya, Ukraine

E-Mail: jamil@utem.edu.

ABSTRACT

Algorithm visualization which is a form of high-level dynamic visualization of software that uses user interface techniques to portray and monitor the computational steps of algorithms. Moreover, algorithm visualization systems are also useful tools in algorithm engineering, particularly at several stages during the design, implementation, analysis, tuning, experimental evaluation, and presentation of the algorithms process. Algorithms are a captivating use case for visualization. It does not simply fit data to a chart to visualize an algorithm, there is no main data set. Rather there are consistent principles that depict conduct. This is because algorithm visualizations are so uncommon, as designer's experiment with novel forms to better communicate. Algorithm visualization (AV) uses graphics to portray an algorithm's actions. AV holds the promise of helping computer science students understand algorithms more effectively and in more prominent profundity. The purpose of this study is to design a system for sorting algorithm visualization and implement the system.

Keywords: algorithms visualization, java code, sorting, search algorithms, perception algorithms.

INTRODUCTION

Algorithms can be classified and can be grouped together by utilizing comparative critical thinking methodology. The reason of algorithm classification is to feature the several ways in which an issues can be tackled. It classes in simple recursive, backtracking, divide and conquer, randomized and etc. Recursive algorithm is the algorithm that tackles the base cases straightforwardly and repeats with a simple sub issue. For example, recursive algorithms can count the number of elements in a list or test if a value occurs in a list. For backtracking algorithm, it depends on the depth-first recursive search. For example, backtracking algorithm can solve puzzles like crossword and Sudoku. Divide and conquer algorithms divides the problem into sub problems. After solving recursively, combine the solve problem. Normally used with quick and merge sort. Randomized algorithm utilizes a random number in any event once amid the calculation for decision making. For example, quick sort using a random number to choose a pivot. Powerful and large graphics does means that the system is more effective in learning. The most important part is that, algorithm visualization (AV) systems must provide educators with an accumulation of tools that make it moderately easy to consolidate AV and at that point assess its viability in an observational manner. Although there are many AV tools have been build and available over the internet, the adaptation of the tools in computer science world has been unsuccessful. There are several types of the sorting algorithm. Differences types of sorting algorithm have differences ways sorting method. Although there are several types of sorting, the efficiency of sorting is critical for enhancing the utilization of different algorithms which require input information to be in sorted list.

Other than that, a portion of the sorting algorithms like bubble sort, selection sort and heap sort, it sorts by moving elements to final position. For the insertion sort, quick sort, counting sort and radix sort keep items into an impermanent position nearer to definite position. it will re-scan and move items nearer to the definite position with every emphasis. As there are several methods to sort elements, factors such as algorithmic complexity, memory requirements, worst-case behavior and other problems should be considered in algorithm.

The objective of this study is to design a system of sorting algorithm visualization, implement the system and visualize the run time for each implemented sorting algorithm aims to help students understand algorithms more effectively.

PROPOSED SYSTEM

The proposed system involves the simulation of the different type of sorting algorithms codes. The scope has its limitations. Only 6 types of sorting algorithms codes are created which are bubble sort, insertion sort, selection sort, heap sort, merge sort and quick sort. Only

the software application development began with desktop applications used in this project, it can be used on standalone personal computer only. Once the synthesize and simulation of the codes for the software application have been run, the following animations will show how successfully data sets from distinct unsorted data can be sorted using distinct algorithms.

SORTING ALGORITHMS

Sorting algorithm is an effective algorithm in computer science. It carries out a vital undertaking that puts data of a list in a specific request or organizes a compilation of items into a specific request. Sorting data has been produced to orchestrate the array values in different routes for a database. For example, sorting will dictate an array of numbers in ascending or descending order. Usually, it sorts an array into ascending or descending order. Most basic sorting algorithms include two stages which are compare two items and swap two items or copy one item. It will keep on executing until the data has been fully arranged [1].

BUBBLE SORT

Bubble sort is a casual and common sorting algorithm. It frequently compares the pairs of adjoining elements and swap the element when they are in reversed order. The algorithm passes through the list until the entire list has been sorted. Bubble sort is quite inefficient for sorting large data but it is stable and adaptive [2].

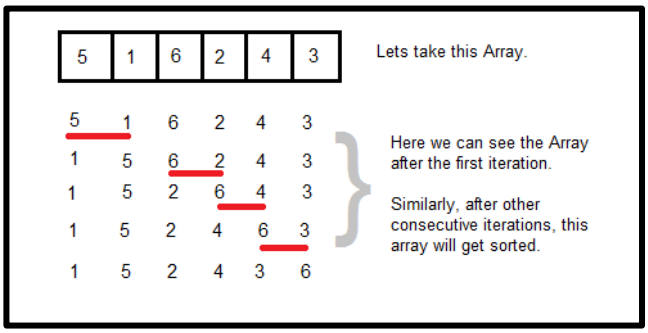

Figure-1. Bubble sort.

INSERTION SORT

Insertion sort continuously keeps up a sorted sub array in the small some portion of the list. It loops the input element by growing the sorted array and contrasts the present data of element with the biggest value in the sorted array. Every iteration moves an element from unsorted array to sorted array until all of the elements are sorted in the list [2].

