Part 1: Sorting and Searching: Algorithm Analysis (70 marks)

1. Write a Bubble Sort algorithm that sorts the data using a column based on your student number. If two items have the same value sort based on column 1.
   You will receive higher marks for optimal (low run-time) solutions. **Highlight in the submission the reason why you chose your sorting algorithm with reference to the run-time complexity.** The sorting algorithm must be your own implementation. You will receive 0 marks for using an imported library to complete this task.

```
package org.example.main;

import java.io.FileWriter;
import java.util.Comparator;

import org.example.people.People;
import org.example.people.PeopleReader;


// Question 1 method implementation and example

/**
* A generic class for sorting an array using the Bubble Sort algorithm.
*
* @param <T> The type of objects to be sorted.
*/
public class BubbleSort<T> {
   // init generic properties of data and comparator
   private final T[] data;
   private final Comparator<T> comparator;

   // constructor
   public BubbleSort(T[] data, Comparator<T> comparator) {
       this.data = data;
       this.comparator = comparator;
   }

   /**
    * Sorts the array using the Bubble Sort algorithm.
    *
```

```java
     * @return The sorted array.
     */
    public T[] bubbleSort() {
        // init length and swapped variables
        int n = this.data.length;
        boolean swapped;

        // outer loop
        for (int i = 0; i < n - 1; i++) {
            swapped = false;

            // inner loop
            for (int j = 0; j < n - i - 1; j++) {
                // compare indices
                int compareResult = comparator.compare(this.data[j], this.data[j + 1]);
                // if differences is greater than 0, swap elements
                if (compareResult > 0) {
                    // Swap elements
                    T temp = this.data[j];
                    this.data[j] = this.data[j + 1];
                    this.data[j + 1] = temp;
                    // set swapped as true for current index to avoid corruption
                    swapped = true;
                }
            }

            // If no swapping occurred, array is sorted
            if (!swapped) {
                break;
            }
        }
        return data;
    }

    public static void main(String[] args) {
        try {
            // init reader and read people into memory from csv
            PeopleReader peopleReader = new PeopleReader("resources/people.csv");
            People[] people = peopleReader.readPeople();


            // given we have set compareTo method in People class, the natural order of comparator
            // will compare based on our desired column, in this case age, and then next ID.
            Comparator<People> peopleComparator = Comparator
                    .<People>naturalOrder()
                    .thenComparing(People::getID);


            // init bubble sort and then sort the people returning new array
            BubbleSort<People> bs = new BubbleSort<>(people, peopleComparator);
            People[] sortedPeople = bs.bubbleSort();


            // get users home directory based on system
            String homeDirectory = System.getProperty("user.home");
            // init File writer and write file to home dir + /bubbleSortedPeople.csv
            try (FileWriter writer = new FileWriter(homeDirectory + "/bubbleSortedPeople.csv")) {
                // write header
                writer.write("ID,Name,Surname,Jon,Age,Credit\n");


                // for each person write it to csv file and add line break
                for (People person : sortedPeople) {
                    writer.write(person.toString() + "\n");
                }
            }
        } catch (Exception e) {
            System.err.println("Failed to read people: " + e.getMessage());
        }
    }
}
```

This bubble sort algorithm's runtime complexity can be measured in terms of O(n^2) where n is the number of elements or data in the array. This is both the average and worst case outcome due to the nature of bubble sort. The N^2 time complexity come due to the nested for loop iterating over the elements in the array twice. In the best case, if the array is already sorted the runtime complexity would be O(n).

2. Experimentally analyse the time complexity of your sorting algorithm that you wrote for question 1 above. **Show your results by taking the average elapsed time for 10, 100, 1000, 5000 and 10000 records.**

