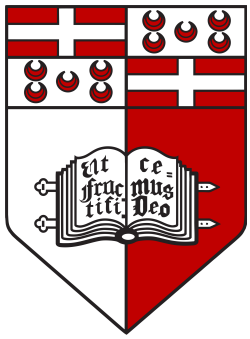


# CSA 1017
# Data Structures and Algorithms 1
## Assignment

matrices Mark Said Camilleri
B.Sc. (Hons) (Computing Science)
University of Malta

Friday 4th May 2016

# Contents

# Statement of Completion

The questions below were the ones that have been attempted:

**Question 1** This question has been successfully completed.

**Question 2** This question has been successfully completed.

**Question 3** This question has been successfully completed.

**Question 4** This question has been successfully completed.

**Question 5** This question has been successfully completed.

**Question 6** This question has been successfully completed.

**Question 7** This question has been successfully completed.

**Question 8** This question has been successfully completed.

**Question 9** This question has been successfully completed.

Signature

Date

# Task 1

# Arabic to Roman Numeral Converter

| Input | Expected Output | Actual Output |
|---|---|---|
| 1 | I | I |
| 2 | II | II |
| 4 | IV | IV |
| 5 | V | V |
| 6 | VI | IV |
| 9 | IX | IX |
| 10 | X | X |
| 20 | XX | XX |
| 40 | XL | XL |
| 49 | XLIX | XLIX |
| 50 | L | L |
| 60 | LX | LX |
| 90 | XC | XC |
| 99 | XCIX | XCIX |
| 100 | C | C |
| 150 | CL | CL |
| 400 | CD | CD |
| 499 | CDXCIX | CDXCIX |
| 500 | D | D |
| 600 | DC | DC |
| 900 | CM | CM |
| 999 | CMXCIX | CMXCIX |
| 1000 | M | M |
| 1024 | MXXIV | MXXIV |
| 2000 | The input was not a valid number between 1 and 1024 Please try again and enter a number to convert: | The input was not a valid number between 1 and 1024 Please try again and enter a number to convert: |
| abc | The input was not a valid number between 1 and 1024 Please try again and enter a number to convert: | The input was not a valid number between 1 and 1024 Please try again and enter a number to convert: |

## 1.1 The source code for Task 1

```
/**
 * @author Mark Said Camilleri
 * @version 20160509
 */


import java.util.InputMismatchException;
import java.util.Scanner;


public class Question1 {

    public static void main(String args[]) {

        //Initialize Scanner object
        Scanner in = new Scanner(System.in);
        in.useDelimiter("\n");

        /*======================== WELCOME MESSAGE TO USER ========================*/
        System.out.println("-----------------------------------------------------");
        System.out.println("|    CSA 1017 - Data Structures and Algorithms 1    |");
        System.out.println("|---------------------------------------------------|");
        System.out.println("|         Submission by Mark Said Camilleri         |");
        System.out.println("|     Task 1: Arabic to Roman Numeral Converter     |");
        System.out.println("|---------------------------------------------------|");
        System.out.print("| Please enter a number between 1 and 1024: ");


        int toConvert = 0; //value to be converted.
        boolean isError; // temporary boolean value used for error checking of the input.
        do {
            isError = false;
            try {
                toConvert = in.nextInt();
            } catch (InputMismatchException e) {
                isError = true;
                in.next(); //To clear the buffer
            }
            /*=========== Makes sure input is a number is between 1 and 1024 ==============*/
            if (isError || toConvert < 1 || toConvert > 1024) {
                /*==================== OUTPUT ERROR MESSAGE TO THE USER ==================*/
                System.out.println("|---------------------------------------------------|");
                System.out.println("|The input was not a valid number
```

```
    between 1 and 1024|");
44                  System.out.print("|Please try again and enter a number
    to convert: ");
45              }
46          } while (isError || toConvert < 1 || toConvert > 1024);
47
48          System.out.printf("| %4d = %-24s in Roman Numerals |", toConvert
    , convert(toConvert));
49
50      }
51
52      /**
53       * Takes an int decimal value and outputs a string of the same value
        in Roman Numerals.
54       *
55       * @param toConvert the decimal value to ve converted to Roman
    Numerals
56       * @return The roman numeral equivalent of the input parameter
57       */
58      private static String convert(int toConvert) {
59
60          //Defining the decimal and roman counterparts
61          final int dec[] = {1, 4, 5, 9, 10, 40, 50, 90, 100, 400, 500,
    900, 1000};
62          final String rom[] = {"I", "IV", "V", "IX", "X", "XL", "L", "XC"
    , "C", "CD", "D", "CM", "M"};
63
64          /* Begins by checking the input paramerer against the largest
    roman numeral/numeral pair.
65           * and works it's way down to the unit numeral.
66           */
67          for (int i = dec.length - 1; i >= 0; i--) {
68              //If the value is larger, then the output is concatenated
    with the output of the difference.
69              if (toConvert >= dec[i])
70                  return rom[i] + convert(toConvert - dec[i]);
71          }
72          return ""; //What to return at 0, the base case.
73      }
74  }
```

