App.java

```
package app;

import java.time.Duration;
import java.time.Instant;
import java.util.Arrays;

/**
 *
 * <p><strong><em>Application Name: </em></strong>Lab 5 Sorts</p>
 * <p><strong><em>Class Name: </em></strong>App</p>
 *
 * <p><strong><em>Application Notes: </em></strong>none</p>
 *
 * <p><strong><em>Class Notes: </em></strong>none</p>
 *
 * <p><strong><em>Pre-Conditions: </em></strong>none</p>
 *
 * <p><strong><em>Post-Conditions: </em></strong>none</p>
 *
 * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
 * <p><strong><em>Instructor: </em></strong>Dr. Robert Walsh</p>
 * <p><strong><em>Course: </em></strong>SP20-SE-CSCI-C202-17057</p>
 * <p><strong><em>Due Date: </em></strong>03.03.2020</p>
 *
 */
public class App {

    // class constants
    public static int SIZE = 100000;

    /**
     *
     * <p><strong><em>Description: </em></strong>application entry point</p>
     *
     * <p><strong><em>Method Name: </em></strong>main</p>
     *
     * <p><strong><em>Method Notes: </em></strong>none</p>
     *
     * <p><strong><em>Pre-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Post-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
     * <p><strong><em>Start Date: </em></strong>03.03.2020</p>
     *
```

```
     * @param args not used
     * @throws Exception error trapping
     */
    public static void main(String[] args) throws Exception {

        // variables

        int [] _list = new int [SIZE];
        Instant _sTime = null;
        Instant _eTime = null;
        Duration _tElapsed = null;

        System.out.println("==============Bubble Sort============\n");
        _sTime = Instant.now();
        System.out.println("\tSTART TIME: " + _sTime);

        System.out.println("\n********** Original Array");
        makeArray(_list);
        showArray(_list);
        System.out.println("Array Size: "+ SIZE);
        System.out.println("********** Sorted Array");
        bubbleSort(_list);
        showArray(_list);

        _eTime = Instant.now();
        _tElapsed = Duration.between(_sTime, _eTime);
        System.out.println("\n\n\tEND TIME: " + _eTime);
        System.out.println("\tTime for completion (milliseconds): " + _tElapsed.toMillis());
        System.out.println("============== Bubble Sort ============\n");


        System.out.println("============== Insertion Sort ============\n");
        _sTime = Instant.now();
        System.out.println("\tSTART TIME: " + _sTime);
        System.out.println("\n********** Original Array");
        makeArray(_list);
        showArray(_list);
        System.out.println("Array Size: "+ SIZE);
        System.out.println("********** Sorted Array");
        insertionSort(_list);
        showArray(_list);

        _eTime = Instant.now();
        _tElapsed = Duration.between(_sTime, _eTime);
        System.out.println("\n\n\tEND TIME: " + _eTime);
        System.out.println("\tTime for completion (milliseconds): " + _tElapsed.toMillis());
        System.out.println("============== Insertion Sort ============\n");


        System.out.println("============== Merge Sort ============\n");
```

```
_sTime = Instant.now();
System.out.println("\tSTART TIME: " + _sTime);
System.out.println("\n********** Original Array");
makeArray(_list);
showArray(_list);
System.out.println("Array Size: "+ SIZE);
System.out.println("********** Sorted Array");
mergeSort(_list);
showArray(_list);

_eTime = Instant.now();
_tElapsed = Duration.between(_sTime, _eTime);
System.out.println("\n\n\tEND TIME: " + _eTime);
System.out.println("\tTime for completion (milliseconds): " + _tElapsed.toMillis());
System.out.println("=============== Merge Sort =============\n");


System.out.println("=============== Quick Sort =============\n");
_sTime = Instant.now();
System.out.println("\tSTART TIME: " + _sTime);
System.out.println("\n********** Original Array");
makeArray(_list);
showArray(_list);
System.out.println("Array Size: "+ SIZE);
System.out.println("********** Sorted Array");
quickSort(_list);
showArray(_list);

_eTime = Instant.now();
_tElapsed = Duration.between(_sTime, _eTime);
System.out.println("\n\n\tEND TIME: " + _eTime);
System.out.println("\tTime for completion (milliseconds): " + _tElapsed.toMillis());