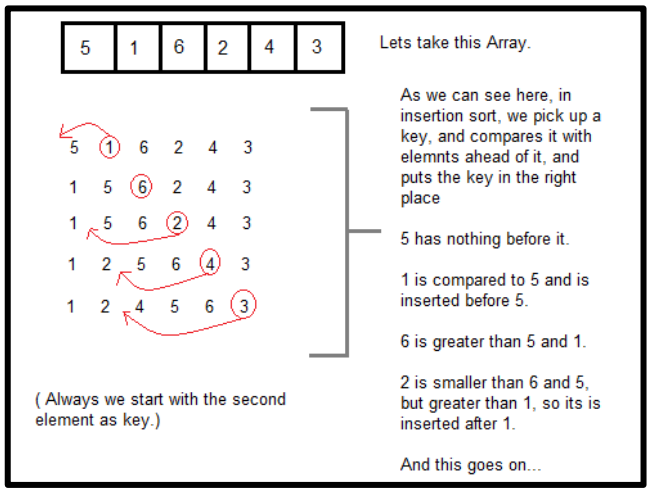

Figure-2. Insertion sort.

SELECTION SORT

Selection sort making just a single trade for each go through the array that improves on the bubble sort. It is a set up correlation based algorithm in which the list is separated into two segments, the arranged segment at the left end and the unsorted part at the correct end. Because of the average and worst case time complexities are of O(n2), this algorithm is not suitable for large data sets [2].

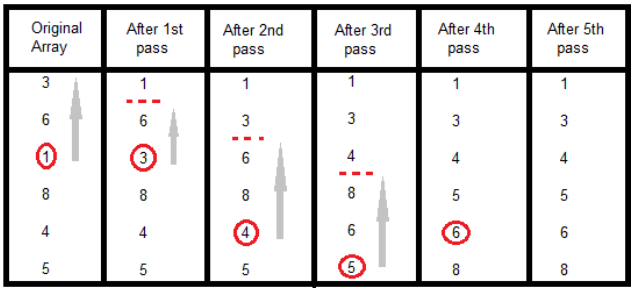

Figure-3. Selection sort.

The student easily understands the concept of selection sort by looking at the visualization. The text of a learning material is more effective if it is provided with graphics, animation or video for the student to learn [3].

HEAP SORT

Heap sorting performs sorting by arranging the input data in a heap. A heap is a complete binary tree with the greatest or smallest element at the root node. Heap sort creates an ordered list by deleting the root and placing it at the end of the array. Then the heap is realigned and the process is repeated. The best and worst case of time complexity for heap sort is O (n log 2n) [4].

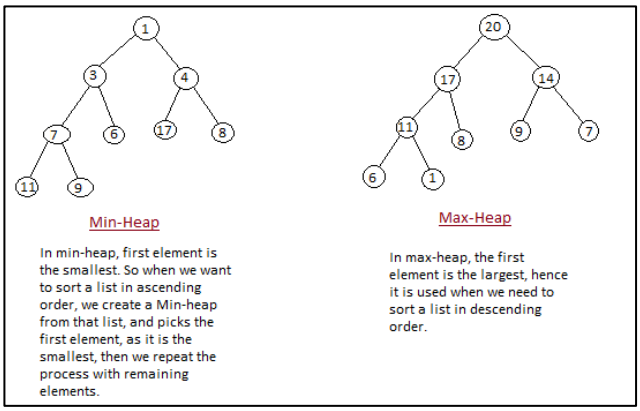

Figure-4. Heap sort.

MERGE SORT

Merge sort is similar to quick sort which is using divide and conquer algorithm to enhance the execution of sorting algorithms. It divides the array into 2 portion, calls itself for the 2 portion and then merges the 2 sorted portions. Once the 2 halves are sorted, merge is performed. Merges the sorted halves into one [5].

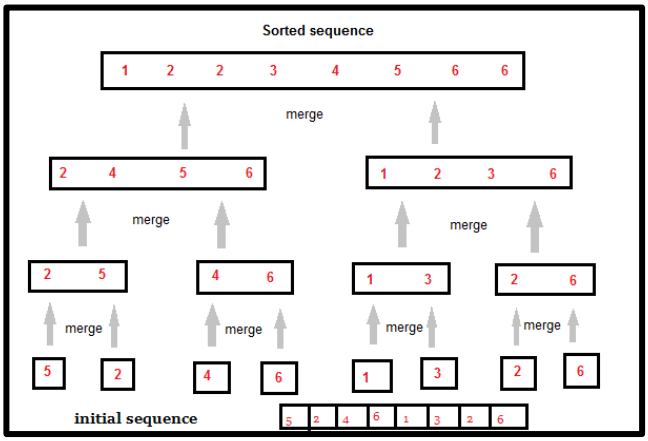

Figure-5. Merge sort.

QUICK SORT

Quick sort as the name suggests, it is a fast sorting algorithm. It is very fast and requires less additional space but it is not a stable search. Quick sort takes a divide and conquer approach to sorting lists. It divides the list of elements to be sorted into 2 segments and after that call the quick sort strategy recursively to sort the 2 segments. For example, divide the problem into 2 smaller segments and overcome by understanding the smaller segments.

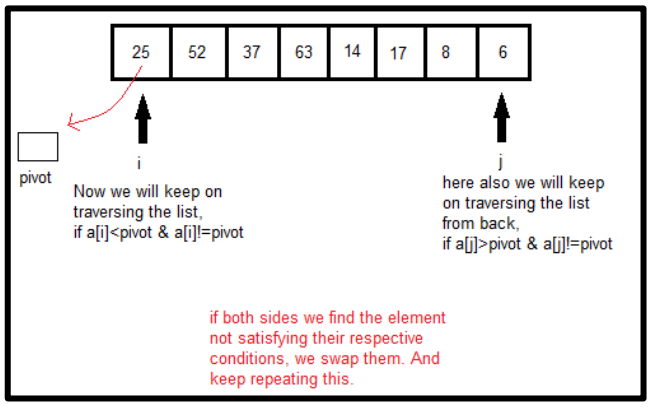

Figure-6. Quick sort.