# Task 2

# Reverse Polish Notation Evaluator

| Input | Expected Output | Actual Output |
|---|---|---|
| 4 5 + | 9.0 | 9.0 |
| 12 6 - | 6.0 | 6.0 |
| 3 2 / | 1.5 | 1.5 |
| 7 2 * | 14.0 | 14.0 |
| 23 85 + 92 * | 9936.0 | 9936.0 |
| 43.5 3.2 - 4.5 * 3.24 + | 184.59 | 184.59 |
| matrices 34.8 62.11 * -76 / | -28.43984211 | -28.439842105263157 |
| 282 -56 * 1024 - -55.6 * | 934969.6 | 934969.6 |
| Test | Your expression contained invalid characters. For input string "T" Your expression is invalid. Evaluation failed | Your expression contained invalid characters. For input string "T" Your expression is invalid. Evaluation failed |
| 1 + | Stack is Empty. Your expression is invalid. Evaluation failed. | Stack is Empty. Your expression is invalid. Evaluation failed. |
| 3 64 6 + | The stack has not been emptied. There are too many operands in your expression. Your expression is invalid. Evaluation failed. | The stack has not been emptied. There are too many operands in your expression. Your expression is invalid. Evaluation failed. |
| 4 0 / | Infinity | Infinity |

## 2.1 The source code for Task 2

### 2.1.1 Stack Class

```
1  import java.util.ArrayList;
2  import java.util.Collection;
3  import java.util.EmptyStackException;
4
5  /**
6   * Created by mark on 14/02/16.
7   *
8   * A stack implemented as an ArrayList to have it dynamically increase
      its size.
9   */
10 public class Stack<E> extends ArrayList {
11
12     /**
13      * Default constructor. Calls the ArrayList default constructor
14      */
15     public Stack() {
16         super();
17     }
18
19     /**
20      * Initialised a stack with the contents of the Collection in the
      parameter.
21      *
22      * @param c the contents to initialise the stack with.
23      */
24     public Stack(Collection<? extends E> c) {
25         super(c);
26     }
27
28     /**
29      * Pushes the data onto the stack
30      *
31      * @param data data to be pushed on the stack
32      * @throws IndexOutOfBoundsException if not sucessfully pushed
33      */
34     public void push(E data) throws IndexOutOfBoundsException {
35         int prevSize = this.size();
36         this.add(data);
37
38         //Condition to check if the data has been successfully added.
39         if (!(this.size() == prevSize + 1))
40             throw new IndexOutOfBoundsException("Failed to push to stack
      ");
41     }
42
43     /**
44      * Pops the topmost item from the stack.
45      * @return the data from the top of the stack is not empty.
46      * @throws EmptyStackException if the stack is empty.
47      */
48     public E pop() throws ArrayIndexOutOfBoundsException {
49         if (this.size() == 0) throw new EmptyStackException();
```

```
50          else return (E) this.remove(this.size() - 1);
51      }
52
53      /**
54       * Returns the data at the top of the stack without popping it.
55       * @return the data if the stack is not empty. null if it is empty.
56       */
57      public E peek() {
58          if (this.size() == 0) return null;
59          else return (E) this.get(this.size() - 1);
60      }
61
62      /**
63       * Returns a string representation of the contents of the stack.
64       * @overrides toString() in class AbstractCollection<E>
65       * @return The string representation of the ArrayList if not empty.
    "Stack is empty" if it is empty.
66       */
67      public String toString() {
68          if (this.size() == 0) return "Stack is empty";
69          else return super.toString();
70      }
71  }
```