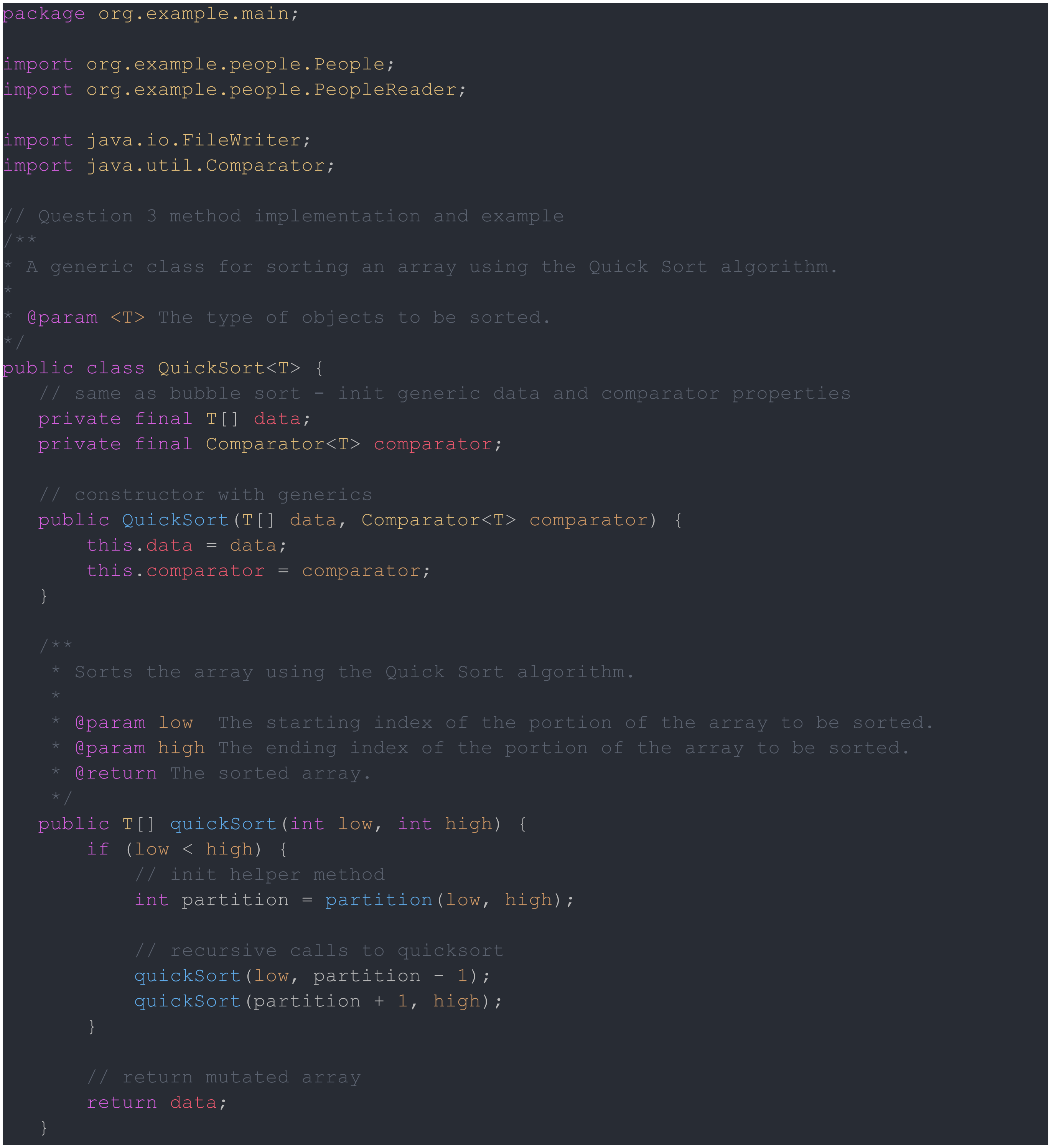
```
// Question 2 Implementation using unit test
@Test
void TestBubbleSortTime() {
   try {
       // Set up of data and reader
       PeopleReader peopleReader = new PeopleReader("resources/people.csv");
       People[] people = peopleReader.readPeople();

       // init comporator
                                         Comparator<People>     peopleComparator      =
Comparator.<People>naturalOrder().thenComparing(People::getID);

       // init sizes array
       int[] sizes = {10, 100, 1000, 5000, 10000};
       // loop through sizes
       for (int size : sizes) {
           // init time variable
           long totalTime = 0;
           for (int i = 0; i < 10; i++) {
               // copy array with current szr
               People[] subset = Arrays.copyOf(people, size);

               // init starttime in milliseconds
               long startTime = System.currentTimeMillis();
               BubbleSort<People> bs = new BubbleSort<>(subset, peopleComparator);
               People[] sortedPeople = bs.bubbleSort();
               long endTime = System.currentTimeMillis();

               totalTime += (endTime - startTime);
               AssertArraySorted(sortedPeople);
           }
           // get average time
           double averageTime = totalTime / 10.0;
              System.out.println("Bubble Sort - Average time for sorting " + size + "
records: " + averageTime + " ms");
       }

   } catch (Exception e) {
       System.err.println("Failed to read people: " + e.getMessage());
   }
}
```

Results:

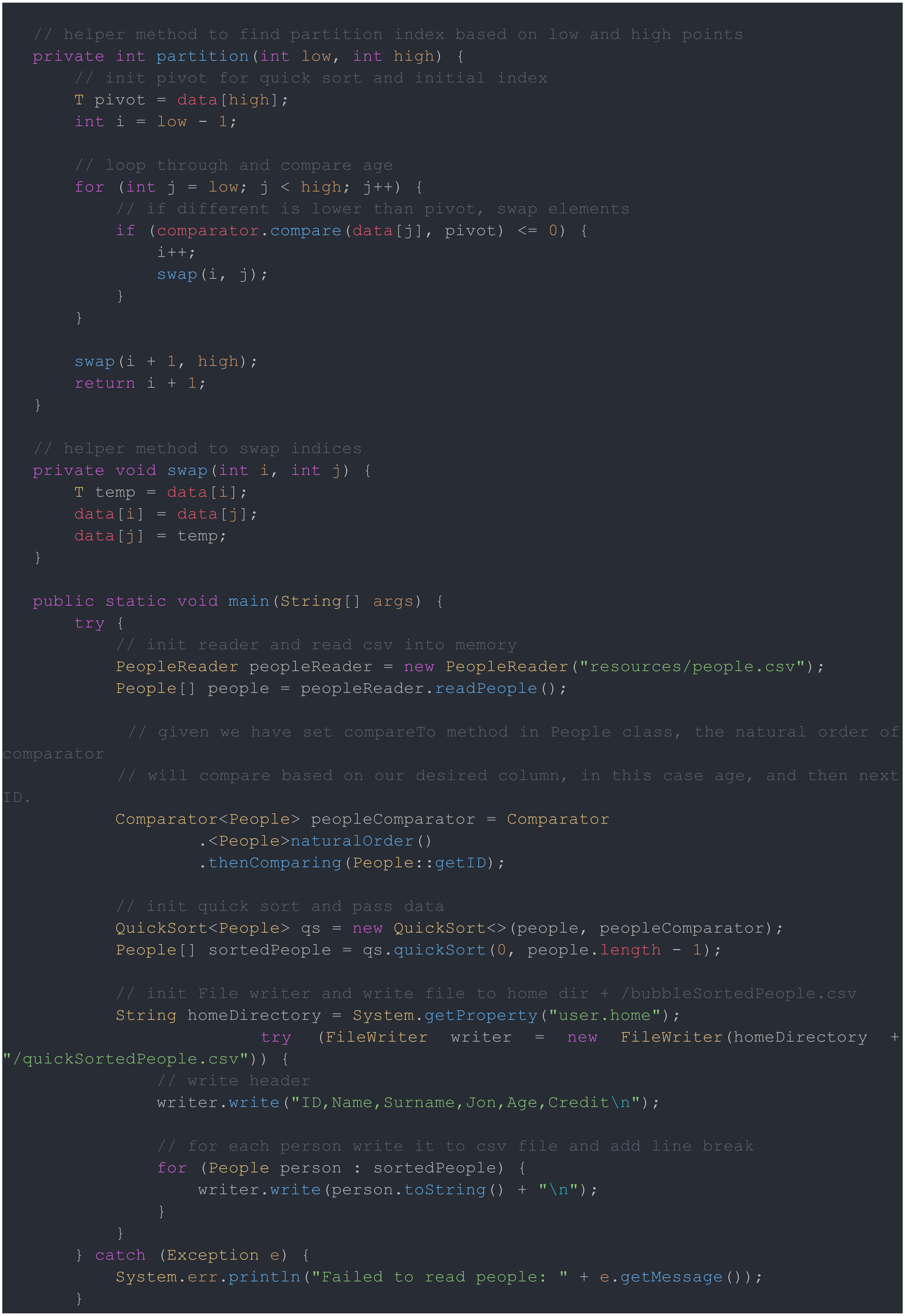
```
Bubble Sort - Average time for sorting 10 records: 0.0 ms
Bubble Sort - Average time for sorting 100 records: 0.5 ms
Bubble Sort - Average time for sorting 1000 records: 5.5 ms
Bubble Sort - Average time for sorting 5000 records: 75.4 ms
Bubble Sort - Average time for sorting 10000 records: 293.9 ms
```

Confirming the assumptions made in the description of question 1, we can confirm based on the above result that the bubble sort algorithm has an O(n^2) time complexity and that the differences in time can be said to be close to quadratic growth.

3. Write a Quick Sort algorithm that sorts the data using a column based on your student number. If two items have the same value sort based on column 1.
   You will receive higher marks for optimal (low run-time) solutions. **Highlight in the submission the reason why you chose your sorting algorithm with reference to the run-time complexity.** The sorting algorithm must be your own implementation. You will receive 0 marks for using an imported library to complete this task.



```java
package org.example.main;

import org.example.people.People;
import org.example.people.PeopleReader;

import java.io.FileWriter;
import java.util.Comparator;

// Question 3 method implementation and example
/**
* A generic class for sorting an array using the Quick Sort algorithm.
*
* @param <T> The type of objects to be sorted.
*/
public class QuickSort<T> {
   // same as bubble sort - init generic data and comparator properties
   private final T[] data;
   private final Comparator<T> comparator;

   // constructor with generics
   public QuickSort(T[] data, Comparator<T> comparator) {
       this.data = data;
       this.comparator = comparator;
   }

   /**
    * Sorts the array using the Quick Sort algorithm.
    *
    * @param low  The starting index of the portion of the array to be sorted.
    * @param high The ending index of the portion of the array to be sorted.
    * @return The sorted array.
    */
   public T[] quickSort(int low, int high) {
       if (low < high) {
           // init helper method
           int partition = partition(low, high);

           // recursive calls to quicksort
           quickSort(low, partition - 1);
           quickSort(partition + 1, high);
       }

       // return mutated array
       return data;
   }
```



```
    // helper method to find partition index based on low and high points
    private int partition(int low, int high) {
        // init pivot for quick sort and initial index
        T pivot = data[high];
        int i = low - 1;

        // loop through and compare age
        for (int j = low; j < high; j++) {
            // if different is lower than pivot, swap elements
            if (comparator.compare(data[j], pivot) <= 0) {
                i++;
                swap(i, j);
            }
        }

        swap(i + 1, high);
        return i + 1;
    }

    // helper method to swap indices
    private void swap(int i, int j) {
        T temp = data[i];
        data[i] = data[j];
        data[j] = temp;
    }

    public static void main(String[] args) {
        try {
            // init reader and read csv into memory
            PeopleReader peopleReader = new PeopleReader("resources/people.csv");
            People[] people = peopleReader.readPeople();

             // given we have set compareTo method in People class, the natural order of
comparator
            // will compare based on our desired column, in this case age, and then next
ID.
            Comparator<People> peopleComparator = Comparator
                    .<People>naturalOrder()
                    .thenComparing(People::getID);

            // init quick sort and pass data
            QuickSort<People> qs = new QuickSort<>(people, peopleComparator);
            People[] sortedPeople = qs.quickSort(0, people.length - 1);

            // init File writer and write file to home dir + /bubbleSortedPeople.csv
            String homeDirectory = System.getProperty("user.home");
                         try (FileWriter writer = new FileWriter(homeDirectory +
"/quickSortedPeople.csv")) {
                // write header
                writer.write("ID,Name,Surname,Jon,Age,Credit\n");

                // for each person write it to csv file and add line break
                for (People person : sortedPeople) {
                    writer.write(person.toString() + "\n");
                }
            }
        } catch (Exception e) {
            System.err.println("Failed to read people: " + e.getMessage());
        }
```

```
   }
}
```

In the average and best case scenarios, this quick sort algorithm will have O(nlogn) time complexity where n is the number of elements in the array. This is due to the quicksort algorithm dividing the array into two roughly equal halves. This type of time complexity is faster than quadratic growth (O(n^2), but slower than linear growth O(n). However, the worst case scenario occurs when the pivot chosen is either the first or last element of the array, therefore quicksort will recursively go through all elements in O(n^2) time, leading to a quadratic time growth.

