Table of Contents

Introduction

In computer science, a sorting algorithm is an algorithm that is able to sort the elements in a list to an order. There are many factors to consider when choosing the most suitable sorting algorithm. In this assignment, we will study sorting algorithm (radix sort, insertion sort, merge sort) and the concept of analysing algorithm through empirical method and compare the result with the result of theoretical analysis in the textbook. The empirical method is carried by an implementation of a program that counts the number of primitive operations performed by using different datasets. We are also required to write Java code based on the given radix sort pseudocode.

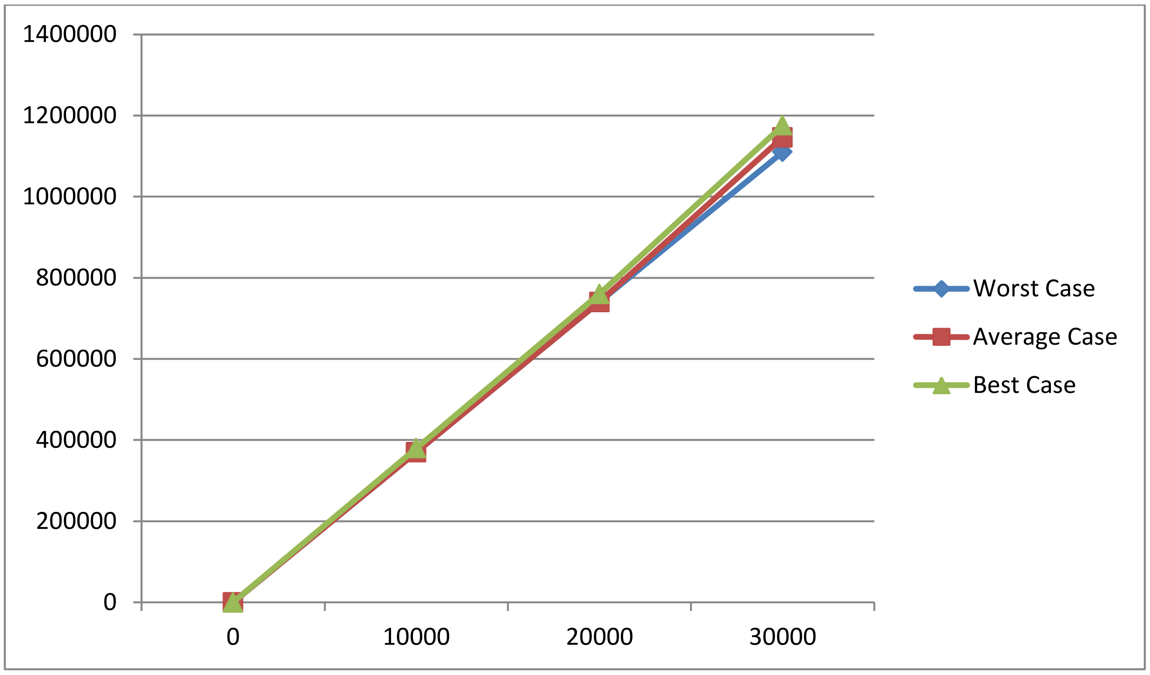
Radix Sort in Java

```
package radixsort;

import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.util.Random;
import java.io.PrintStream;
import java.io.FileOutputStream;

public class RadixSort {

    public static void main(String[] args) throws FileNotFoundException {

        sort();
        // TODO code application logic here
    }

    public static void sort() throws FileNotFoundException {

        long assignment = 0;
        long comparison = 0;
        long arithmetic = 0;

        Scanner scanner = new Scanner(System.in);

        System.out.print("How many numbers do you want to sort?: ");

        int totnum = scanner.nextInt();
        Random rand = new Random();
        Integer[] generated = new Integer[totnum];
        for (int i = 0; i < generated.length; i++) {
            generated[i] = rand.nextInt(1000000) + 1;
        }

        int temp = 0;
        System.out.println("Do you want to sort it before executing the
sorting algorithm?");
        System.out.println("1. Yes, in descending order");
        System.out.println("2. No");
        System.out.println("3. Yes, in ascending order");
        System.out.print("Choice: ");
        int choice = scanner.nextInt();
        switch (choice) {
            case 1:
                for (int i = 0; i < generated.length; i++) {
                    for (int j = i + 1; j < generated.length; j++) {
                        if (generated[i] < generated[j]) {
                            temp = generated[i];
                            generated[i] = generated[j];
                            generated[j] = temp;
                        }
                    }
                }
                break;

            case 2:
                break;
            case 3:
                for (int i = 0; i < generated.length; i++) {
                    for (int j = i + 1; j < generated.length; j++) {
                        if (generated[i] > generated[j]) {
                            temp = generated[i];
                            generated[i] = generated[j];
                            generated[j] = temp;
                        }
                    }
                }
                break;
        }

        List<Integer>[] array1 = new List[10]; //Create two arrays which
store list
        List<Integer>[] array2 = new List[10];
        int max = 0; //For finding the maximum number

        for (int i = 0; i < array1.length; i++) {
            List<Integer> list1 = new ArrayList<>(); //Initializing
ArrayList inside of each element in the array
            array1[i] = list1;
        }
        for (int i = 0; i < array2.length; i++) {
            List<Integer> list2 = new ArrayList<>(); //Initializing
ArrayList inside of each element in the array
            array2[i] = list2;
        }

        //Calculate the running time in millisecond
        long startTime = System.currentTimeMillis();
        long total = 0;
        for (int i = 0; i < 10000000; i++) {
            total += i;
        }

        //First sort iteration will start right after numbers are
inputted by users, maximum number will be found
        for (int i = 0; i < totnum; i++) {
            comparison++;
            int number = generated[i];
            int getmod = number % 10;
            assignment += 2;
            arithmetic++;
            array1[getmod].add(number);
            if (number > max) {
                max = number;
                assignment++;
            }
            comparison++;
            arithmetic++;
            assignment++;
        }

        //Setting another variables as pointers to those arrays
        List<Integer>[] source = array1;
        List<Integer>[] destination = array2;

        String strmax = Integer.toString(max);

        int modulo = 10; //For getting the last digit of the number based
on the sort iteration
        int divid = 10;
        assignment += 5;

        for (int i = 1; i <= strmax.length() - 1; i++) {
            comparison++; //comparison of for loop
            arithmetic++; //strmax.length()-1

            for (int j = 0; j < source.length; j++) {
                comparison++;
                while (!source[j].isEmpty()) {
                    comparison++;
                    int getnum = source[j].get(0);
                    assignment++;
                    int dividnum = getnum / divid;
                    assignment++;
                    arithmetic++;
                    int lastnum = dividnum % modulo; //Getting the digit
which is needed for comparison
                    assignment++;
                    arithmetic++;
                    source[j].remove(0); //Remove the number from source
then insert it to destination with proper place
                    destination[lastnum].add(getnum);
                }
                arithmetic++;//j++
                assignment++;//j++

            }

            divid *= 10;
            arithmetic++;
            assignment++;

            //Setting source and destination after each sort iteration
            if (i % 2 == 0) {
                source = array1;
                destination = array2;
            } else {
                source = array2;
                destination = array1;
            }
            assignment += 2;
            comparison++;
            arithmetic++; //i++
            assignment++; //i++
        }

        //Setting the destination after the sort has been completed
        if ((strmax.length() - 1) % 2 == 0) {
            source = array2;
            destination = array1;
        } else {
            source = array1;
            destination = array2;
        }
        comparison++;
        assignment += 2;
        arithmetic += 2;//minus and modulus

        //sort end
        long stopTime = System.currentTimeMillis();
        long elapsedTime = stopTime - startTime;

        for (int i = 0; i < destination.length; i++) {
            while (!destination[i].isEmpty()) {
                System.out.println(destination[i].get(0));
                destination[i].remove(0);
            }
        }

        System.out.println();
        System.out.println("The numbers of comparison is " + comparison);
        System.out.println("The numbers of assignment is " + assignment);
        System.out.println("The numbers of arithmetic is " + arithmetic);
        System.out.println(elapsedTime + " milliseconds");

        PrintStream out = new PrintStream(new
FileOutputStream("DataSet.txt"));
        for (int i = 0; i < generated.length; i++) {
            out.println(generated[i]);
        }
        out.close();

    }

}
```