CRITERIA FOR COMPARISON

Many algorithms that have the similar proficiency do not really have a similar speed and conduct on a similar information. The first and most important factor is that algorithms must be judged in light of their best-case, average-case, and worst-case efficiency. There are some algorithms that show different behavior of different combinations of inputs. Algorithms like quick sort perform particularly well for some sources of information, however awfully for others. Other algorithms, such as merge sort, are unaffected by the ordering of the input data [6].

Table-1. Complexity of algorithm [7].

T(n)	Name	Problems
O(1)	Constant	Easy-solved
O(log n)	Logarithmic	
O(n)	Linear	
O(n log)	Linear-log.	
O(n2)	Quadratic	
O(n3)	Cubic	
O(2n)	Exponential	Hard-solved
O(n!)	Factorial	

From the above analysis it can be said that fairly straightforward sorting techniques are Bubble Sort, Insertion Sort and Selection Sort, but they are not better and efficient for small elements lists. Quick Sort and Merge Sort are more complicated as compare with Bubble Sort, Insertion Sort and Selection Sort but it is much faster for large number of elements lists. On average the Quick Sort is the faster Sorting Algorithm. Bubble Sort is slower Sorting Algorithm [8] [9].

ECLIPSE

Eclipse is an integrated development environment (IDE) for developing applications using the Java programming language and other programming languages such as C/C++, Python, PERL, Ruby etc. [10]. The Eclipse platform that provides the basis for the Eclipse

IDE consists of plugins and is designed to be scalable with plugins. It was developed using Java; The Eclipse platform can be used to develop rich client applications, IDEs, and other tools. Eclipse can be used as an IDE for any programming language for which a plugin is available [11].

The Java Development Tools (JDT) provides a plugin that allows Eclipse to be used as a Java IDE, PyDev is a plugin that allows Eclipse to be used as a Python IDE, C/C++ Development Tools (CDT) is a plugin that allows Eclipse to be used for developing application using C/C++, the Eclipse Scala plugin allows Eclipse to be used an IDE to develop Scala applications and PHP eclipse is a plugin to eclipse that provides a complete PHP development tool [12].

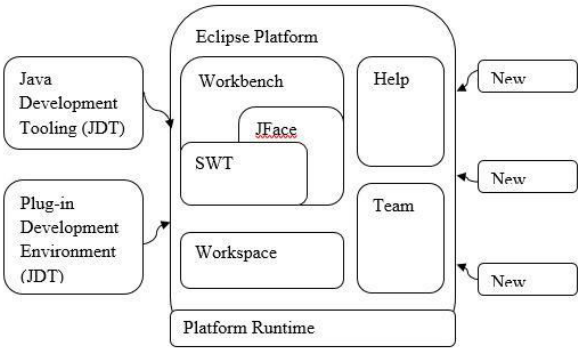

Figure-7. Eclipse system architecture.

Brian Faria provides visualization of sorting algorithms. This program has been implemented to activate sort algorithms as a learning aid for classroom instruction. A web-based animation tool was created to visualize four common sorting algorithms: Selection Sort, Bubble Sort, Insertion Sort, and Merge Sort. The animation tool would represent data as a bar-graph and after specifying a data-ordering and algorithm, the user can run an automated animation or step through it at their own pace. The limitation is it using Data-Driven Documents to bring JavaScript library for manipulating based on data using HTML, SVG and CSS [13].

Kanasz Robert introduces a Visualization and Comparison of Sorting Algorithms in c#. In this program, it is using GUI visualization for sorting algorithm to allow user to observe the result for each of the sorting algorithms. The limitation is it cannot run all types of sorting algorithm simultaneously [14].

Tim Martin introduces a Visualizing Sorting Algorithm. In this program, it shows the different sorting algorithm with D3. The author provides the animation for sorting algorithm: Bubble Sort, Insertion Sort, Selection Sort, Shell Sort, Merge Sort and Quick Sort to visualize how the sorting algorithms is work. The limitation is the animation for the visualizing sorting algorithm is still progress and the merge sort and quick sort does not function properly [15] [16]. Software visualization tools are intended to be used in the early stages of the programmer learning path, teaching students the basics of programming, algorithms, and the software development cycle [17]. Software visualization is the use of the crafts of typography, graphic design, animation and cinematography with modern human-computer interaction and computer graphics technology to facilitate both the human understanding and effective use of computer software [18].

RESEARCH METHODOLOGY

The main goal of this project is to design a system for sorting algorithm visualization as well as investigating and visualizing the best and worst case for each implemented sorting algorithm.

Design Steps

The software which has been used in this system is Eclipse. Java language is used to develop the algorithm visualization. For the Java code here, Java methods is used because of method can be used many times in the program by call the method. It is reusable and the program will become more readable.

After the sorting algorithms codes have been written and designed, they have been synthesized and simulated in the Eclipse software. Once the synthesize and simulation of the Java codes have been run, the user can choose the types of the sorting algorithm and searching algorithm to find the time complexity for the each of the sorting algorithms. The flow chart of designing this sorting algorithm visualization via Eclipse could be seen below, as in Figure-8.