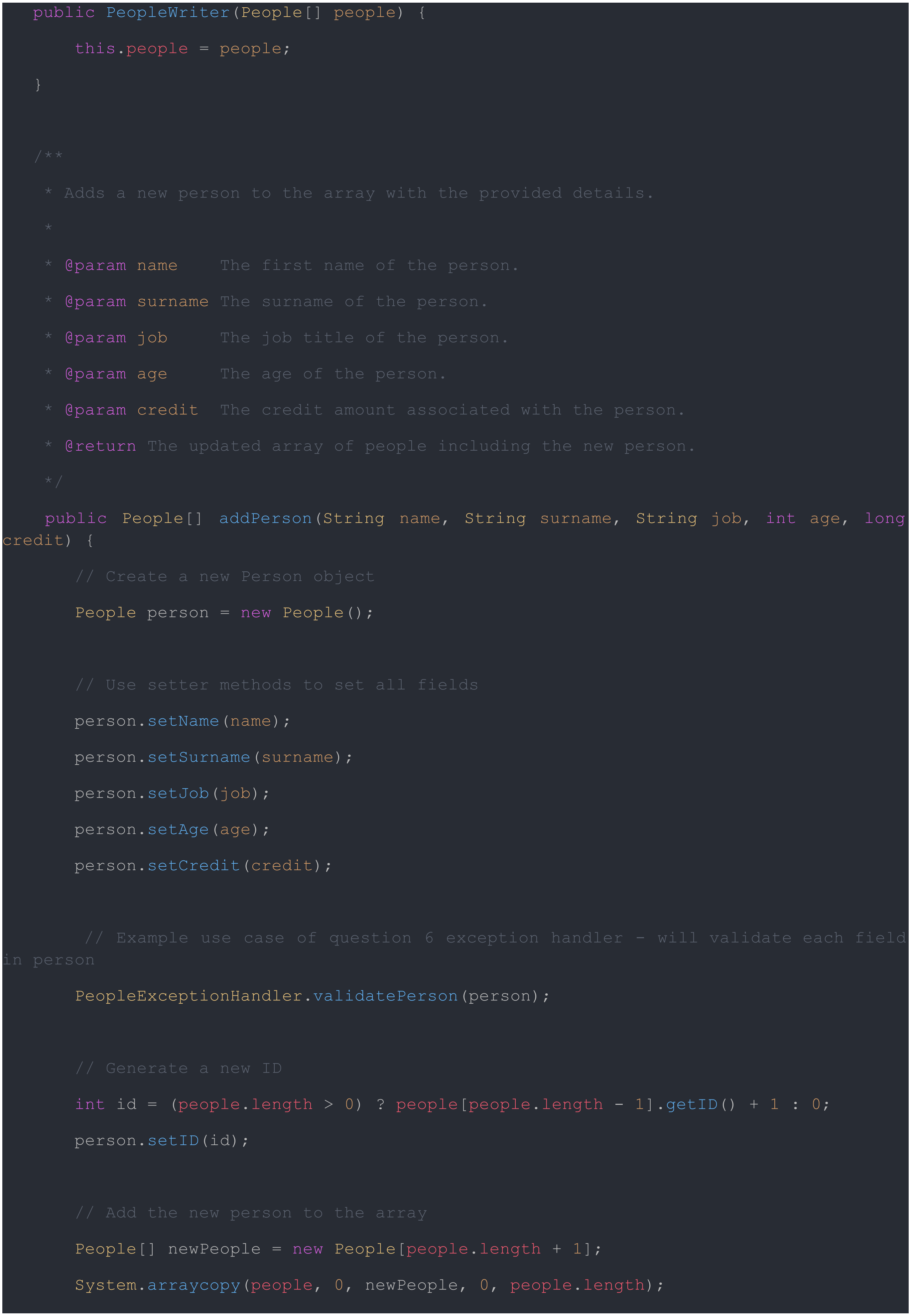
4. Write a Binary Search algorithm that accepts a sorted column type and searches the data record from the dataset. For this you can use the sort() Java method to sort the elements in that column (can be any column between 2 and 4). If an element X is not found, display “X was not found in the Y list!”, where Y is the title of the chosen sorted column (e.g., Name, Country, Location, Age, etc.). If the element was found in the list, display "X was found in the Y list".

```
package org.example.main;

import org.example.people.People;
import org.example.people.PeopleReader;

import java.util.Arrays;
import java.util.Comparator;
import java.util.function.Function;


// Question 4 method implementation and example

/**
* A generic class for performing binary search on an array.
*
* @param <T> The type of objects in the array.
*/
public class BinarySearch<T> {
   // generic property
   private final T[] data;

   // constructor with generic
   public BinarySearch(T[] data) {
       this.data = data;
   }

   /**
    * Performs a binary search on the array for a specified target value.
    *
     * @param func    A function to extract a comparable key from each element in the
array.
    * @param target The target value to search for, of type U.
    * @param <U>    The type of the comparable key extracted from the elements.
    * @return The index of the target value in the array, or -1 if not found.
    */
   public <U extends Comparable<U>> int binarySearch(Function<T, U> func, U target) {
       // init comparator with callback function and sort array based on comparator
       Comparator<T> comparator = Comparator.comparing(func);
       Arrays.sort(data, comparator);

       // init low and high indices as start and end of array - two pointer approach
       int low = 0;
       int high = data.length - 1;

       // loop through until pointers meet
       while (low <= high) {
           // get mid point
           int mid = (low + high) / 2;
           // extract generic data from the index of midpoint
           T midVal = data[mid];
           // apply comparison of midval to target through callback function
           int cmp = func.apply(midVal).compareTo(target);

           // if difference less than 0 were too low
           if (cmp < 0) {
               low = mid + 1;
               // if greater than zero were too high
           } else if (cmp > 0) {
               high = mid - 1;
           } else {
               // goldilocks zone were just right
               return mid;
           }
       }
       // target not found
       return -1;
   }

   public static void main(String[] args) {
       // Scanner could be used to take user input of course
       try {
           // init people reader and read people from csv into memory
           PeopleReader peopleReader = new PeopleReader("resources/people.csv");
           People[] people = peopleReader.readPeople();

           // init binary search
           BinarySearch<People> bs = new BinarySearch<>(people);

           // select column and target, could we taken from user input if using scanner
           // if using age, pass age as a string and it will be parsed to an integer in
switch statement
           String columnToSearch = "name";
           String targetValue = "Joe";

           // switch statement for columns 2 - 4 of people
           int result = switch (columnToSearch.toLowerCase()) {
               case "name" -> bs.binarySearch(People::getName, targetValue);
               case "surname" -> bs.binarySearch(People::getSurname, targetValue);
               case "job" -> bs.binarySearch(People::getJob, targetValue);
               case "age" -> bs.binarySearch(People::getAge,
Integer.parseInt(targetValue));
               default -> throw new IllegalArgumentException("Invalid column: " +
columnToSearch);
           };

           // if result found communicate to user else let them know its not found
           if (result != -1) {
               System.out.println(targetValue + " was found in the " + columnToSearch +
" list");

               System.out.println("Record details: " + people[result]);

           } else {

                       System.out.println(targetValue + " was not found in the " +
columnToSearch + " list!");

           }

       } catch (Exception e) {

           System.err.println("Failed to read people: " + e.getMessage());

       }

   }

}
```