System.out.println("=============== Quick Sort =============\n");


System.out.println("=============== Selection Sort =============\n");
_sTime = Instant.now();
System.out.println("\tSTART TIME: " + _sTime);
System.out.println("\n********** Original Array");
makeArray(_list);
showArray(_list);
System.out.println("Array Size: "+ SIZE);
System.out.println("********** Sorted Array");
selectionSort(_list);
showArray(_list);

_eTime = Instant.now();
_tElapsed = Duration.between(_sTime, _eTime);
System.out.println("\n\n\tEND TIME: " + _eTime);
System.out.println("\tTime for completion (milliseconds): " + _tElapsed.toMillis());
```

```
        System.out.println("=============== Selection Sort =============\n");

        System.out.println("=============== System Sort =============\n");
        _sTime = Instant.now();
        System.out.println("\tSTART TIME: " + _sTime);

        System.out.println("\n********** Original Array");
        makeArray(_list);
        showArray(_list);
        System.out.println("Array Size: "+ SIZE);
        System.out.println("********** Sorted Array");
        systemSort(_list);
        showArray(_list);

        _eTime = Instant.now();
        _tElapsed = Duration.between(_sTime, _eTime);
        System.out.println("\n\n\tEND TIME: " + _eTime);
        System.out.println("\tTime for completion (milliseconds): " + _tElapsed.toMillis());
        System.out.println("=============== System Sort =============\n");
    }

    /**
     *
     * <p><strong><em>Description: </em></strong>creates an array based on constant SIZE</p>
     *
     * <p><strong><em>Method Name: </em></strong>makeArray</p>
     *
     * <p><strong><em>Method Notes: </em></strong>none</p>
     *
     * <p><strong><em>Pre-Conditions: </em></strong>array needed for population</p>
     *
     * <p><strong><em>Post-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
     * <p><strong><em>Start Date: </em></strong>03.05.2020</p>
     *
     * @param array array to process
     */
    public static void makeArray(int[] array) {

        for (int i = 0; i < array.length; i++){ array[i] = (int) (Math.random() * SIZE); } // end For

    } // end makeArray

    /**
     *
     * <p><strong><em>Description: </em></strong>display contents of array</p>
     *
     * <p><strong><em>Method Name: </em></strong>showArray</p>
     *
```

```
     * <p><strong><em>Method Notes: </em></strong>none</p>
     *
     * <p><strong><em>Pre-Conditions: </em></strong>some array</p>
     *
     * <p><strong><em>Post-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
     * <p><strong><em>Start Date: </em></strong>03.05.2020</p>
     *
     * @param array array to process
     */
    public static void showArray(int [] array){
        for (int _lC = 0; _lC < 10; _lC++){ System.out.print(" " + array[_lC]); } // end _lC

        System.out.print("....");

        for(int _lC = array.length - 10; _lC < array.length; _lC++){ System.out.print(" " + array[_lC]); } // end _lC

    } // end showArray

    /**
     *
     * <p><strong><em>Description: </em></strong>uses the bubble sort algorithm to sort an array</p>
     *
     * <p><strong><em>Method Name: </em></strong>bubbleSort</p>
     *
     * <p><strong><em>Method Notes: </em></strong>none</p>
     *
     * <p><strong><em>Pre-Conditions: </em></strong>an array</p>
     *
     * <p><strong><em>Post-Conditions: </em></strong>sorted array</p>
     *
     * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
     * <p><strong><em>Start Date: </em></strong>03.05.2020</p>
     *
     * @param list array to process
     */
    public static void bubbleSort(int[] list) {

        // variables
        boolean needNextPass = true;

        // loop outter
        for (int _k = 1; _k < list.length && needNextPass; _k++) {

            // Array may be sorted and next pass not needed
            needNextPass = false;

            for (int _i = 0; _i < list.length - _k; _i++) {
```

```
            if (list[_i] > list[_i + 1]) {

                // Swap list[i] with list[i + 1]
                int temp = list[_i];
                list[_i] = list[_i + 1];
                list[_i + 1] = temp;
                needNextPass = true; // Next pass still needed