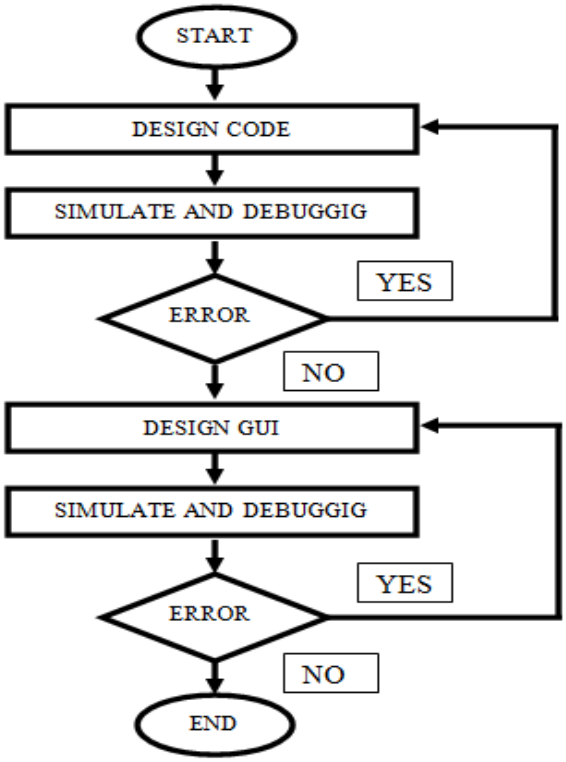

Figure-8. Flowchart of system for algorithms visualization.

Method of Sorting

While there are a large number of sorting algorithms, in practical implementations some algorithms predominate. Insertion sort is widely used for small datasets, whereas for large datasets, an asymptotically efficient sort is used, primarily heap sort, merge sort, or quick sort. Each algorithm uses a different mechanism to sort the data shown in the following pseudocode:

```
Bubble Sort(A) [4]
for i ← 1 to length[A]
        do for j ← length[A] down to i + 1
                do if A[j] < A[j - 1]
                        then exchange A[j] & A[j - 1]

Insertion Sort(A) [4]
for j ← 2 to length[A]
        do key ← A[j]
        Insert A[j] into the sorted sequence A[1 :  j - 1].
        i ← j - 1
        while i > 0 and A[i] > key
                do A[i + 1] ← A[i] i ← i - 1
                A[i + 1] ← key

Selection Sort (A) [4]
for i ← 1 to length[A] – 1
        do min ← i
                for j ← i + 1 to length[A]
                do if A[j] < A[min]
                                then  min ← j
exchange A[i] & A[min]


Mergesort(A, P, R) [4]
if p < r
        then q ← (p + r)/2
                MERGE-SORT(A, p, q)
                MERGE-SORT(A, q + 1, r)
                MERGE(A, p, q, r)

Heapsort(A) [4]
BUILD-MAX-HEAP(A)
for i ← length[A] downto 2
        do exchange A[1] ↔ A[i]
                heap-size[A] ← heap-size[A] - 1
                MAX-HEAPIFY(A, 1)

Quicksort(A, P, R) [4]
if p < r
        then q ← PARTITION(A, p, r)
                QUICKSORT(A, p, q - 1)
                QUICKSORT(A, q + 1, r)
```

The strategy of bubble sort is quite simple, however can be quite inefficient. The algorithm iterates through the array, contrast each pair of side by side elements, and if the elements are in the wrong order swaps them.

Insertion sort is another straightforward approach to sorting an array. Insertion sort starts at the beginning of the array and checks each element with the next and swaps them if necessary.

Selection sort is very intuitive and incredibly inefficient. The algorithm starts by creating an empty list to store values as they are sorted from the unsorted array. It then iterates through the entire unsorted array and adds the smallest (or largest) value to the sorted array.

Merge sort is one of the more effective sorting algorithms out there with a time complexity of only O(n log n). The algorithm starts by creating n lists with single values in each list and then proceeds to combine the lists into an array, sorting the values as the lists are combined.

The algorithms that based on comparison in sorting algorithm. Albeit to some degree slower by and by on most machines than a decent execution of quick sort. The time complexity for all case is O(n log n). Heap sort joint the effective of time for merge sort and effective of storage for quick sort.

Quick sort uses divide and conquer to pick up an indistinguishable focal points from the merge sort, while not utilizing extra memory. As an exchange off, nonetheless, it is conceivable that the list may not be partitioned into equal parts. At the points when this happens, the execution is reduced.

Solutions to sorting problems have pulled in a lot of research in the current year and in this procedure many sorting algorithm have started enhanced effectiveness. Throughout the year analysts have been contrasting and analyzing the sorting algorithm with decide their applicability to applications [19].

RESULT AND DISCUSSIONS

The result obtained from this system involving the output of the simulation of the Java code. There are six types of the sorting algorithms and two types of the searching algorithms.

Java Code

In order to design a system for sorting algorithm visualization and implement the system, the Java codes of this system have been constructed and synthesized using Eclipse software.

The Java code is written to create a form or visualization of sorting algorithms. The algorithm that had been done here is sorting and searching algorithms. Sorting algorithm is use to sort the 6 types of the sorting algorithm and searching algorithm is use to find the time complexity for each of the sorting algorithm. A set of integer number is created by using the java.util.Random class. The range for the number of data sorting is between 10,000 to 500,000 and use to perform the sorting algorithm. For the range of the random integer number is start from 0 to 100 only [20] [21].