The time complexity of the binary search algorithm could be described as O(nlogn); whereby sorting of the array and searching the array are both O(nlogn) leading to a result of 2O(nlogn) but constants are cancelled out.

5. Write a Java program that accepts a new record (with all the six fields) and adds it at the end of the record array, with a new consecutive ID number.

```
package org.example.main;


import org.example.people.People;

import org.example.people.PeopleExceptionHandler;

import org.example.people.PeopleReader;


import java.io.FileWriter;


// Question 5 Example


/**

* A class for managing and writing people data.

*/

public class PeopleWriter {

    // init people

    private People[] people;


    // constructor
```



```java
    public PeopleWriter(People[] people) {
        this.people = people;
    }

    /**
     * Adds a new person to the array with the provided details.
     *
     * @param name    The first name of the person.
     * @param surname The surname of the person.
     * @param job     The job title of the person.
     * @param age     The age of the person.
     * @param credit  The credit amount associated with the person.
     * @return The updated array of people including the new person.
     */
     public People[] addPerson(String name, String surname, String job, int age, long credit) {
        // Create a new Person object
        People person = new People();

        // Use setter methods to set all fields
        person.setName(name);
        person.setSurname(surname);
        person.setJob(job);
        person.setAge(age);
        person.setCredit(credit);

         // Example use case of question 6 exception handler - will validate each field in person
        PeopleExceptionHandler.validatePerson(person);

        // Generate a new ID
        int id = (people.length > 0) ? people[people.length - 1].getID() + 1 : 0;
        person.setID(id);

        // Add the new person to the array
        People[] newPeople = new People[people.length + 1];
        System.arraycopy(people, 0, newPeople, 0, people.length);
        newPeople[people.length] = person;
        this.people = newPeople;

        // return copied array with new person at the end with consecutive id
        return newPeople;
    }

    public static void main(String[] args) {
        try {
            // init people reader and read people into memory from csv
            PeopleReader peopleReader = new PeopleReader("resources/people.csv");
            People[] people = peopleReader.readPeople();

            // init people writer clas and add new person with approptiate values
            // user input could be taken from scanner if necessary
            PeopleWriter peopleWriter = new PeopleWriter(people);
             People[] newPeople = peopleWriter.addPerson("Mark", "Grant", "Manager", 33,
472132554L);

            // write file to users home directory with new person
            String homeDirectory = System.getProperty("user.home");
            try (FileWriter writer = new FileWriter(homeDirectory + "/newPeople.csv")) {
                // write header
                writer.write("ID,Name,Surname,Job,Age,Credit\n");

                // loop through new people and write with a line break
                for (People person : newPeople) {
                    writer.write(person.toString() + "\n");
                }
            }
        } catch (Exception e) {
            System.err.println("Failed to read people: " + e.getMessage());
        }
    }
}
```