Analysis and Discussion of Sorting Algorithm

The analysis starts by generating integer values ranging from 1 to 1 million of size 500, 1000, 5000, 10000, 20000 and 30000. For each set, it will run for 5 times and a graph for the best case, average case and worst case is plotted. Best case consists of integer that are already sorted in ascending order while worst case consist of integer that are already sorted in descending order. Moreover, average case consists of integers that are in random order. Then, the total primitive operations of comparison, assignment and arithmetic are calculated and tabulated. After that, a graph of the total of primitive operation is plotted and the result is being compared and discussed with the time complexity found on the textbook. The details of each run of respective sorting algorithm can be found in the appendices.

Radix Sort

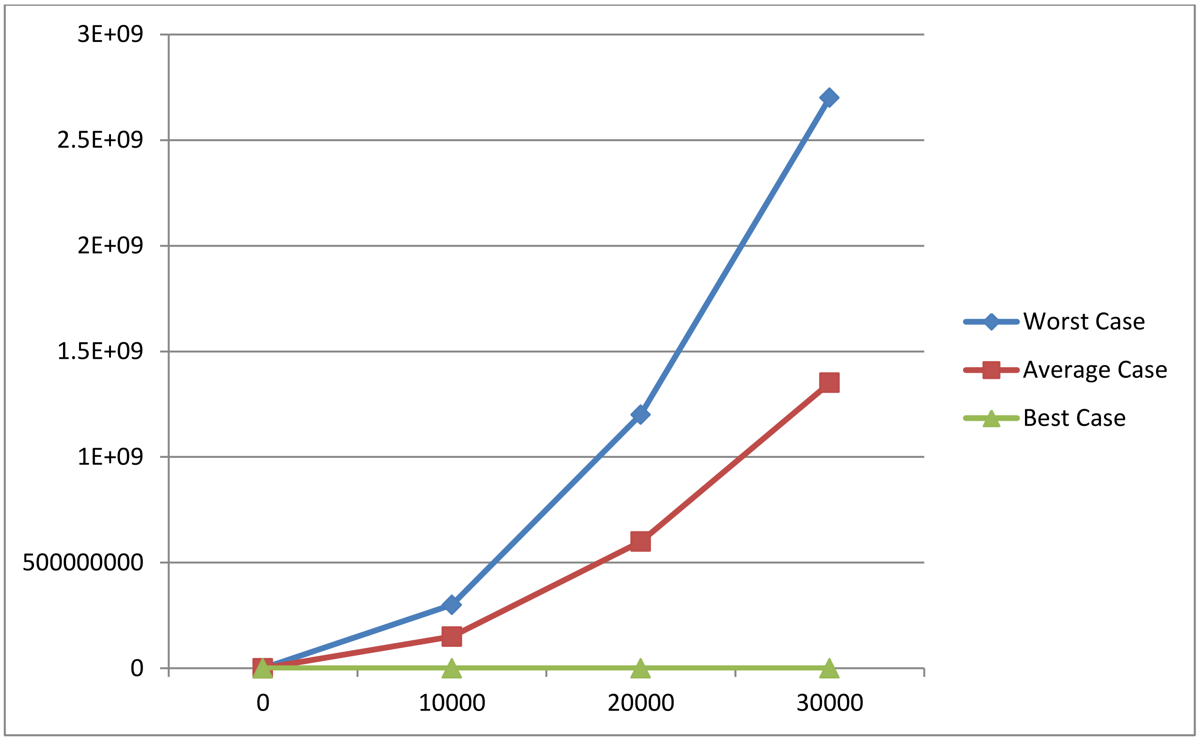


Type of Case / Size of Input	Worst Case	Average Case	Best Case
500	18706	18712.2	19205
1000	37206	37212.6	38204.8
5000	185206	191222	190192.2
10000	370206	370213.2	380152.6
20000	740206	740215.8	760009.6
30000	1110206	1146224	1175752