Java methods are similar to what is called functions in other programming languages. It makes the programs more readable and easier to maintain. For example, if the code separate in few distinct Java methods with the names that match the conduct of the methods, it is simpler to make sense of what the program really doing. Besides, the “if else” method is used to let the user choose

for the types of sorting algorithms that want to sort. The user can choose only 1 type of the sorting algorithm at each of the time when it was generating the code. For the method of switch case in this program, it is used to calculate the elapsed time for each of the sorting algorithm [22] [23].

Sorting Algorithms in Java Code

```
//BUBBLE SORT
private void bubbleSort(int []array) {
int n = array.length;

    for (int passedNum = 1; passedNum < n; passedNum++) {
        for (int i = 0; i < n - passedNum; i++) {
            if (array[i] > array[i+1]) {
                int temp = array[i];    //swap
                array[i] = array[i+1];
                array[i+1] = temp;
            }
        }
    }
}
```

Figure-9. Java code for bubble sort.

```
//INSERTION SORT
public static void insertionSort(int[] array) {
for (int i = 1; i < array.length; i++){
    int j = i;
    int B = array[i];

    while ((j > 0) && (array[j-1] > B)){
        array[j] = array[j-1];
        j--;
        }
    array[j] = B;
    }
}
```

Figure-10. Java code for insertion sort.

```
//SELECTION SORT
private void selectionSort(int []array) {
    for(int i = 0; i < array.length - 1; i++) {
        int minPosition = i;

        for(int j = i + 1; j < array.length; j++) {
            if(array[minPosition] > array[j])
            minPosition = j;    //location for the new minimum number
        }
        if (minPosition != i) {
            //swap the current element with the smallest remaining
            int temp = array[i];
            array[i] = array[minPosition];
            array[minPosition] = temp;
        }
    }
}
```

Figure-11. Java code for selection sort.

```
//MERGE SORT
private static void mergeSort(int[]array) {
    merge(array, 0, array.length);
}

//Merge data for merge sort
private static void merge(int[] array, int low, int high) {
    if (high - low == 1) {
        return;     //only one element in the array, return.
    }
    int mid = low + (high - low) / 2;   //find middle number
    merge(array, low, mid);     //sort 1st part
    merge(array, mid, high);    //sort 2nd part
    MergeNumber(array, new int[array.length], low, mid, high);
}
```

Figure-12. Java code for merge sort.

```
//HEAP SORT
private static void heapSort(int[] array) {
    heapSize = array.length;
    BuildMaxHeap(array);     //build MaxHeap
    for (int i = array.length-1; i>0; i--) {
     //swap biggest number with the current last one
    int temp = array[0];
    array[0] = array[i];
    array[i] = temp;
    heapSize--;
    MaxHeapify(array,0);     //find new biggest number
    }
}

//Build MaxHeap
private static void BuildMaxHeap(int[] array) {
for(int i = array.length/2-1;i>=0;i--) {
        MaxHeapify(array, i);
    }
}
```

Figure-13. Java code for heap sort.

```
//QUICK SORT
private static void quickSort(int[]array) {
    if(array== null || array.length<=1) {
        return;
    }
    quick(array, 0, array.length-1);
}

private static void quick(int[] array, int low, int high) {
    if (high <= low){
        return;
    }
    int j = partition(array, low, high);
    sort(array, low, high);
    assert isSorted(array, low, high);
}
```

Figure-14. Java code for quick sort.

Algorithms Visualization

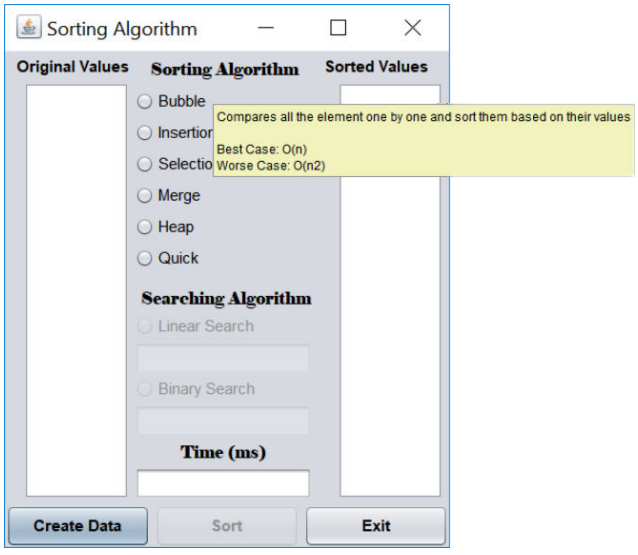

Figure-15. Visualization of algorithm.

Figure-15 shows the visualization of the sorting algorithm. There were 5 parts in this user interface which are original values, sorted values, sorting algorithm, searching algorithm and time.

PERFORMANCE ANALYSIS

The Tables below show the run time for six types of the sorting algorithm using linear and binary searching algorithm. The data collected and analysis in this table by using different number of random integer numbers that create by the program. The smaller range of the number of data sorting is between 10 to 50 and the larger range of the number of data sorting is between 10,000 to 500,000.

Table-2. Run time in different range of number (smaller number - linear search).