6. Write a Java Exception that handles special cases and communicates to users to correct the cases. A typical special case is that the "name" field cannot be empty or cannot contain digits only. The exception should generate a message similar to the following: “Person's name cannot be empty. It cannot have only digits! Please correct this!” (15 M

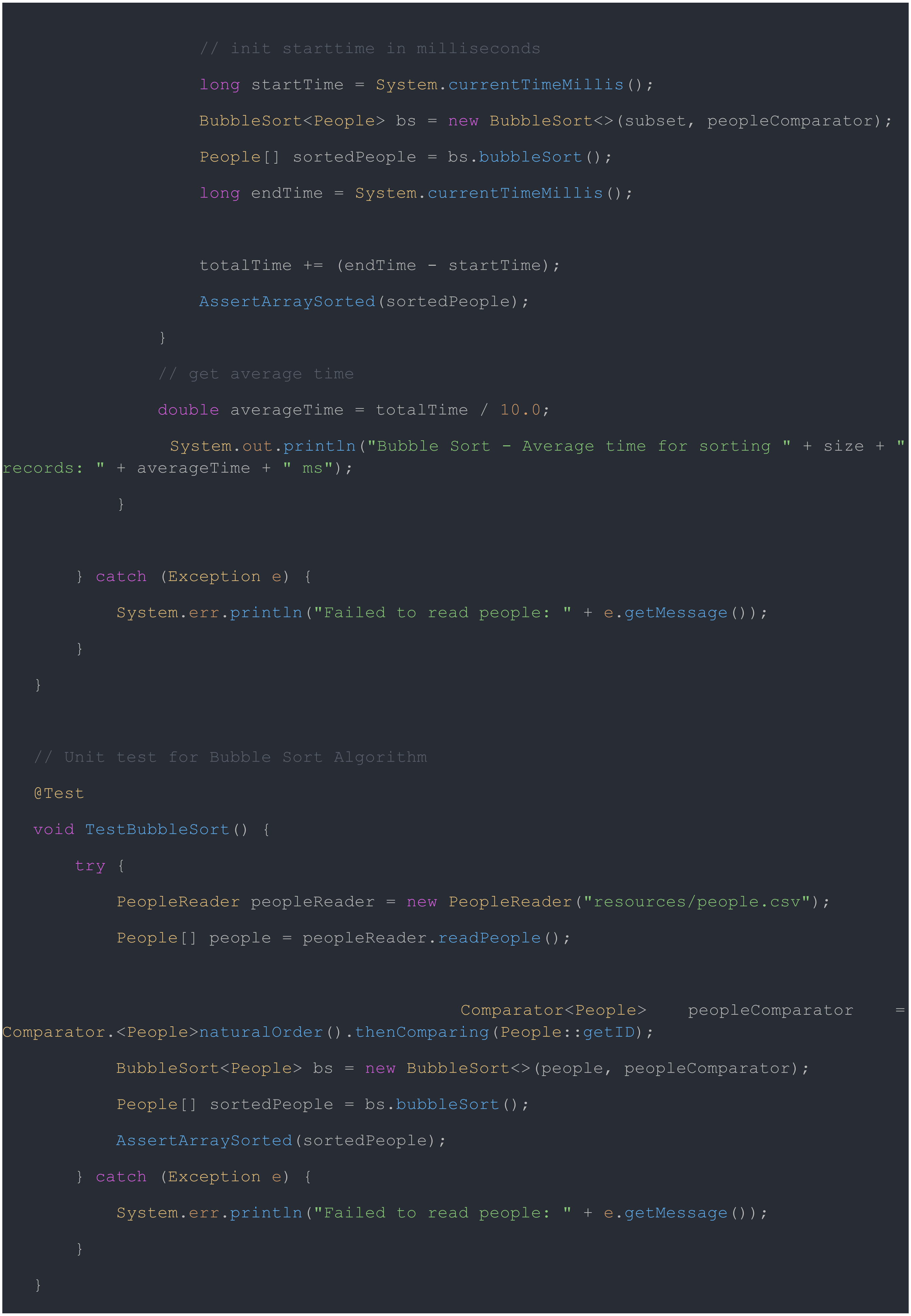
```java
package org.example.people;


// Question 6 class implementation - example is instantiate in People Writer class


/**
* Custom exception handler for validating People objects.
* This class extends RuntimeException and provides methods to validate
* various properties of People objects.
*/
public class PeopleExceptionHandler extends RuntimeException {
   // super method
   public PeopleExceptionHandler(String message) {
       super(message);
   }

   /**
    * Validates all properties of the given People object -- Controller method
    *
    * @param person The People object to validate.
    */
   public static void validatePerson(People person) {
       validateName(person.getName());
       validateSurname(person.getSurname());
       validateJob(person.getJob());
       validateAge(person.getAge());
       validateCredit(person.getCredit());
   }

   // validate name - must be not null, emtpy string or cant be a digit
   private static void validateName(String name) {
       if (name == null || name.trim().isEmpty() || name.matches("\\d+")) {
           throw new PeopleExceptionHandler("Person's name cannot be empty or only have digits!");
```

```
        }
    }

    // validate surname - must be not null, emtpy string or cant be a digit
    private static void validateSurname(String surname) {
        if (surname == null || surname.trim().isEmpty() || surname.matches("\\d+")) {
            throw new PeopleExceptionHandler("Person's surname cannot be empty or only have digits!");
        }
    }

    // validate job, can't be null or empty
    private static void validateJob(String job) {
        if (job == null || job.trim().isEmpty()) {
                throw new PeopleExceptionHandler("Person's job cannot be empty. Please provide a valid job title.");
        }
    }

    // validate age must be greater than zero
    private static void validateAge(int age) {
        if (age <= 0) {
            throw new PeopleExceptionHandler("Person's age must be greater than zero. Please provide a valid age.");
        }
    }

    // validate credit must be greater than zero
    private static void validateCredit(long credit) {
        if (credit < 0) {
            throw new PeopleExceptionHandler("Person's credit cannot be negative. Please provide a valid credit value.");
        }
    }
}
```