### 2.1.2 Question 2 Main Class

```
1   import java.util.EmptyStackException;
2   import java.util.Scanner;
3
4   /**
5    * Created by mark on 09/02/16.
6    * This answer assumes the RPN input is correct.
7    */
8   public class Question2 {
9       public static void main(String args[]) {
10
11          //Initialize Scanner object
12          Scanner in = new Scanner(System.in).useDelimiter("\n");
13
14          /*======================== WELCOME MESSAGE TO USER
    ========================*/
15          System.out.println("
    ------------------------------------------------------");
16          System.out.println("|    CSA 1017 - Data Structures and
    Algorithms 1    |");
17          System.out.println("
    |------------------------------------------------------|");
18          System.out.println("|         Submission by Mark Said Camilleri
            |");
19          System.out.println("|     Task 2: Reverse Polish Notation
    evaluator     |");
20          System.out.println("
    |------------------------------------------------------|");
21          System.out.println("|     Note: This program can only do +,-,*
    and /     |");
22          System.out.print("| Please enter an expression to evaluate: ");
    //prompt for user input. Assumes it is rpn.
```

```
23
24          /* Initialized a stack object (using the stack defined here).
25           * Note, no importing of the Stack class.
26           */
27          Stack<Double> nums = new Stack<>();
28
29          //Reads user input. Must be a valid RPN expression.
30          String expression = in.next();
31          System.out.println("
    |-----------------------------------------------------|");
32          System.out.println("| Contents of the stack at each step:
                |");//some message to user.
33
34          boolean exceptionRaised = false; //used for error checking.
35          try {
36              //Iterates through the string inputted by the user.
37              for (int i = 0; i < expression.length(); i++) {
38                  char cChar = expression.charAt(i);
39
40                  /* If the current character is a space,
41                   * nothing needs to be done.
42                   */
43                  if (Character.isWhitespace(cChar)) continue;
44
45                  /* If it's a '+', then 2 numbers are popped, added and
46                   * the answer is pushed onto the stack.
47                   */
48                  else if (cChar == '+') {
49                      double num1 = nums.pop();
50                      double num2 = nums.pop();
51
52                      nums.push(num2 + num1);
53                  }
54                  /* If it's a '-', then 2 numbers are popped,
55                   * subtracted and the answer is pushed onto the stack.
56                   * The conjunction is to make sure that it's not
57                   * detecting a negative number. The disjunction is
58                   * true if the '-' is at the end of the string or
59                   * there is a space after it. These both make sure
60                   * that the '-' does not belong to a negative number
61                   */
62                  else if (cChar == '-' && (i == expression.length() - 1
    || Character.isWhitespace(expression.charAt(i + 1)))) {
63                      double num1 = nums.pop();
64                      double num2 = nums.pop();
65
66                      nums.push(num2 - num1);
67                  }
68                  /* If it's a '*', then 2 numbers are popped,
69                   * multiplied and the answer is pushed onto the stack.
70                   */
71                  else if (cChar == '*') {
72                      double num1 = nums.pop();
73                      double num2 = nums.pop();
74
75                      nums.push(num2 * num1);
76                  }
77                  /* If it's a '/', then 2 numbers are popped,
```

```
78                 * divided and the answer is pushed onto the stack.
79                 */
80                else if (cChar == '/') {
81                    double num1 = nums.pop();
82                    double num2 = nums.pop();
83
84                    nums.push(num2 / num1);
85                }
86            /*
87             * Otherwise, assuming it's inputted correctly, the
88             * character must be a number. In which case it is
89             * converted ot a double (allowing for any real number
90             * to be inputted) and pushed onto the stack.
91             */
92                else {
93                    int start = i++;
94                    while (Character.isDigit(expression.charAt(i)) ||
    expression.charAt(i) == '.')
95                        i++;
96
97                    nums.push(Double.parseDouble(expression.substring(
    start, i)));
98                }
99                System.out.printf("| %-49s |\n", nums.toString());
100           }
101           if (nums.size() != 1) {
102               System.out.println("| The stack has not been emptied.
    There are too     |");
103               System.out.println("| many operands in your expression.
                      |");
104               exceptionRaised = true;
105           }
106
107       } catch (IndexOutOfBoundsException ioobe) {
108           System.out.printf("| %-49s |\n", ioobe.getMessage());
109           exceptionRaised = true;
110       } catch (EmptyStackException ese) {
111           System.out.println("| Stack is Empty.
                     |");
112           exceptionRaised = true;
113       } catch (NumberFormatException nfe) {
114           System.out.println("| Your expression contained invalid
    characters.     |");
115           System.out.printf("| %-49s |\n", nfe.getMessage());
116           exceptionRaised = true;
117       } finally {
118           if (exceptionRaised) {
119               System.out.println("| Your expression is invalid.
    Evaluation failed.    |");
120               System.out.println("
                                                                    |
    ");
121               System.exit(1);
122           }
123       }
124       /*When the above iteration is complete, there should only be
125        * one item on the stack which is the answer.
126        */
```

```
127         System.out.println("
    |-------------------------------------------------|");
128         System.out.printf("| Answer of Evaluation = %-26s |\n", nums.pop
    ().toString());
129         System.out.println("

    ");
130      }
131 }
```