            } // end if

        } // end _i

    } // end _k

} // end bubblesort

/**
 *
 * <p><strong><em>Description: </em></strong>process array using insertion sort algorithm</p>
 *
 * <p><strong><em>Method Name: </em></strong>insertionSort</p>
 *
 * <p><strong><em>Method Notes: </em></strong>none</p>
 *
 * <p><strong><em>Pre-Conditions: </em></strong>none</p>
 *
 * <p><strong><em>Post-Conditions: </em></strong>none</p>
 *
 * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
 * <p><strong><em>Start Date: </em></strong>03.05.2020</p>
 *
 * @param list array to process
 */
public static void insertionSort(int[] list) {

    // loop the array
    for (int _i = 1; _i < list.length; _i++) {

        int _currentElement = list[_i];
        int _k;

        for (_k = _i - 1; _k >= 0 && list[_k] > _currentElement; _k--) { list[_k + 1] = list[_k]; } // en d_k

        list[_k + 1] = _currentElement;

    } // end _i

}   // end insertionSort

/**
```

```
     *
     * <p><strong><em>Description: </em></strong>merge sort recursion</p>
     *
     * <p><strong><em>Method Name: </em></strong>mergeSort</p>
     *
     * <p><strong><em>Method Notes: </em></strong>none</p>
     *
     * <p><strong><em>Pre-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Post-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
     * <p><strong><em>Start Date: </em></strong>03.05.2020</p>
     *
     * @param list list to process
     */
    public static void mergeSort(int[] list) {

        if (list.length > 1) {

            // Merge sort the first half
            int[] _firstHalf = new int[list.length / 2];

            System.arraycopy(list, 0, _firstHalf, 0, list.length / 2);

            mergeSort(_firstHalf);

            // Merge sort the second half
            int _secondHalfLength = list.length - list.length / 2;
            int[] _secondHalf = new int[_secondHalfLength];

            System.arraycopy(list, list.length / 2, _secondHalf, 0, _secondHalfLength);

            mergeSort(_secondHalf);

            // Merge firstHalf with secondHalf into list
            merge(_firstHalf, _secondHalf, list);

        } // end if

    } // end mergeSort

    /**
     *
     * <p><strong><em>Description: </em></strong>Description</p>
     *
     * <p><strong><em>Method Name: </em></strong>merge</p>
     *
     * <p><strong><em>Method Notes: </em></strong>none</p>
     *
```

```
     * <p><strong><em>Pre-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Post-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
     * <p><strong><em>Start Date: </em></strong>03.05.2020</p>
     *
     * @param list1 list to process
     * @param list2 list to process
     * @param temp temp
     */
    public static void merge(int[] list1, int[] list2, int[] temp) {

        // varialbes
        int _current1 = 0; // Current index in list1
        int _current2 = 0; // Current index in list2
        int _current3 = 0; // Current index in temp

        while (_current1 < list1.length && _current2 < list2.length) {
            if (list1[_current1] < list2[_current2]) { temp[_current3++] = list1[_current1++]; } // end if
            else { temp[_current3++] = list2[_current2++]; } // end else
        } // end while

        while (_current1 < list1.length) { temp[_current3++] = list1[_current1++]; } // end while

        while (_current2 < list2.length) { temp[_current3++] = list2[_current2++]; } // end while
    }//Merge

    /**
     *
     * <p><strong><em>Description: </em></strong>Description</p>
     *
     * <p><strong><em>Method Name: </em></strong>quickSort</p>
     *
     * <p><strong><em>Method Notes: </em></strong>none</p>
     *
     * <p><strong><em>Pre-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Post-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
     * <p><strong><em>Start Date: </em></strong>03.05.2020</p>
     *
     * @param list list to process
     */
    public static void quickSort(int[] list) { quickSort(list, 0, list.length - 1); } // end quickSort helper

    /**
     *
     * <p><strong><em>Description: </em></strong>Description</p>
```

```
     *
     * <p><strong><em>Method Name: </em></strong>quickSort</p>
     *