Based on: / Type of Case	Collected data and plotted graph	Textbook
Worst Case	O(nk)	O(nk)
Average Case	O(nk)	O(nk)
Best Case	O(nk)	O(nk)

Based on the collected data, we have plotted a graph which indicates a time complexity of O(nk). It also shows that it has time complexity of O(nk) which the running time increase at most linearly with the size of the input and dependent of the number k in all three cases. In addition, the time complexity of O(nk) based on the collected data and plotted graph is the same as the time complexity of radix sort algorithm in the textbook.

Insertion Sort

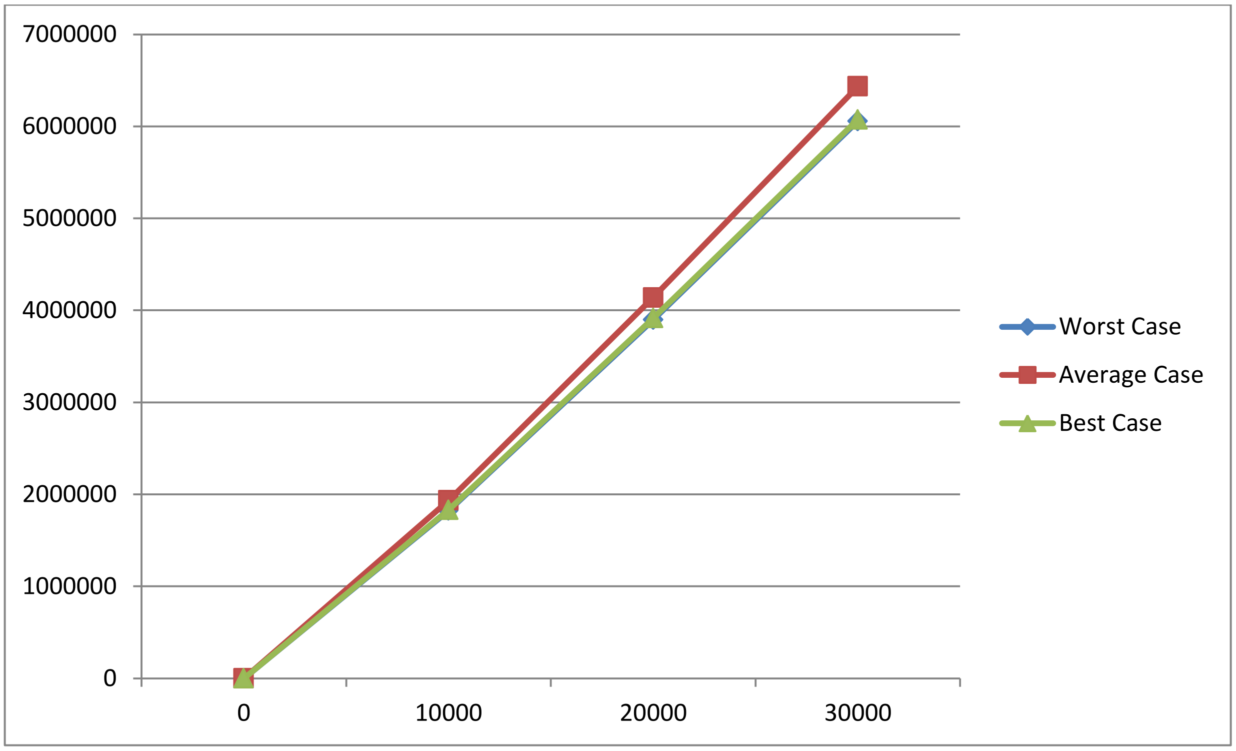


Type of Case / Size of Input	Worst Case	Average Case	Best Case
500	752491.6	373543.6	3994
1000	3004988	1506122	7994
5000	75024926.8	37332437.2	39994
10000	300049701.2	150713915.6	65594
20000	1200098838	599902711.6	159994
30000	2700147334	1351174666	196794

Based on: / Type of Case	Collected data and plotted graph	Textbook
Worst Case	$O(n^2)$	$O(n^2)$
Average Case	$O(n^2)$	$O(n^2)$
Best Case	O(n)	O(n)

Based on the collected data, we have plotted a graph which indicates a time complexity of $O(n^2)$ in both worst case and average case and O(n) in best case. It also shows that it has quadratic time complexity, $O(n^2)$ which the running time is proportional to the square of the size of the input in worst case and average case. Moreover, it has linear time complexity, $O(n)$ in best case which the time grows linearly as the size of input increase. In addition, the time complexity of $O(n^2)$ in worst case and average case and O(n) in best case based on the collected data and plotted graph is the same as the time complexity of insertion sort algorithm in the textbook.

Merge Sort



Type of Case / Size of Input	Worst Case	Average Case	Best Case
500	64002.4	67270.8	64114
1000	140518	148078	140742
5000	849297.2	900056.8	853686
10000	1823648.8	1935311.2	1832386
20000	3897404.4	4140666.8	3914789
30000	6058528.4	6436230.4	6074498

Based on: / Type of Case	Collected data and plotted graph	Textbook
Worst Case	O(n log n)	O(n log n)
Average Case	O(n log n)	O(n log n)
Best Case	O(n log n)	O(n log n)

Based on the collected data, we have plotted a graph which indicates a time complexity of O(n log n). It also shows that it has time complexity of O(n log n) which the running time increase at n times the logarithm of n where n is the size of input in all three cases. In addition, the time complexity of O(n log n) based on the collected data and plotted graph is the same as the time complexity of merge sort algorithm in the textbook.

