```java
/*
    CPSC 319 Assignment 1
    @author John Ezekiel Juliano
    @version 1.0
    @since  February 2, 2018
*/
import java.util.Random;
import java.util.Arrays;
import java.util.ArrayList;
import java.util.Collections;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;
public class ComplexityAnalysis{

    int[] data;
    int[] tempArr;
    int[] randArray;
    int size;
    String order;
    String sortingAlgorithm;
    String fileName;
    long start;
    long stop;
    long totalTime;
    /*
        This program is borrowed from the implementation discussed in lectures
    */
    public void selectionSort(){
        int i, j, min;
        for(i=0; i<data.length-1; i++){
            for(j=i+1, min=i; j<data.length; j++){
                if(data[j] < data[min])
                    min = j;
            }
            int temp = data[min];
            data[min] = data[i];
            data[i] = temp;
        }
    }
    /*
        This program is borrowed from the implementation discussed in lectures
    */
    public void insertionSort(){
        for(int i=1,j; i<data.length; i++){
            int temp = data[i];
            for(j=i; (j>0)&&(temp<data[j-1]); j--){
                data[j] = data[j-1];
            }
            data[j] = temp;
        }
    }
    /*
        This program is borrowed from the implementation discussed in lectures
    */
    public void mergeSort(int first, int last){
        if(first<last){
            int mid = (first+last)/2;
            mergeSort(first, mid);
            mergeSort(mid+1, last);
            merge(first, mid, last);
        }
    }
    public void merge(int first, int mid, int last){
        // copy both parts into tempArr
        for(int i=first; i<=last; i++){
```

```
            tempArr[i] = data[i];
        }
        int leftSubIndex = first, rightSubIndex = mid+1, dataIndex = first;
        // copy smallest values by comparing left and right subarray elements
        // one by one in to the original array
        while(leftSubIndex<=mid && rightSubIndex<=last){
            if(tempArr[leftSubIndex] <= tempArr[rightSubIndex]){
                data[dataIndex] = tempArr[leftSubIndex];
                leftSubIndex++;
            }else{
                data[dataIndex] = tempArr[rightSubIndex];
                rightSubIndex++;
            }
            dataIndex++;
        }
        // copy the rest of the left subarray since we are sorting misplaced
        // items from the right subarray
        while(leftSubIndex<=mid){
            data[dataIndex] = tempArr[leftSubIndex];
            dataIndex++;
            leftSubIndex++;
        }
    }
    /*
        This program is borrowed from the implementation discussed in lectures
    */
    public void quickSort(int lo, int hi){
        int first = lo, last = hi, temp;
        // set a pivot element
        int pivot = data[(lo+hi)/2];
        // divide arrays
        while(first<=last){
            // identify a number greater than pivot value from left subarray
            while(data[first]<pivot){
                first++;
            }
            // identify a number less than pivot value from right subarray
            while(data[last]>pivot){
                last--;
            }
            // swap
            if(first<=last){
                temp = data[first];
                data[first] = data[last];
                data[last] = temp;
                first++;
                last--;
            }
        }
        // recursive method call
        if(lo < last)
            quickSort(lo, last);
        if(first < hi)
            quickSort(first, hi);
    }
    /*
        manage all sorting algorithms
    */
    public void sorter(int numOfItems, String algorithm, int[] arr){
        // setup the array to be sorted
        this.data = arr;
        this.tempArr = new int[size];
        // switch statements
        switch(algorithm){
            case "selection":
                // measure the duration of the algorithm
```

```
                start = System.nanoTime();
                selectionSort();
                stop = System.nanoTime();
                // print out results
                printToFile();
                break;

            case "insertion":
                // measure the duration of the algorithm
                start = System.nanoTime();
                insertionSort();
                stop = System.nanoTime();
                // print out results
                printToFile();
                break;

            case "merge":
                // measure the duration of the algorithm
                start = System.nanoTime();
                mergeSort(0, size-1);
                stop = System.nanoTime();
                // print out results
                printToFile();
                break;

            case "quick":
                // measure the duration of the algorithm
                start = System.nanoTime();
                quickSort(0, size-1);
                stop = System.nanoTime();
                totalTime = stop - start;
                // print out results
                printToFile();
                break;
            // if invalid algorithm argument
            default:
                System.out.println("Invalid sorting algorithm. Program will now terminate.");
                System.exit(1);
        }
        System.out.print("\nThe "+ sortingAlgorithm +"sort algorithm took " + totalTime + "
        nanoseconds to sort an array of " + size + " items.\n");
    }
    /*
        This method will be used to prin program output to a file
    */
    public void printToFile(){
        PrintWriter cursor = null;
        try{
            cursor = new PrintWriter(fileName);
            cursor.print("This is a sample run with an initial "+order+" order of "+size+"
            items sorted using the "+sortingAlgorithm+"Sort algorithm.\n");
            for(int index = 0; index < data.length; index++){
                if(((index%10)==0) && (index!=0)){
                    cursor.print("\n");
                }
                else{
                    cursor.print(data[index] + " ");
                }
            }
            totalTime = stop - start;
            cursor.print("\nThe "+ sortingAlgorithm +"sort algorithm took " + totalTime + "
            nanoseconds to sort an array of " + size + " items.\n");
        }
        catch(Exception e){
            e.printStackTrace();
            System.out.println("File does not exist.");
```

```
        }
        finally{
            if(cursor != null){
                System.out.println("Closing PrintWriter.");
                cursor.close();
            }
            else{
                System.out.println("PrintWriter is not open!");
            }
        }
    }
    public static void main(String[] args){
        ComplexityAnalysis test = new ComplexityAnalysis();
        if(args.length == 4){
            test.order = args[0];
            test.size = Integer.parseInt(args[1]);
            test.sortingAlgorithm = args[2];
            test.fileName = args[3];
            test.randArray = new int[test.size];
            ArrayList<Integer> randomList = new ArrayList<Integer>(test.size);
            // check if valid size argument
            if(test.size < 0){
                System.out.println("Invalid array size. Must be at least 1. Program will now
                exit.");
                System.exit(1);
            }
            else{
                Random gen = new Random();
                for(int i=0; i<test.size; i++){
                    Integer num = gen.nextInt(100);
                    randomList.add(num);
                }
                // check if valid order argument
                if(test.order.equals("random")){
                    System.out.println("Generating an array with " + test.size + " random
                    values.");
                }
                else if(test.order.equals("ascending")){
                    System.out.println("Generating an array with " + test.size + " ascending
                    values.");
                    Collections.sort(randomList);
                }
                else if(test.order.equals("descending")){
                    System.out.println("Generating an array with " + test.size + " descending
                    values.");
                    Collections.sort(randomList, Collections.reverseOrder());
                }
                else{
                    System.out.println("Invalid argument for array order. Program will now
                    exit.");
                    System.exit(1);
                }
                // convert Integer object to primitive integer
                for(int i=0; i<test.size; i++){
                    test.randArray[i] = randomList.get(i).intValue();
                }
                // testing
                test.sorter(test.size, test.sortingAlgorithm, test.randArray);
            }
        }
        else{
            System.out.println("Insufficient arguments. Program will now exit.");
            System.exit(1);
        }
    }
}
```