Number of elements (n)	Time (ms)					
	Bubble	Insertion	Selection	Merge	Heap	Quick
10	0.003	0.003	0.007	0.006	0.009	0.010
20	0.007	0.005	0.006	0.014	0.015	0.023
30	0.020	0.006	0.011	0.016	0.038	0.046
40	0.031	0.010	0.028	0.032	0.043	0.052
50	0.045	0.018	0.034	0.044	0.039	0.063

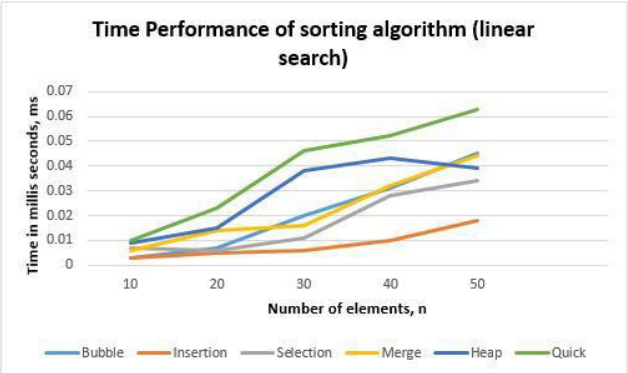

Figure-16. Graph for time performance of sorting algorithm in smaller element of numbers (linear search).

Table-3. Run time in different range of number (smaller number - binary search).

Number of elements (n)	Time (ms)					
	Bubble	Insertion	Selection	Merge	Heap	Quick
10	0.005	0.009	0.012	0.016	0.025	0.049
20	0.007	0.011	0.017	0.020	0.031	0.052
30	0.014	0.013	0.011	0.030	0.042	0.066
40	0.026	0.017	0.020	0.029	0.049	0.063
50	0.040	0.027	0.026	0.028	0.055	0.079

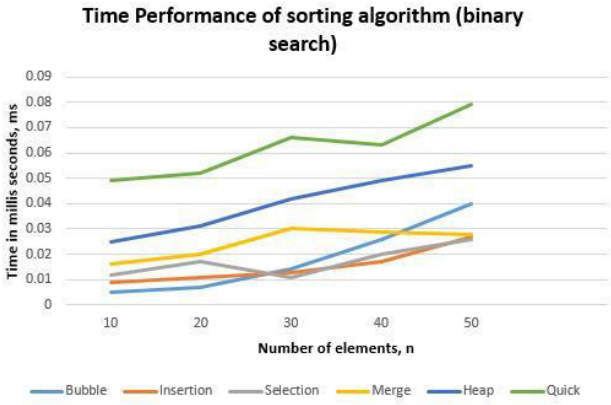

Figure-17. Graph for time performance of sorting algorithm in smaller element of numbers (binary search).

Table-4. Run time in different range of number (larger number - linear search).

Number of elements (n)	Time (ms)				Number of elements (n)	Time (ms)	
	Bubble	Insertion	Selection	Merge		Heap	Quick
10 000	150	34	74	111	1 000 000	151	189
20 000	760	130	330	400	2 000 000	307	400
30 000	1670	260	580	820	3 000 000	467	612
40 000	2820	450	1030	1410	4 000 000	633	770
50 000	4450	720	1580	2220	5 000 000	803	1050

Table-4 shows that the run time for different types of sorting algorithm with the different number of elements using linear search. The range for the number of elements in first group (bubble, insertion, selection and merge) is between 10,000 to 50,000 while the range for the number of elements in second group (heap and quick) is between 100,000 to 500,000.

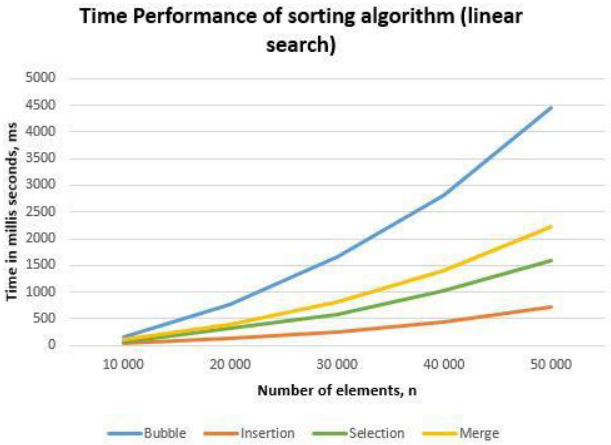

Figure-18. Graph for time performance of bubble, insertion, selection and merge sort in larger element of numbers (linear search).

Figure-18 is the graph for time performance of first group using linear search. The result show insertion is the faster and bubble sort is the slower one compared to others.

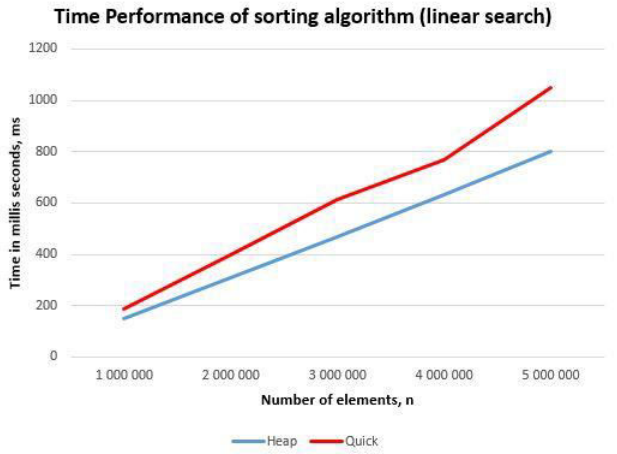

Figure-19. Graph for time performance of heap and quick sort in larger element of numbers (linear search).