Discussion of Runtime(ms) of Sorting Algorithm

Radix Sort

Type of Case / Size of Input	Worst Case	Average Case	Best Case
500	13.2	16	12.8
1000	18.6	18.8	19
5000	25.2	27.6	24.2
10000	38	34.6	35.6
20000	49.4	64.2	52.8
30000	**72**	**77**	**74.4**

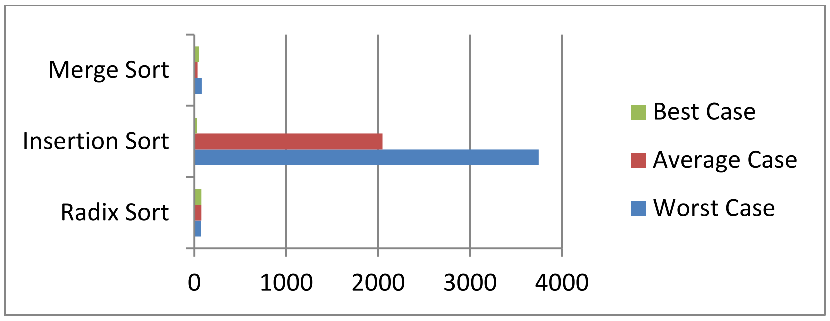
Insertion Sort

Type of Case / Size of Input	Worst Case	Average Case	Best Case
500	29.2	22.8	9
1000	79.6	49.6	10.6
5000	212.2	154.2	14
10000	613	306.2	17.4
20000	1744.8	952.8	25
30000	**3745.6**	**2047**	**31**

Merge Sort

Type of Case / Size of Input	Worst Case	Average Case	Best Case
500	9	11.8	9
1000	11	13.8	12
5000	16.8	18	19.2
10000	29.2	26.2	22.8
20000	53	28.2	32.4
30000	**78**	**34.8**	**53.2**

For this analysis, we will take the largest set of data which is 30000 as our object because the difference among the sorting algorithm in different cases is the largest.



From the chart, we can see that insertion sort perform very badly in average cases and worst cases because the number is in reverse sorted and the next element in the list is always greater than the previous element, and thus swapping have to occur all the time which affect the performance badly. However, insertion sort has the best performance in best case when there is no swapping occurs. Moreover, radix sort has average performance in all three cases which prove that it has linear complexity in any cases. In addition, merge sort is the best sorting algorithm based on the runtime because it performs well in all three cases. Nevertheless, in a sorted input, insertion sort can sort faster than merge sort but merge sort is much better than insertion sort in worst case and average case.

Conclusion

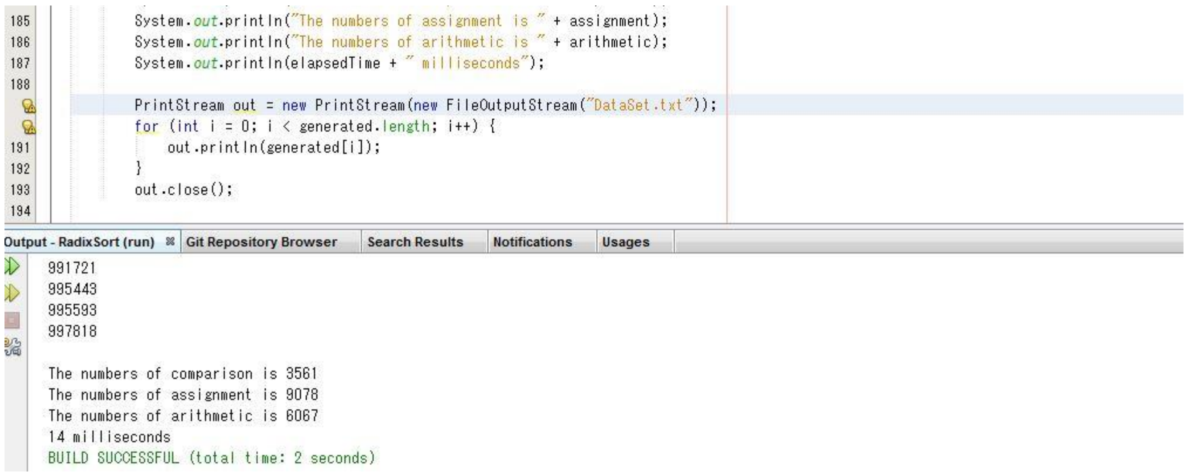
We have to consider factors such as the time complexity, the expected format and order of the input and the runtime of the sorting algorithm when choosing the most suitable sorting algorithm. Insertion sort doesn’t require additional memory and easy to implement and the time complexity can reach until O(n) which is fast if the input is neatly sorted. However, it has a time complexity of $O(n^2)$ and performs badly in both worst case and average case. Thus, it is not the most effective sorting algorithm among discussed sorting algorithm. Furthermore, Radix sort has average performance and a time complexity of O(nk) in all three cases but it requires additional memory and it can only handle integer. In our opinions, we think that merge sort is the best among discussed algorithm in term of overall performance because it has good performance and a time complexity of O(n log n) in all three cases and it can even handle floating point number. It is also easy to implement and still give good performance. However, it is slower than insertion sort when it comes to sorted input.

In conclusion, it is important that we consider factors such as the size of the input data and expected order and format of the input data so that we can determine the best sorting algorithm depending on the given input.