     * <p><strong><em>Method Notes: </em></strong>none</p>
     *
     * <p><strong><em>Pre-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Post-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
     * <p><strong><em>Start Date: </em></strong>03.05.2020</p>
     *
     * @param list list to process
     * @param first first
     * @param last last
     */
    private static void quickSort(int[] list, int first, int last) {

        if (last > first) {

            int _pivotIndex = partition(list, first, last);

            quickSort(list, first, _pivotIndex - 1);

            quickSort(list, _pivotIndex + 1, last);

        } // end if

    } // end quickSort

    /**
     *
     * <p><strong><em>Description: </em></strong>Description</p>
     *
     * <p><strong><em>Method Name: </em></strong>partition</p>
     *
     * <p><strong><em>Method Notes: </em></strong>none</p>
     *
     * <p><strong><em>Pre-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Post-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
     * <p><strong><em>Start Date: </em></strong>03.05.2020</p>
     *
     * @param list list to process
     * @param first first
     * @param last last
     * @return
     */
```

```
    private static int partition(int[] list, int first, int last) {

        int _pivot = list[first]; // Choose the first element as the pivot
        int _low = first + 1; // Index for forward search
        int _high = last; // Index for backward search

        while (_high > _low) {
            // Search forward from left
            while (_low <= _high && list[_low] <= _pivot) { _low++; } // end while

            // Search backward from right
            while (_low <= _high && list[_high] > _pivot) { _high--; } // end while

            // Swap two elements in the list
            if (_high > _low) {

                int temp = list[_high];

                list[_high] = list[_low];
                list[_low] = temp;

            } // enif

        } // end while

        while (_high > first && list[_high] >= _pivot) { _high--; } // end while

        // Swap pivot with list[high]
        if (_pivot > list[_high]) {

            list[first] = list[_high];
            list[_high] = _pivot;

            return _high;

        } // endif

        else { return first; } // end else

    } // end partition

    /**
     *
     * <p><strong><em>Description: </em></strong>Description</p>
     *
     * <p><strong><em>Method Name: </em></strong>selectionSort</p>
     *
     * <p><strong><em>Method Notes: </em></strong>none</p>
     *
     * <p><strong><em>Pre-Conditions: </em></strong>none</p>
```

```
     *
     * <p><strong><em>Post-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
     * <p><strong><em>Start Date: </em></strong>03.05.2020</p>
     *
     * @param list list to process
     */
    public static void selectionSort(int[] list) {

        for (int _i = 0; _i < list.length - 1; _i++) {

            // Find the minimum in the list[i..list.length-1]
            int _currentMin = list[_i];
            int _currentMinIndex = _i;

            for (int _j = _i + 1; _j < list.length; _j++) {

                if (_currentMin > list[_j]) {
                _currentMin = list[_j];
                _currentMinIndex = _j;

                } // end if

            } // end for

            // Swap list[i] with list[currentMinIndex] if necessary;
            if (_currentMinIndex != _i) {

                list[_currentMinIndex] = list[_i];
                list[_i] = _currentMin;

            } // end if

        } // end for

    } // selectionSort

    /**
     *
     * <p><strong><em>Description: </em></strong>Description</p>
     *
     * <p><strong><em>Method Name: </em></strong>systemSort</p>
     *
     * <p><strong><em>Method Notes: </em></strong>none</p>
     *
     * <p><strong><em>Pre-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Post-Conditions: </em></strong>none</p>
     *
```

```
     * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
     * <p><strong><em>Start Date: </em></strong>03.05.2020</p>
     *
     * @param list list to process
     */
    public static void systemSort(int [] list){ Arrays.sort(list); } // end systemSort
}
```