Figure-19 is the graph for time performance of second group. The result show heap sort is the fastest sorting algorithm compare to quick sort even through the number of elements reached until 500,000, the heap sort also performance well.

Table-5. Run time in different range of number (larger number - binary search).

Number of elements (n)	Time (ms)				Number of elements (n)	Time (ms)	
	Bubble	Insertion	Selection	Merge		Heap	Quick
10 000	157	35	70	108	1 000 000	154	193
20 000	701	129	271	418	2 000 000	313	419
30 000	1637	263	583	952	3 000 000	486	635
40 000	3000	496	1018	1865	4 000 000	645	804
50 000	4549	760	1550	2726	5 000 000	823	1185

Table-5 show that the run time for different types of sorting algorithm with the different number of elements using binary search.

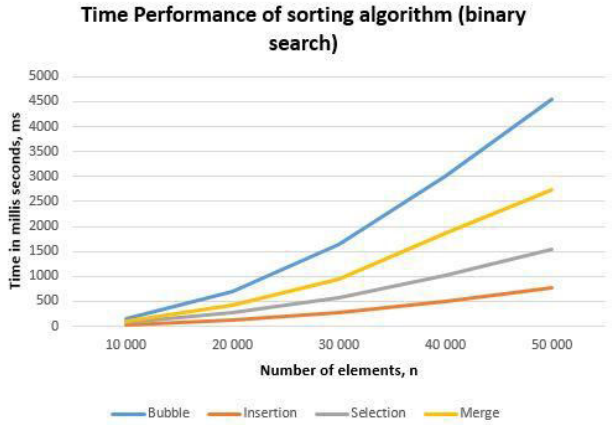

Figure-20. Graph for time performance of bubble, insertion, selection and merge sort (binary search).

Figure-20 is the graph for time performance of first group using binary search. The result show insertion is the faster and bubble sort is the slower one compared to others.

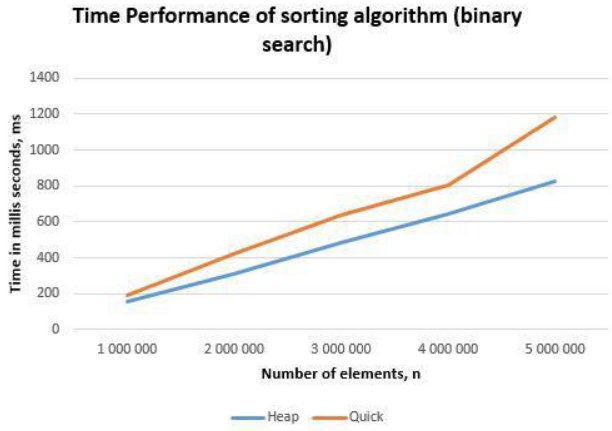

Figure-21. Graph for time performance of heap and quick sort in larger element of numbers (binary search).

Based on the Figure-16 and Figure-17 the data sort in smaller number of elements, insertion sort is the fastest sorting algorithm compare to other. Based on the Figure-19 and Figure-21 the result show that heap sort and quick sort are the fastest sorting algorithm compare to other even the number of the data sorting is bigger, the sorting run time also can perform faster. This result clearly show that heap sort and quick sort are more efficiency because of the best case of the time complexity is O(n log n).

Supposedly, binary search is faster than linear search, but the result shows that linear search is faster than binary search. This is because when the number of the elements is increasing, the program will create few of same integer number. So that, when linear search is perform, it will take the first same number as the last number. This make the linear search become faster than binary search and this result is not accurate compare to binary search.

CONCLUSIONS

From the results that have been obtained throughout this project, the Java codes of the sorting algorithm user interface were successfully designed, synthesized and analyzed. The sorting algorithm for the Java program has proven that each of the types for sorting algorithm is working precisely. Not only is that, based on the table of the result for run time given by each of the sorting algorithm in line with the theory. This result verified that the half of the objectives for this project have been achieved. This is because this project does not have any animation or visualization for sorting algorithm. It can only measure the performance of the run time based on different algorithms. As a conclusion, the development of system for algorithms visualization using SimJava has been successfully made by wanted targets and particulars. Aside from that, this study can be enhanced to make further developed and dependable.

There are some suggestions in order to improve this study. First of all, the design of this sorting algorithm visualization should be transform to sorting algorithm animation in order to use as a good teaching tool and able to gain a good understanding of the sorting algorithm for computer science student. Besides that, this program should be done using Netbeans software. This is because the Netbeans has their own user interface design. Netbeans can be used when prototyping and designing applications on top of the NetBeans Platform. It has drag-and-drop capabilities and point-and-click features to make the ideal environment for user interface design or visualization.

ACKNOWLEDGMENT

The authors would like to thank Centre for Research and Innovation Management (CRIM) for the support given to this research by Universiti Teknikal Malaysia Melaka (UTeM). We thank also those who contributed in any other forms for providing their continuous support throughout this work.

REFERENCES

[1] Sareen P. 2013. Comparison of sorting algorithms (on the basis of average Case). International Journal of Advanced Research in Computer Science and Software Engineering. 3(3): 522-532.

[2] Mahfooz Alam and Ayush Chugh. 2014. Sorting Algorithm: An Empirical Analysis. International Journal of Engineering Science and Innovative Technology. 3(2): 118-126.