References

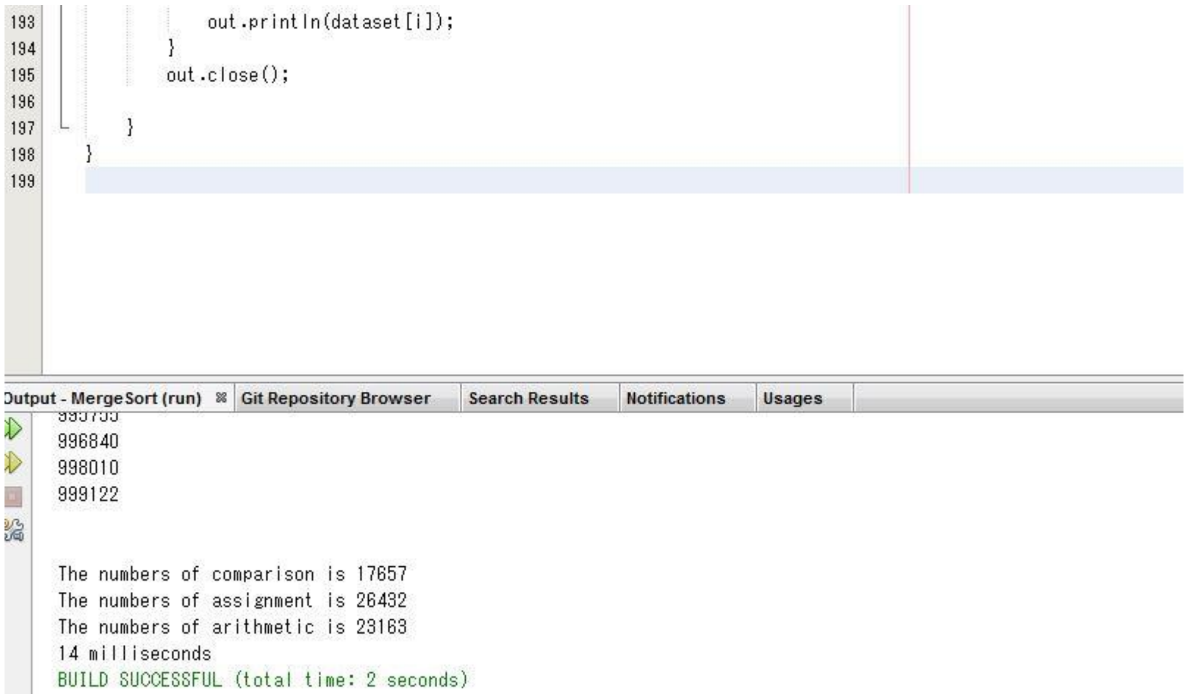
1. M. A. Weiss, Data structures and algorithm analysis in Java. Boston: Peason Addison-Wesley, 2007.
2. *Random (Java Platform SE 7)*, 19-Dec-2017. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/util/Random.html.
3. "Asymptotic Analysis and comparison of sorting algorithms," *GeeksforGeeks*, 07-Jan-2017. [Online]. Available: https://www.geeksforgeeks.org/asymptotic-analysis-comparison-sorting-algorithms/.
4. "Java Tutorial," *www.tutorialspoint.com*, 25-Mar-2018. [Online]. Available: https://www.tutorialspoint.com/java/index.htm.
5. "Measure execution time for a Java method," *Measure execution time for a Java method - Stack Overflow*. [Online]. Available: https://stackoverflow.com/questions/3382954/measure-execution-time-for-a-java-method.
6. "Quicksort, Insertion Sort and Radix Sort Comparison," *Can Güney Aksakalli*, 29-Nov-2011. [Online]. Available: https://aksakalli.github.io/2011/11/29/sorting-algorithms.html.
7. "Time Complexities of all Sorting Algorithms," *GeeksforGeeks*, 08-May-2017. [Online]. Available: https://www.geeksforgeeks.org/time-complexities-of-all-sorting-algorithms/.
8. "Know Thy Complexities!," *Big-O Algorithm Complexity Cheat Sheet (Know Thy Complexities!) @ericdrowell*. [Online]. Available: http://bigocheatsheet.com/.

Sample Output



```
        System.out.println("The numbers of assignment is " + assignment);
        System.out.println("The numbers of arithmetic is " + arithmetic);
        System.out.println(elapsedTime + " milliseconds");

        PrintStream out = new PrintStream(new FileOutputStream("DataSet.txt"));
        for (int i = 0; i < generated.length; i++) {
            out.println(generated[i]);
        }
        out.close();
```

```
991721
995443
995593
997818

The numbers of comparison is 3561
The numbers of assignment is 9078
The numbers of arithmetic is 6067
14 milliseconds
BUILD SUCCESSFUL (total time: 2 seconds)
```



```
                out.println(dataset[i]);
            }
            out.close();

        }
    }
```

```
996840
998010
999122


The numbers of comparison is 17657
The numbers of assignment is 26432
The numbers of arithmetic is 23163
14 milliseconds
BUILD SUCCESSFUL (total time: 2 seconds)
```

Appendices

Worst Case

Radix Sort

Size of input :500					
Run / Primitive Operation	1	2	3	4	5
Comparison	3561	3561	3561	3561	3561
Assignment	9078	9078	9078	9078	9078
Arithmetic	6067	6067	6067	6067	6067
Total	18706	18706	18706	18706	18706
Runtime (ms)	14	15	13	12	12
Size of input :1000					
Run / Primitive Operation	1	2	3	4	5
Comparison	7061	7061	7061	7061	7061
Assignment	18078	18078	18078	18078	18078
Arithmetic	12067	12067	12067	12067	12067
Total	37206	37206	37206	37206	37206
Runtime (ms)	18	16	16	20	23
Size of input :5000					
Run / Primitive Operation	1	2	3	4	5