# Task 3

# Prime Numbers

## 3.1 Prime Number Checker using divisibility check

| Input | Expected Output | Actual Output |
|---|---|---|
| 1 | 1 is not a prime number | 1 is not a prime number |
| 2 | 2 is a prime number | 2 is a prime number |
| 3 | 3 is a prime number | 3 is a prime number |
| 4 | 4 is not a prime number | 4 is not a prime number |
| -2 | -2 is not a prime number | -2 is not a prime number |
| a 677 | 677 is a prime number | 677 is a prime number |
| 34939 | 34939 is a prime number | 34939 is a prime number |
| 188737204 | 188737204 is not a prime number | 188737204 is not a prime number |
| 381165334 | 381165334 is not a prime number | 381165334 is not a prime number |
| 947396057 | 947396057 is a prime number | 947396057 is a prime number |
| 9223372036854775807 | 9223372036854775807 is not a prime number | 9223372036854775807 is not a prime number |
| -9223372036854775808 | -9223372036854775808 is not a prime number | -9223372036854775808 is not a prime number |
| 9223372036854775808 | Your input was not accepted. Please restart the program and try again with a valid input. | Your input was not accepted. Please restart the program and try again with a valid input. |
| Test | Your input was not accepted. Please restart the program and try again with a valid input. | Your input was not accepted. Please restart the program and try again with a valid input. |

### 3.1.1 The source code for 3.1

```
import java.util.InputMismatchException;
import java.util.Scanner;

/**
 * @author Mark Said Camilleri
 * @version 20160511
 */

public class Question3a {

    public static void main(String args[]) {

        //Initialize Scanner object
        Scanner in = new Scanner(System.in).useDelimiter("\n");

        /*======================== WELCOME MESSAGE TO USER ========================*/
        System.out.println("-----------------------------------------------------");
        System.out.println("|    CSA 1017 - Data Structures and Algorithms 1    |");
        System.out.println("|---------------------------------------------------|");
        System.out.println("|         Submission by Mark Said Camilleri         |");
        System.out.println("|           Task 3.1: Prime Number Checker          |");
        System.out.println("|---------------------------------------------------|");
        System.out.print("| Please enter an integer to check: ");

        try {
            //The number to check whether or not it's prime
            long prime = in.nextLong();

            System.out.println("|---------------------------------------------------|");
            if (isPrime(prime))
                System.out.printf("| %31d is a prime number |\n", prime);
            else
                System.out.printf("| %27d is not a prime number |\n", prime);

            System.out.println("-----------------------------------------------------");
        } catch (InputMismatchException ime) {
            System.out.println("| Your input was not accepted. Please restart the   |");
            System.out.println("| program and try again with a valid input.         |");
            System.out.println("-----------------------------------------------------");
            System.exit(1);
```

```
41          }
42      }
43
44      /**
45       * This method checks the input parameter to see if it is a prime
46       * number or not.
47       *
48       * @param prime The number to check if it is prime
49       * @return true if the number is prime, false if it isn't
50       */
51      private static boolean isPrime(long prime) {
52          /*Firstly, if the number is 1, 0 or negative then it's not
53           * prime
54           */
55          if (prime <= 1) return false;
56              //If the number is 2, 2 is prime.
57          else if (prime == 2) return true;
58              //If the number is even and not 2, then it's not prime.
59          else if (prime % 2 == 0) return false;
60          /* Otherwise, divide this number by all odd numbers till the
61           * square root of the number. If one is divisible then it's
62           * not prime.
63           */
64          else {
65              double root = Math.sqrt(prime);
66              for (int i = 3; i <= root; i += 2) {
67                  if (prime % i == 0) return false;
68              }
69              return true;
70          }
71      }
72  }
```

## 3.2 Prime Number Checker using The Sieve of Eratosthenes

For this an `int` variable had to be used, due to it being the length of an array. As a result the range of numbers that can be checked was smaller.