[3] Hadi Sutopo. 2011. Selection Sorting Algorithm Visualization. The International Journal of Multimedia & Its Applications (IJMA). 3(1): 22-35.

[4] Reyha Verma and Jasbir Singh. 2015. A Comparative Analysis of Deterministic Sorting Algorithms based on Runtime and Count of Various Operations. International Journal of Advanced Computer Research (IJACR). 5(21): 380-385.

[5] Jamil Abedalrahim Jamil Alsayaydeh, Mohamed Nj, Syed Najib Syed Salim, Adam Wong Yoon Khang, Win Adiyansyah Indra, Vadym Shkarupylo and Christina Pellipus. 2019. Homes Appliances Control Using Bluetooth. ARPN Journal of Engineering and Applied Sciences. 14(19): 3344-3357.

[6] Neha Gupta. 2016. Comparison and Enhancement of Sorting Algorithms. International Journal on Recent and Innovation Trends in Computing and Communication. 4(2):162-166.

[7] Niraj Kumar and Rajesh Singh. 2014. Performance Comparison of Sorting Algorithms on The Basis of Complexity. International Journal of Computer Science and Information Technology Research. 2(3): 394-398.

[8] Waqas Ali, Tahir Islam, Habib Ur Rehman, Izaz Ahmad, Muneeb Khan, Amna Mahmood. 2016. Comparison of different sorting algorithms. International Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE). 5(7): 63-71.

[9] Abbasloo and Pour. 2013. Sorting algorithms and difference of this algorithms. International Journal of Computer and Electronics Research. 2(5): 0-3.

[10] Core Java and Database Concepts, 'Eclipse Integrated Development Environment', Simply Easy Learning. I (2015): 1-81.

[11] Jamil Abedalrahim Jamil Alsayaydeh, Adam Wong Yoon Khang, Win Adiyansyah Indra, J.Pusppanathan, Vadym Shkarupylo, A K M Zakir Hossain, Saravanan S/O Saminathan. 2019. Development of Vehicle Door Security using Smart Tag and Fingerprint System.International Journal of Engineering and Advanced Technology (IJEAT). 9(1): 3108-3114. DOI: 10.35940/ijeat.E7468.109119.

[12] Kavitha S., Sindhu S. 2015. Comparison of Integrated Development Environment (IDE) Debugging Tools: Eclipse Vs Netbeans. International Research Journal of Engineering and Technology (IRJET). 2(4): 432-437.

[13] Faria Brian. 2017. Visualizing Sorting Algorithms. Honors Projects Overview. 127. https://digitalcommons.ric.edu/honors_projects/127.

[14] Kanasz Robert. 2015. Visualization and Comparison of Sorting Algorithms in C# - CodeProject. Algorithm. pp. 1-10 <https://www.codeproject.com/Articles/132757/Visualization-and-Comparison-of-sorting-algorithms> [accessed 12 May 2017].

[15] T. Martin. 2015. Visualizing Sorting Algorithms. San Franciso. <http://timothymartin.azurewebsites.net/visualizing-sorting-algorithms/> [accessed: 12-April-2017].

[16] Jamil Abedalrahim Jamil Alsayaydeh, Win Adiyansyah Indra, Adam Wong Yoon Khang, Vadym Shkarupylo, Dhanigaletchmi A/P P Jkatisan. 2019. Development of Vehicle Ignition Using Fingerprint. ARPN Journal of Engineering and Applied Sciences. 14(23): 4045-4053.

[17] A. Moreno, M. S. Joy. 2006. Jeliot 3 in a Demanding Educational Setting, Fourth International Program Visualization Workshop, 29-30 June, Florence, Italy.

[18] S. Maravić Čisar, D. Radosav, R. Pinter, P. Čisar. 2011. Effectiveness of Program Visualization in Learning Java: a Case Study with Jeliot 3. International Journal of Computers, Communications & Control. VI(4): 668-680. DOI: 10.15837/ijccc.2011.4.2094.

[19] B. Miller, D. Ranum. 2013. Problem Solving with Algorithms and Data Structures. [https://www.cs.auckland.ac.nz/compsci105s1c/resources/ProblemSolvingwithAlgorithmsandDataStructures.pdf]

[20] Jamil Abedalrahim Jamil Alsayaydeh, Adam Wong Yoon Khang, Win Adiyansyah Indra, Vadym Shkarupylo and Jayananthinii Jayasundar. 2019. Development of smart dustbin by using apps. ARPN Journal of Engineering and Applied Sciences. 14(21): 3703-3711.

[21] N. Ishihara, N. Funabiki, M. Kuribayashi, and W.-C. Kao. 2011. A Software Architecture for Java Programming Learning Assistant System. International Journal of Computer & Software Engineering. 2(1): 116. DOI:10.15344/2456-4451/2017/116.

[22] H. H. S. Kyaw, S. T. Aung, H. A. Thant, and N. Funabiki. 2018. A proposal of code completion problem for Java programming learning assistant system. in Proc. VENOA 2018, pp. 855-864.

[23] Jamil Abedalrahim Jamil Alsayaydeh, Win Adiyansyah Indra, Adam Wong Yoon Khang, A K M Zakir Hossain, Vadym Shkarupylo and J. Pusppanathan. 2020. The experimental studies of the automatic control methods of magnetic separators performance by magnetic product. ARPN Journal of Engineering and Applied Sciences. 15(7): 922-927.