Comparison	35061	35061	35061	35061	35061
Assignment	90078	90078	90078	90078	90078
Arithmetic	60067	60067	60067	60067	60067
Total	185206	185206	185206	185206	185206
Runtime (ms)	27	22	25	26	26
Size of input :10000					
Run Primitive Operation	1	2	3	4	5
Comparison	70061	70061	70061	70061	70061
Assignment	180078	180078	180078	180078	180078
Arithmetic	120067	120067	120067	120067	120067
Total	370206	370206	370206	370206	370206
Runtime (ms)	34	38	34	41	43
Size of input :20000					
Run Primitive Operation	1	2	3	4	5
Comparison	140061	140061	140061	140061	140061
Assignment	360078	360078	360078	360078	360078
Arithmetic	240067	240067	240067	240067	240067
Total	740206	740206	740206	740206	740206
Runtime (ms)	48	53	51	47	48
Size of input :30000					
Run Primitive Operation	1	2	3	4	5

Comparison	210061	210061	210061	210061	210061
Assignment	540078	540078	540078	540078	540078
Arithmetic	360067	360067	360067	360067	360067
Total	1110206	1110206	1110206	1110206	1110206
Runtime (ms)	71	76	65	76	72

Insertion Sort

Size of input :500					
Run / Primitive Operation	1	2	3	4	5
Comparison	249998	250000	249998	250000	250000
Assignment	251495	251497	251495	251497	251497
Arithmetic	250995	250997	250995	250997	250997
Total	752488	752494	752488	752494	752494
Runtime (ms)	28	30	33	25	30
Size of input :1000					
Run / Primitive Operation	1	2	3	4	5
Comparison	1000000	999998	999992	1000000	1000000
Assignment	1002997	1002995	1002989	1002997	1002997
Arithmetic	1001997	1001995	1001989	1001997	1001997
Total	3004994	3004988	3004970	3004994	3004994
Runtime (ms)	70	88	87	76	77
Size of input :5000					

Run / Primitive Operation	1	2	3	4	5
Comparison	24999980	24999982	24999978	24999968	24999980
Assignment	25014977	25014979	25014975	25014965	25014977
Arithmetic	25009977	25009979	25009975	25009965	25009977
Total	75024934	75024940	75024928	75024898	75024934
Runtime (ms)	188	227	203	227	216

Size of input :10000

Run / Primitive Operation	1	2	3	4	5
Comparison	99999930	99999888	9999908	99999892	999999894
Assignment	100029927	100029885	100029905	100029889	100029891
Arithmetic	100019927	100019885	100019905	100019889	100019891
Total	300049784	300049658	300049718	300049670	300049676
Runtime (ms)	598	624	597	614	632

Size of input :20000

Run / Primitive Operation	1	2	3	4	5
Comparison	399999614	399999620	399999664	399999596	399999580
Assignment	400059611	400059617	400059661	400059593	400059577
Arithmetic	400039611	400039617	400039661	400039593	400039577
Total	1200098836	1200098854	1200098986	1200098782	1200098734
Runtime (ms)	1756	1736	1762	1735	1735

Size of input :30000

Run / Primitive Operation	1	2	3	4	5
Comparison	899999122	899999082	899999092	899999134	899999136
Assignment	900089119	900089079	900089089	900089131	900089133
Arithmetic	900059119	900059079	900059089	900059131	900059133
Total	2700147360	2700147240	2700147270	2700147396	2700147402
Runtime (ms)	3800	3300	3776	3853	3999

Merge Sort

Size of input :500					
Run / Primitive Operation	1	2	3	4	5
Comparison	14409	14407	14407	14407	14407
Assignment	26432	26432	26432	26432	26432
Arithmetic	23163	23163	23163	23163	23163
Total	64004	64002	64002	64002	64002
Runtime (ms)	8	9	9	9	10

Size of input :1000					
Run / Primitive Operation	1	2	3	4	5
Comparison	31815	31815	31815	31815	31815
Assignment	57872	57872	57872	57872	57872
Arithmetic	50831	50831	50831	50831	50831
Total	140518	140518	140518	140518	140518
Runtime (ms)	11	11	11	11	11
Size of input :5000					
Run / Primitive Operation	1	2	3	4	5
Comparison	193237	193231	193237	293235	193231
Assignment	349032	349032	349032	349032	349032
Arithmetic	307031	307031	307031	307031	307031
Total	849300	849294	849300	849298	849294
Runtime (ms)	16	16	20	15	17
Size of input :10000					
Run / Primitive Operation	1	2	3	4	5
Comparison	416521	416515	416493	416513	416507
Assignment	748072	748072	748072	748072	748072
Arithmetic	659067	659067	659067	659067	659067
Total	1823660	1823654	1823632	1823652	1823646
Runtime (ms)	32	30	29	30	25

Size of input :20000					
Run / Primitive Operation	1	2	3	4	5
Comparison	893101	893123	893125	893101	893117
Assignment	1596152	1596152	1596152	1596152	1596152
Arithmetic	1408139	1408139	1408139	1408139	1408139
Total	3897392	3897414	3897416	3897392	3897408
Runtime (ms)	45	62	50	55	53
Size of input :30000					
Run / Primitive Operation	1	2	3	4	5
Comparison	1393917	1393921	1393953	1393985	1393971
Assignment	2476152	2476152	2476152	2476152	2476152
Arithmetic	2188427	2188427	2188427	2188427	2188427
Total	6058496	6058500	6058532	6058564	6058550
Runtime (ms)	75	78	85	74	78

Average Case

Radix Sort

Size of input :500					
Run / Primitive Operation	1	2	3	4	5
Comparison	3561	3561	3561	3561	3561