Appendix:

// Test Class

```
import org.example.main.BinarySearch;

import org.example.main.BubbleSort;

import org.example.main.QuickSort;

import org.example.people.People;

import org.example.people.PeopleReader;

import org.junit.jupiter.api.Test;


import java.util.Arrays;

import java.util.Comparator;


import static org.junit.jupiter.api.Assertions.assertTrue;

import static org.junit.jupiter.api.Assertions.fail;


public class SorterTest {


    // Question 2 Implementation using unit test

    @Test

    void TestBubbleSortTime() {

        try {

            // Set up of data and reader

            PeopleReader peopleReader = new PeopleReader("resources/people.csv");

            People[] people = peopleReader.readPeople();


            // init comporator

            Comparator<People> peopleComparator = Comparator.<People>naturalOrder().thenComparing(People::getID);


            // init sizes array

            int[] sizes = {10, 100, 1000, 5000, 10000};

            // loop through sizes

            for (int size : sizes) {

                // init time variable

                long totalTime = 0;

                for (int i = 0; i < 10; i++) {

                    // copy array with current szr

                    People[] subset = Arrays.copyOf(people, size);
```



```java
                    // init starttime in milliseconds
                    long startTime = System.currentTimeMillis();
                    BubbleSort<People> bs = new BubbleSort<>(subset, peopleComparator);
                    People[] sortedPeople = bs.bubbleSort();
                    long endTime = System.currentTimeMillis();

                    totalTime += (endTime - startTime);
                    AssertArraySorted(sortedPeople);
                }
                // get average time
                double averageTime = totalTime / 10.0;
                 System.out.println("Bubble Sort - Average time for sorting " + size + " records: " + averageTime + " ms");
            }

        } catch (Exception e) {
            System.err.println("Failed to read people: " + e.getMessage());
        }
    }

    // Unit test for Bubble Sort Algorithm
    @Test
    void TestBubbleSort() {
        try {
            PeopleReader peopleReader = new PeopleReader("resources/people.csv");
            People[] people = peopleReader.readPeople();

            Comparator<People> peopleComparator = Comparator.<People>naturalOrder().thenComparing(People::getID);
            BubbleSort<People> bs = new BubbleSort<>(people, peopleComparator);
            People[] sortedPeople = bs.bubbleSort();
            AssertArraySorted(sortedPeople);
        } catch (Exception e) {
            System.err.println("Failed to read people: " + e.getMessage());
        }
    }
    // Unit Test for Quick sort
    @Test
    void TestQuickSort() {
        try {
            PeopleReader peopleReader = new PeopleReader("resources/people.csv");
            People[] people = peopleReader.readPeople();

            Comparator<People> peopleComparator = Comparator.<People>naturalOrder().thenComparing(People::getID);
            QuickSort<People> qs = new QuickSort<>(people, peopleComparator);
            People[] sortedPeople = qs.quickSort(0, people.length - 1);
            AssertArraySorted(sortedPeople);
        } catch (Exception e) {
            System.err.println("Failed to read people: " + e.getMessage());
        }
    }

    // Unit Test for Binary Search
    @Test
    void TestBinarySearchString() {
        try {
            PeopleReader peopleReader = new PeopleReader("resources/people.csv");
            People[] people = peopleReader.readPeople();

            BinarySearch<People> bs = new BinarySearch<>(people);

            String columnToSearch = "name";
            String targetValue = "Joe";

            int result = switch (columnToSearch.toLowerCase()) {
                case "name" -> bs.binarySearch(People::getName, targetValue);
                case "surname" -> bs.binarySearch(People::getSurname, targetValue);
                case "job" -> bs.binarySearch(People::getJob, targetValue);
                case "age" -> bs.binarySearch(People::getAge, Integer.parseInt(targetValue));
                default -> throw new IllegalArgumentException("Invalid column: " + columnToSearch);
            };


            if (result != -1) {
                System.out.println(targetValue + " was found in the " + columnToSearch + " list");
                System.out.println("Record details: " + people[result]);
            } else {
                System.out.println(targetValue + " was not found in the " + columnToSearch + " list!");
            }
        } catch (Exception e) {
            System.err.println("Failed to read people: " + e.getMessage());
        }
    }


    // Unit Test for Binary search for age
    @Test
    void TestBinarySearchInt() {
        try {
            PeopleReader peopleReader = new PeopleReader("resources/people.csv");
            People[] people = peopleReader.readPeople();


            BinarySearch<People> bs = new BinarySearch<>(people);


            String columnToSearch = "age";
            // set age and string and parse in switch statement - if scanner was used for user input, then input could be parsed too
            String targetValue = "56";


            int result = switch (columnToSearch.toLowerCase()) {
                case "name" -> bs.binarySearch(People::getName, targetValue);
                case "surname" -> bs.binarySearch(People::getSurname, targetValue);
                case "job" -> bs.binarySearch(People::getJob, targetValue);
                case "age" -> bs.binarySearch(People::getAge, Integer.parseInt(targetValue));
                default -> throw new IllegalArgumentException("Invalid column: " + columnToSearch);
            };
            if (result != -1) {
                System.out.println(targetValue + " was found in the " + columnToSearch + " list");
                System.out.println("Record details: " + people[result]);
            } else {
                System.out.println(targetValue + " was not found in the " + columnToSearch + " list!");
            }
        } catch (Exception e) {
            System.err.println("Failed to read people: " + e.getMessage());
        }
    }

    // Helper Method to check if Array is sorted by age and then also ID
    private void AssertArraySorted(People[] people) {
        for (int i = 0; i < people.length - 1; i++) {

            if (people[i].compareTo(people[i + 1]) <= 0) {
                assertTrue(people[i].compareTo(people[i + 1]) <= 0, "Array not sorted at index " + i);
                if (people[i].compareTo(people[i + 1]) == 0) {
                    assertTrue(people[i].getID() < people[i + 1].getID(), "Array not sorted at index " + i);
                }
            } else {
                int start = Math.max(0, i - 2);
                int end = Math.min(people.length - 1, i + 3);
                System.out.println("Surrounding elements:");
                for (int j = start; j <= end; j++) {
                    System.out.println("Index " + j + ": " + people[j]);
                }
                fail("Array not sorted at index " + i);
            }
        }
    }
}
```