Console Output

==========Bubble Sort==========

START TIME: 2020-03-05T13:09:06.272Z

********** Original Array
57746 250 20579 57951 59628 34263 88459 57500 70451 80328.... 37875 6368 983 33916 58493 22164 55014 51290 85491 34297Array Size: 100000
********** Sorted Array
0 1 3 3 3 7 9 10 14 15.... 99987 99989 99989 99989 99993 99995 99996 99997 99997 99999

END TIME: 2020-03-05T13:09:20.295Z
Time for completion (milliseconds): 14023
========== Bubble Sort ==========

========== Insertion Sort ==========

START TIME: 2020-03-05T13:09:20.295Z

********** Original Array
39919 41676 93075 22167 52120 83895 78640 88476 15346 799.... 67918 19680 76177 31572 11831 40296 51888 24705 3586 88437Array Size: 100000
********** Sorted Array
0 0 0 2 2 4 4 4 5 5.... 99988 99990 99991 99991 99995 99996 99997 99997 99997 99999

END TIME: 2020-03-05T13:09:22.864Z
Time for completion (milliseconds): 2569
========== Insertion Sort ==========

========== Merge Sort ==========

START TIME: 2020-03-05T13:09:22.864Z

********** Original Array
97643 81600 79381 77923 97177 68088 56743 28089 9040 54316.... 4371 30281 61208 20668 22679 38954 11572 79206 75953 33273Array Size: 100000
********** Sorted Array
3 5 6 6 7 7 8 9 10 10.... 99993 99993 99993 99995 99995 99996 99996 99997 99997 99998

END TIME: 2020-03-05T13:09:22.889Z
Time for completion (milliseconds): 25
========== Merge Sort ==========

========== Quick Sort ==========

START TIME: 2020-03-05T13:09:22.891Z

********** Original Array
78762 20045 14012 46837 24716 11592 57049 87603 95172 32957.... 47878 2640 87772 63517 26623 261 11355 88867 81110 64668Array Size: 100000
********** Sorted Array
0 1 1 1 2 4 7 7 8 8.... 99988 99989 99990 99991 99992 99993 99995 99995 99997 99997

END TIME: 2020-03-05T13:09:22.927Z
Time for completion (milliseconds): 36
========== Quick Sort ==========

========== Selection Sort ==========

START TIME: 2020-03-05T13:09:22.929Z

********** Original Array
83806 65033 48915 23129 56316 20999 15767 3050 17161 44243.... 94806 99866 69423 81286 3907 24720 14208 69048 29063 95123Array Size: 100000
********** Sorted Array
1 2 2 2 2 4 4 4 5 7.... 99991 99991 99992 99993 99993 99994 99995 99997 99998 99999

END TIME: 2020-03-05T13:09:24.706Z
Time for completion (milliseconds): 1777
========== Selection Sort ==========

========== System Sort ==========

START TIME: 2020-03-05T13:09:24.709Z

********** Original Array
46946 83075 4751 6927 51802 25851 37918 9151 30412 42882.... 66637 20141 76284 48232 24456 8266 9117 61263 21950 71448Array Size: 100000
********** Sorted Array
1 2 2 3 3 4 4 6 10 11.... 99986 99987 99990 99991 99993 99993 99995 99997 99999 99999

END TIME: 2020-03-05T13:09:24.755Z
Time for completion (milliseconds): 46
========== System Sort ==========