Assignment	9082	9093	9083	9081	9082
Arithmetic	6067	6067	6067	6067	6067
Total	18710	18721	18711	18709	18710
Runtime (ms)	14	17	14	17	18
Size of input :1000					
Run / Primitive Operation	1	2	3	4	5
Comparison	7061	7061	7061	7061	7061
Assignment	18086	18084	18085	18086	18082
Arithmetic	12067	12067	12067	12067	12067
Total	37214	37212	37213	37214	37210
Runtime (ms)	16	20	20	19	19
Size of input :5000					
Run / Primitive Operation	1	2	3	4	5
Comparison	35061	40073	35061	35061	35061
Assignment	90083	105104	90085	90082	90091
Arithmetic	60067	7080	60067	60067	60067
Total	185211	215257	185213	185210	185219
Runtime (ms)	27	25	27	30	29
Size of input :10000					
Run / Primitive Operation	1	2	3	4	5
Comparison	70061	70061	70061	70061	70061

Assignment	180090	180082	180081	180090	180083
Arithmetic	120067	120067	120067	120067	120067
Total	370218	370210	370209	370218	370211
Runtime (ms)	34	35	37	32	35
Size of input :20000					
Run / Primitive Operation	1	2	3	4	5
Comparison	140061	140061	140061	140061	140061
Assignment	360087	360088	360087	360089	360088
Arithmetic	240067	240067	240067	240067	240067
Total	740215	740216	740215	740217	740216
Runtime (ms)	70	70	54	58	69
Size of input :30000					
Run / Primitive Operation	1	2	3	4	5
Comparison	210061	210061	210061	210061	240073
Assignment	540090	540086	540089	540088	630100
Arithmetic	360067	360067	360067	360067	420080
Total	1110218	1110214	1110217	1110216	1290253
Runtime (ms)	85	82	69	76	73

Insertion Sort

Size of input :500

Run / Primitive Operation	1	2	3	4	5
Comparison	126938	125982	120554	122748	122194
Assignment	128435	127479	122051	124245	123691
Arithmetic	127935	126979	121551	123745	123191
Total	383308	380440	364156	370738	369076
Runtime (ms)	22	25	24	23	20

Size of input :1000

Run / Primitive Operation	1	2	3	4	5
Comparison	508026	492306	503090	500746	497712
Assignment	511023	495303	506087	503743	500709
Arithmetic	510023	494303	505087	502743	499709
Total	1529072	1481912	1514264	1507232	1498130
Runtime (ms)	50	48	49	52	49

Size of input :5000

Run / Primitive Operation	1	2	3	4	5
Comparison	12441136	12551834	12466396	12590984	12128722
Assignment	12456133	12566831	12481393	12605981	12143719
Arithmetic	12451133	12561831	12476393	12600961	12138719
Total	37348402	37680496	37424182	37797946	36411160

Runtime (ms)	139	155	192	148	137
Size of input :10000					
Run / Primitive Operation	1	2	3	4	5
Comparison	50349960	50320014	50259410	50042622	50134530
Assignment	50379957	50350011	50289407	50072619	50164527
Arithmetic	50369957	50340011	50279407	50062619	50154527
Total	151099874	151010036	150828224	150177860	150453584
Runtime (ms)	279	295	320	314	323
Size of input :20000					
Run / Primitive Operation	1	2	3	4	5
Comparison	199650974	199960312	200770018	200277690	199012202
Assignment	199710971	200020309	200830015	200337687	199072199
Arithmetic	199690971	200000309	200810015	200317687	199052199
Total	599052916	599980930	602410048	600933064	597136600
Runtime (ms)	922	971	972	966	933
Size of input :30000					
Run / Primitive Operation	1	2	3	4	5
Comparison	450470690	451095474	449548294	448605342	451987986
Assignment	450560687	451185471	449638291	448695339	452077983
Arithmetic	450530687	451155471	449608291	448665339	452047983
Total	1351562064	1353436416	1348794876	1345966020	1356113952

Runtime (ms)	2018	1971	2050	1991	2205

Merge Sort

Size of input :500					
Run / Primitive Operation	1	2	3	4	5
Comparison	17697	17693	17669	17691	17629
Assignment	26432	26432	26432	26432	26432
Arithmetic	23163	23163	23163	23163	23163
Total	67292	67288	67264	67286	67224
Runtime (ms)	13	11	12	11	12
Size of input :1000					
Run / Primitive Operation	1	2	3	4	5
Comparison	39389	39395	39373	39339	39379
Assignment	57872	57872	57872	57872	57872
Arithmetic	50831	50831	50831	50831	50831
Total	148092	148098	148076	148042	148082
Runtime (ms)	15	12	13	15	14

Size of input :5000					
Run / Primitive Operation	1	2	3	4	5
Comparison	244053	243915	243951	243933	244117
Assignment	349032	349032	349032	349032	349032
Arithmetic	307031	307031	307031	307031	307031
Total	900116	899978	900014	899996	900180
Runtime (ms)	20	18	20	17	15
Size of input :10000					
Run / Primitive Operation	1	2	3	4	5
Comparison	528083	528117	528329	528373	527959
Assignment	748072	748072	748072	748072	748072
Arithmetic	659067	659067	659067	659067	659067
Total	1935222	1935256	1935468	1935512	1935098
Runtime (ms)	26	22	34	22	27
Size of input :20000					
Run / Primitive Operation	1	2	3	4	5
Comparison	1136293	1136553	1136273	1136373	1136387
Assignment	1596152	1596152	1596152	1596152	1596152
Arithmetic	1408139	1408139	1408139	1408139	1408139
Total	4140584	4140844	4140564	4140664	4140678
Runtime (ms)	33	28	26	27	27

Size of input :30000					
Run / Primitive Operation	1	2	3	4	5
Comparison	1771769	1771453	1771881	1771753	1771401
Assignment	2476152	2476152	2476152	2476152	2476152
Arithmetic	2188427	2188427	2188427	2188427	2188427
Total	6436348	6436032	6436460	6436332	6435980
Runtime (ms)	36	31	36	35	36