| Input | Expected Output | Actual Output |
|---|---|---|
| 1 | 1 is not a prime number | 1 is not a prime number |
| 2 | 2 is a prime number | 2 is a prime number |
| 3 | 3 is a prime number | 3 is a prime number |
| 4 | 4 is not a prime number | 4 is not a prime number |
| -2 | Sieve of Eratosthenes only works on +ve integers. Please restart the program and try again with a valid input | Sieve of Eratosthenes only works on +ve integers. Please restart the program and try again with a valid input |
| 677 | 677 is a prime number | 677 is a prime number |
| 34939 | 34939 is a prime number | 34939 is a prime number |
| 188737204 | 188737204 is not a prime number | 188737204 is not a prime number |
| 381165334 | 381165334 is not a prime number | 381165334 is not a prime number |
| 947396057 | 947396057 is a prime number | 947396057 is a prime number |
| 2147483646 | 2147483646 is not a prime number | Requested array size exceeds VM limit Please restart the program and try again with a smaller input[1] |
| -2147483646 | Sieve of Eratosthenes only works on +ve integers. Please restart the program and try again with a valid input | Sieve of Eratosthenes only works on +ve integers. Please restart the program and try again with a valid input |
| 9223372036854775807 | Your input was not accepted. Please restart the program and try again with a valid input. | Your input was not accepted. Please restart the program and try again with a valid input. |
| Test | Your input was not accepted. Please restart the program and try again with a valid input. | Your input was not accepted. Please restart the program and try again with a valid input. |

[1] This error depends on the amount of memory allocated for the program

### 3.2.1 The source code for 3.2

```
import java.util.Arrays;
import java.util.InputMismatchException;
import java.util.Scanner;

/**
 * @author Mark Said Camilleri
 * @version 20160511
 */
public class Question3b {

    public static void main(String args[]) {

        //Initialize Scanner object
        Scanner in = new Scanner(System.in).useDelimiter("\n");

        /*======================== WELCOME MESSAGE TO USER ========================*/
        System.out.println("-------------------------------------------------------");
        System.out.println("|    CSA 1017 - Data Structures and Algorithms 1    |");
        System.out.println("|-----------------------------------------------------|");
        System.out.println("|         Submission by Mark Said Camilleri           |");
        System.out.println("|          Task 3.2: Sieve of Eratosthenes            |");
        System.out.println("|-----------------------------------------------------|");
        System.out.print("| Please enter a positive integer to check: ");
        try {
        /*
         * Value to check if it's prime. Note that this is now an int
         * since the Sieve or Eratosthenes requires an array of this
         * amount of elements.
         */
            int prime = in.nextInt();

            System.out.println("|-----------------------------------------------------|");
            if (sieve(prime))
                System.out.printf("| %31d is a prime number |\n", prime);
            else
                System.out.printf("| %27d is not a prime number |\n", prime);
            System.out.println("-------------------------------------------------------");
        } catch (InputMismatchException ime) {
            System.out.println("| Your input was not accepted. Please restart the   |");
            System.out.println("| program and try again with a valid input.         |");
```

```
41              System.out.println("
                                                                    ");
42              System.exit(1);
43          } catch (IllegalArgumentException iae) {
44              System.out.printf("| %-49s |\n", iae.getMessage());
45              System.out.println("| Please restart the program and try
                again with a   |");
46              System.out.println("| valid input.
                    |");
47              System.out.println("
                                                                    ");
48              System.exit(1);
49          } catch (OutOfMemoryError oome) {
50              System.out.printf("| %-49s |\n", oome.getMessage());
51              System.out.println("| Please restart the program and try
                again with a   |");
52              System.out.println("| smaller input.
                    |");
53              System.out.println("
                                                                    ");
54              System.exit(1);
55          }
56      }
57
58      /**
59       * Runs the Sieve of Eratosthenes algorithm to check if the input
60       * is a prime number or not.
61       *
62       * @param prime The number to check if it is prime or not.
63       * @return true if the input is prime, false if it isn't.
64       * @throws IllegalArgumentException if the input is <= 0
65       */
66      private static boolean sieve(int prime) throws
                IllegalArgumentException, OutOfMemoryError {
67          //Checks whether the input is valid or not
68          if (prime <= 0)
69              //Throws exception if input is not valid.
70              throw new IllegalArgumentException("Sieve of Eratosthenes
                only works on +ve integers.");
71          else {
72              //Each ith element is true if i-1 is prime. False otherwise.
73              boolean[] nos = new boolean[prime];
74
75              //We begin by assuming all the values are prime.
76              Arrays.fill(nos, true);
77
78              //Then, 1 is crossed out since it is a square.
79              nos[0] = false;
80
81          /*
82           * After which each value is checked. i is assumed to be prime
83           * unless marked as not prime. Multiples of i are not prime.
84           * Therefore all the multiples of i are marked as not prime.
85           * This keeps going on until the inputted value is set to not
86           * prime or reached and still set to prime.
```

```
87          */
88              for (int i = 2; i <= prime; i++) {
89                  //Checks if i is marked as prime.
90                  if (nos[i - 1]) {
91                      //If it is, all multiples of i are marked as not
    prime
92                      for (int j = i + i; j <= prime; j += i) {
93                          nos[j - 1] = false;
94                      }
95                  }
96              /*
97               * If the inputted value is marked as prime, we can stop
98               * there since we found out whether it's prime or not.
99               * Otherwise, this program keeps going on till the end
100              * (i = prime).
101              */
102                 if (!nos[prime - 1]) break;
103             }
104
105             return nos[prime - 1];
106         }
107     }
108 }
```