// People Reader class

```java
package org.example.people;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

/**
* Utility class for reading People data from a CSV file.
*/
public class PeopleReader {
    // file path property
    private final String filePath;

    // constructor
    public PeopleReader(String filePath) {
        this.filePath = filePath;
    }

    /**
     * Reads People data from the CSV file and loads it into an array.
     *
     * @return An array of People objects read from the CSV file.
     */
    public People[] readPeople() {
        // init properties of file path and file name
        File directory = new File(filePath);
        String name = directory.getAbsolutePath();
        // init new array of people of length 10000
        People[] people = new People[10000];

        // use buffer reader instead of scanner as it is more efficient
        try (BufferedReader buffer = new BufferedReader(new FileReader(name))) {
            buffer.readLine(); // Skip csv header
            // init pointerz
            String line;
            int i = 0;

            // while buffer is inbounds
            while ((line = buffer.readLine()) != null && i < people.length) {
                try {
                    // split each data point on comma, given the file is a csv
                    String[] data = line.split(",");
                    // read people into memory
                    people[i++] = new People(
                            Integer.parseInt(data[0]),
                            data[1],
                            data[2],
                            data[3],
                            Integer.parseInt(data[4]),
                            Long.parseLong(data[5])
                    );
                    // catch exceptions
                } catch (NumberFormatException | ArrayIndexOutOfBoundsException e) {
                    System.err.println("Error parsing line: " + line);
                    System.err.println("Error details: " + e.getMessage());
                }
            }
        } catch (IOException e) {
            System.err.println("Error reading file: " + e.getMessage());
            return null;
        }

        // return people array
        return people;
    }
}
```

// People Class

```
package org.example.people;
```

```
/**
* Represents a person with various attributes such as ID, Name, Surname, Job, Age, and
Credit.
* Implements the Comparable interface to allow for comparison based on age.
*/
public class People implements Comparable<People> {
    // properties
    private int ID;
    private String Name;
    private String Surname;
    private String Job;
    private int Age;
    private long Credit;

    // constructor
    public People(int ID, String Name, String Surname, String Job, int Age, long Credit)
{
        this.ID = ID;
        this.Name = Name;
        this.Surname = Surname;
        this.Job = Job;
        this.Age = Age;
        this.Credit = Credit;
    }

    // no args constructor
    public People() {}

     // override compare to method, which will allow for easy comparisons in algorithm
classes
    @Override
    public int compareTo(People person) {
        return Integer.compare(this.Age, person.Age);
    }

    // to string method for easy writing
    @Override
    public String toString() {
        return (
        "Person [ID= " + ID + ", Name= " + Name + ", Surname= "
        + Surname + ", Job= " + Job + ", Age= "
        + Age + ", Credit= " + Credit + "]"
        );
    }

    // getters and setters
    public int getID() {
        return ID;
    }

    public void setID(int ID) {
        this.ID = ID;
    }

    public String getName() {
        return Name;
    }

    public void setName(String name) {
        this.Name = name;
    }

    public String getSurname() {
        return Surname;
    }

    public void setSurname(String surname) {
        this.Surname = surname;
    }

    public String getJob() {
        return Job;
    }
    public void setJob(String job) {
        this.Job = job;
    }

    public int getAge() {
        return Age;
    }

    public void setAge(int age) {
        this.Age = age;
    }

    public long getCredit() {
        return Credit;
    }

    public void setCredit(long credit) {
        this.Credit = credit;
    }

}
```

// pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>AlgoCA</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
```

```
        <maven.compiler.source>21</maven.compiler.source>
        <maven.compiler.target>21</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
            <version>5.11.0-M1</version>
        </dependency>
    </dependencies>

</project>
```

// Readme

```
### Important Information
* This project was created in intellij using maven build architecture to support the use of unit tests
* Source code and packages can be found in the src directory
* tests can be found in the test directory
* csv file will be read from the resources directory
* I presume netbeans/eclipse have built in maven support for building the project dependancies
* Thanks Hamilton!
```