Best Case

Radix Sort

Size of input :500					
Run / Primitive Operation	1	2	3	4	5
Comparison	3561	3561	3561	3561	3561
Assignment	9577	9577	9577	9577	9577
Arithmetic	6067	6067	6067	6067	6067
Total	19205	19205	19205	19205	19205
Runtime (ms)	12	12	12	16	12
Size of input :1000					
Run / Primitive Operation	1	2	3	4	5
Comparison	7061	7061	7061	7061	7061

Assignment	19077	19076	19077	19077	19077
Arithmetic	12067	12067	12067	12067	12067
Total	38205	38204	38205	38205	38205
Runtime (ms)	25	18	17	19	16
Size of input :5000					
Run / Primitive Operation	1	2	3	4	5
Comparison	35061	35061	35061	35061	35061
Assignment	95058	95069	95064	95064	95066
Arithmetic	60067	60067	60067	60067	60067
Total	190186	190197	190192	190192	190194
Runtime (ms)	30	24	21	22	24
Size of input :10000					
Run / Primitive Operation	1	2	3	4	5
Comparison	70061	70061	70061	70061	70061
Assignment	190032	190029	190016	190029	190017
Arithmetic	120067	120067	120067	120067	120067
Total	380160	380157	380144	380157	380145
Runtime (ms)	44	30	35	35	34
Size of input :20000					
Run / Primitive Operation	1	2	3	4	5
Comparison	140061	140061	140061	140061	140061

Assignment	379908	379898	379861	379864	379877
Arithmetic	240067	240067	240067	240067	240067
Total	760036	760026	759989	759992	760005
Runtime (ms)	56	50	56	45	57

Size of input :30000

Run / Primitive Operation	1	2	3	4	5
Comparison	210061	210061	240073	210061	210061
Assignment	569604	569631	659606	569648	569608
Arithmetic	360067	360067	420080	360067	360067
Total	1139732	1139759	1319759	1139776	1139736
Runtime (ms)	64	79	77	83	69

Insertion Sort

Size of input :500

Run / Primitive Operation	1	2	3	4	5
Comparison	500	500	500	500	500
Assignment	1997	1997	1997	1997	1997
Arithmetic	1497	1497	1497	1497	1497
Total	3994	3994	3994	3994	3994
Runtime (ms)	9	11	9	8	8

Size of input :1000

Run					

Primitive Operation	1	2	3	4	5
Comparison	1000	1000	1000	1000	1000
Assignment	3997	3997	3997	3997	3997
Arithmetic	2997	2997	2997	2997	2997
Total	7994	7994	7994	7994	7994
Runtime (ms)	11	7	11	13	11
Size of input :5000					
Run / Primitive Operation	1	2	3	4	5
Comparison	5000	50000	5000	5000	5000
Assignment	19997	19997	19997	19997	19997
Arithmetic	14997	14997	14997	14997	14997
Total	39994	39994	39994	39994	39994
Runtime (ms)	13	14	14	16	13
Size of input :10000					
Run / Primitive Operation	1	2	3	4	5
Comparison	10000	10000	10000	10000	10000
Assignment	39997	39997	39997	39997	39997
Arithmetic	29997	29997	29997	29997	29997
Total	79994	79994	7994	79994	79994
Runtime (ms)	17	19	17	18	16
Size of input :20000					

Run / Primitive Operation	1	2	3	4	5
Comparison	20000	20000	20000	20000	20000
Assignment	79997	79997	79997	79997	79997
Arithmetic	59997	59997	59997	59997	59997
Total	159994	159994	159994	159994	159994
Runtime (ms)	25	25	27	26	22

Size of input :30000

Run / Primitive Operation	1	2	3	4	5
Comparison	30000	30000	30000	30000	30000
Assignment	119997	119997	119997	119997	119997
Arithmetic	89997	89997	89997	89997	89997
Total	239994	239994	23994	239994	239994
Runtime (ms)	30	32	31	31	31

Merge Sort

Size of input :500

Run / Primitive Operation	1	2	3	4	5
Comparison	14519	14519	14519	14519	14519

Assignment	26432	26432	26432	26432	26432
Arithmetic	23163	23163	23163	23163	23163
Total	64114	64114	64114	64114	64114
Runtime (ms)	9	8	8	9	11
<u>Size of input :1000</u>					
Run Primitive Operation	1	2	3	4	5
Comparison	32039	32039	32039	32039	32039
Assignment	57872	57872	57872	57872	57872
Arithmetic	50831	50831	50831	50831	50831
Total	140742	140742	140742	140742	140742
Runtime (ms)	11	14	12	10	13
<u>Size of input :5000</u>					
Run Primitive Operation	1	2	3	4	5
Comparison	197623	197623	197623	197623	197623
Assignment	349032	349032	349032	349032	349032
Arithmetic	307031	307031	307031	307031	307031
Total	853686	853686	853686	853686	853686
Runtime (ms)	19	20	22	16	19
<u>Size of input :10000</u>					
Run Primitive Operation	1	2	3	4	5
Comparison	425247	425247	425247	425247	425247

Assignment	748072	748072	748072	748072	748072
Arithmetic	659067	659067	659067	659067	659067
Total	1832386	1832386	1832386	1832386	1832386
Runtime (ms)	25	24	22	21	22
Size of input :20000					
Run / Primitive Operation	1	2	3	4	5
Comparison	910495	910495	910495	910495	910495
Assignment	1596152	1596152	1596152	1596152	1596152
Arithmetic	1408139	1408139	1408139	1408139	1408139
Total	3914786	3914786	3914786	3914786	3914786
Runtime (ms)	35	32	30	31	34
Size of input :30000					
Run / Primitive Operation	1	2	3	4	5
Comparison	1409919	1409919	1409919	1409919	1409919
Assignment	2476152	2476152	2476152	2476152	2476152
Arithmetic	2188427	2188427	2188427	2188427	2188427
Total	6074498	6074498	6074498	6074498	6074498
Runtime (ms)	50	52	57	56	51