# Task 4

# Shell Sort

Since the requirement of this task was for the program to generate an array containing 16,384 elements, all of which are random numbers, and sort them, it is impossible to write the contents of the array for a reliable amount of tests and keep this document to a reasonable size. As a result, the program itself checks whether the array that has been sorted, is in fact sorted. This is done using the algorithm `checkSortedAscending` as given in Algorithm 1 (also included in the source code).

```
1 Algorithm: Check if sorted in ascending order
2 matrices Input: int array
  Output: boolean
3 for i = 0... length of array - 1 do
4 |   if element at i > element at i+1 then
5 |   |   return false
6 |   end if
7 end for
8 return true
```

**Algorithm 1:** checkSortedAscending(int[] array)

## 4.1 The source code for Task 4

```
1  import java.util.Arrays;
2
3  /**
4   * @author Mark Said Camilleri
5   * @version 20160516
6   */
7  public class Question4 {
8
9      public static final int SIZE = 16384;
10
11     public static void main(String[] args) {
12
13         /*======================== WELCOME MESSAGE TO USER
       ========================*/
14         System.out.println("
       ------------------------------------------------------");
15         System.out.println("|    CSA 1017 - Data Structures and
       Algorithms 1    |");
16         System.out.println("
       |-----------------------------------------------------|");
17         System.out.println("|         Submission by Mark Said Camilleri
               |");
18         System.out.println("|          Task 4: Shell Sorting Algorithm
               |");
19         System.out.println("
       |-----------------------------------------------------|");
20         System.out.println("| The array to be sorted is:
               |");
21
22         //The array, which  will eventially be sort
23         int[] arr = new int[SIZE];
24
25         //Populating the array with random data.
26         for (int i = 0; i < arr.length; i++) {
27             arr[i] = (int) (Math.random() * SIZE);
28         }
29
30         //Displays the unsorted array to the user.
31         System.out.println("Unsorted Array: " + Arrays.toString(arr));
32
33         //Sorts the array
34         int[] sorted = shellSort(arr);
35
36         //Displays the sorted array to the user
37         System.out.println("Sorted Array:   " + Arrays.toString(sorted))
       ;
38
39         //Displays whether the array is indeed sorted or not.
40         System.out.println("Array is sorted: " + checkSortedAscending(
       sorted));
41     }
42
43     /**
44      * Sorts the inputted array using the shell sort algorithm.
45      *
```

```
46      * @param unsorted The array to be sorted
47      * @return The sorted version of the array
48      */
49     private static int[] shellSort(int[] unsorted) {
50
51         /*
52          * First, the program begins my making a deep copy of the
53          * array to make sure the original one isn't affected.
54          */
55         int[] sorted = Arrays.copyOf(unsorted, SIZE);
56
57         /*
58          * This loop begins with the iterator being set at half the
59          * size of the array, - 1. This then iterates downwards by
60          * incrementing the iterator, dividing it by 2 and subtracting
61          * 1 from it, until it performs it's final iteration at a
62          * value of 1.
63          */
64         for (int sep = (unsorted.length / 2) - 1; sep > 0; sep = (sep +
    1) / 2 - 1) {
65             /*
66              * This for loop goes through the items between sep and
67              * the end of the array. To then use an bu sort
68              * algorithm to switch it
69              */
70             for (int i = sep; i < SIZE; i++) {
71                 /*
72                  * The algorthm begins by storing the value in a
73                  * temporary variable.
74                  */
75                 int temp = sorted[i];
76
77                 //iterator for next for loop. Used also outside loop.
78                 int j;
79                 /*
80                  * Finds good position of temp in steps of sep, sort
81                  * of like insertion sort.
82                  */
83                 for (j = i; j >= sep && sorted[j - sep] > temp; j -= sep
    ) {
84                     /*
85                      * Moves the (j-sep)th by a step of sep to
86                      * leave space for temp at it's proper place.
87                      */
88                     sorted[j] = sorted[j - sep];
89                 }
90                 //Once found temp is stored in it's proper place.
91                 sorted[j] = temp;
92             }
93         }
94         return sorted;
95
96     }
97
98     /**
99      * Method that checks whether the elements in the array in the
100     * parameter are all in ascending order.
101     *
```

```
102      * @param array The array to be checked.
103      * @return true if all elements are in ascending order, false
     otherwise.
104      */
105     public static boolean checkSortedAscending(int[] array) {
106 
107         /* Iterates through all elements except for the last one.
108          * If an unsorted element is found for loop stops.
109          */
110         for (int i = 0; i < array.length - 1; i++) {
111             /*
112              * If the ith element is larger than the i-1th element
113              * then the array is not sorted. Method ends returning
114              * false
115              */
116             if (array[i] > array[i + 1])
117                 return false;
118         }
119         /* If the condition is never satisfied, then the method returns
120          * true.
121          */
122         return true;
123     }
124 }
```

# Task 5

# Approximation of a Square Root

For this task, the Newton-Raphson algorithm was used, with the formula shown in Equation 5.2, which is basically a specific case of the general formula for the Newton-Raphson algorithm whcih is given in Equation 5.1.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{5.1}$$

$$x_{n+1} = x_n - \frac{x_n^2 - \text{input}}{2x_n} \tag{5.2}$$

| Input | Expected Output | Actual Output |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 1.414213562 | 1.414213562373095 |
| 4 | 2 | 2 |
| 100 | 10 | 10 |
| 1000 | 31.6227766 | 31.622776601683793 |
| 9999 | 99.99499987 | 99.99499987499375 |
| 29535083406040468068234 | $1.718577418 \times 10^{11}$ | $1.7185774176929147 \times 10^{11}$ |
| 45.3 | 6.730527468 | 6.730527468185536 |
| 5.6 | 2.366431913 | 2.3664319132398464 |
| 0.25 | 0.5 | 0.5 |
| 0.125 | 0.3535533906 | 0.35355339059327373 |
| -4 | Your input was not accepted since it was ≥ 0 Please enter a POSITIVE number to find its square root: | Your input was not accepted since it was ≥ 0 Please enter a POSITIVE number to find its square root: |
| 0 | Your input was not accepted since it was ≥ 0 Please enter a POSITIVE number to find its square root: | Your input was not accepted since it was ≥ 0 Please enter a POSITIVE number to find its square root: |
| Test | Your input was not accepted. Please try again: | Your input was not accepted. Please try again: |

## 5.1 The source code for Task 5

```
1  import java.util.InputMismatchException;
2  import java.util.Scanner;
3
4  /**
5   * @author Mark Said Camilleri
6   * @version 20160516
7   */
8  public class Question5 {
9      public static void main(String[] args) {
10
11         /*============================ WELCOME USER
       ================================*/
12         System.out.println("
       ------------------------------------------------------");
13         System.out.println("|    CSA 1017 - Data Structures and
       Algorithms 1    |");
14         System.out.println("
       |-----------------------------------------------------|");
15         System.out.println("|         Submission by Mark Said Camilleri
              |");
16         System.out.println("|       Task 5: Approximation of a Square
       Root       |");
17         System.out.println("
       |-----------------------------------------------------|");
18         System.out.print("| Please enter a positive number to find its
       square root: ");
19
20         //variable to state whether input should be accepted or not.
21         boolean isWrong = false;
22
23         //input variable. This shall be x1 and the first value of xn,
24         double input = 1;
25         do {
26             try {
27                 //accpets user input.
28                 input = new Scanner(System.in).nextDouble();
29
30                 /*
31                  * If the value is not positive, then it will raise an
32                  * error
33                  */
34                 if (input <= 0) {
35                     isWrong = true;
36                     System.out.println("| Your input was not accepted
       since it was \u2264 0");
37                     System.out.print("| Please enter a POSITIVE number
       to find its square root: ");
38                 }
39
40                 //It will also raise an error if it's not a number.
41             } catch (InputMismatchException imme) {
42                 isWrong = true;
43                 System.out.print("| Your input was not accepted. Please
       try again: ");
44             }
```

```
45              //This is repeated until a correct value was inputted.
46          } while (isWrong);
47
48          //Setting the first value for x_n
49          double curr_x = Math.random() * input + 1;
50          //Initializing x_(n-1)
51          double prev_x;
52
53          //Counts the number of steps taken.
54          int steps = 0;
55
56          do {
57              steps++;
58              prev_x = curr_x;
59
60              //Newton-Raphson formula on y=sqrt(x)
61              curr_x = prev_x - (Math.pow(prev_x, 2) - input) / (2 *
    prev_x);
62              /*
63               * This will keep on, either until there has been no change
64               * or until it has done Integer.MAX_VALUE number of steps.
65               */
66          }
67          while (curr_x != prev_x && curr_x != Double.NaN && steps <
    Integer.MAX_VALUE);
68
69          System.out.println("
    |-----------------------------------------------------|");
70
71          /*
72           * If the value is not the same as the sqrt computed by the
73           * built-in sqrt method, then it's approximately equal to,
74           * which is denoted by \u2248.
75           */
76          char sign = (curr_x == Math.sqrt(input)) ? '=' : '\u2248';
77
78          System.out.println("The square root of " + input + " " + sign +
    " " + curr_x);
79          System.out.println("This was done in " + steps + " steps");
80      }
81  }
```

# Task 6

# Matrix Multiplication

The methods used in this programme were created to work for any $n \times n$ matrices, with $n$ being specified by a constant in the program called `SIZE`. Since writing the tests for $32 \times 32$ matrices in this document would be too cumbersome, $3 \times 3$ matrices shall be tested instead.

It is worth noting that the matrices used to test the output are the matrices outputted by the program. Since the program rounds each matrix (matrices **A**, **B** and **AB**) to 2 d.p. (and the expected output is also being rounded to 2 d.p) the expected output and the actual output will not be exactly the same, but they should be approximately equal to each other.

| **A** | **B** | Expected **AB** | Actual **AB** |
|---|---|---|---|
| $\begin{bmatrix} 6.72 & 1.99 & 1.73 \\ 3.85 & 0.00 & 3.82 \\ 2.02 & 8.81 & -2.13 \end{bmatrix}$ | $\begin{bmatrix} -1.92 & 2.21 & -2.42 \\ -0.18 & -5.64 & 0.03 \\ -0.70 & -3.86 & -6.60 \end{bmatrix}$ | $\begin{bmatrix} -14.47 & -3.05 & -27.62 \\ -10.07 & -6.24 & -34.53 \\ -3.97 & -37.00 & 9.43 \end{bmatrix}$ | $\begin{bmatrix} -14.48 & -3.01 & -27.66 \\ -10.07 & -6.24 & -34.60 \\ -3.96 & -36.99 & 9.43 \end{bmatrix}$ |
| $\begin{bmatrix} 0.98 & 8.13 & -1.21 \\ 2.36 & -3.08 & -3.51 \\ 9.54 & -7.90 & -8.48 \end{bmatrix}$ | $\begin{bmatrix} 3.67 & -0.20 & -7.32 \\ -4.25 & -7.69 & 9.94 \\ 8.91 & 3.91 & -0.75 \end{bmatrix}$ | $\begin{bmatrix} -41.74 & -67.45 & 74.55 \\ -9.52 & 9.49 & -45.26 \\ -6.97 & 25.69 & -142.00 \end{bmatrix}$ | $\begin{bmatrix} -41.75 & -67.48 & 74.53 \\ -9.47 & 9.52 & -45.27 \\ -6.97 & 25.64 & -141.96 \end{bmatrix}$ |
| $\begin{bmatrix} -5.62 & 0.99 & 4.84 \\ 3.69 & 2.17 & 9.55 \\ -5.38 & -3.36 & -6.33 \end{bmatrix}$ | $\begin{bmatrix} -6.78 & 6.98 & 3.41 \\ 8.55 & 3.79 & -1.39 \\ -8.44 & -1.03 & 4.98 \end{bmatrix}$ | $\begin{bmatrix} 5.72 & -40.46 & 3.56 \\ 25.36 & 76.56 & 77.40 \\ 49.00 & 137.66 & 149.57 \end{bmatrix}$ | $\begin{bmatrix} 5.79 & -40.48 & 3.51 \\ -87.04 & 24.11 & 57.07 \\ 61.22 & -43.76 & -45.17 \end{bmatrix}$ |
| $\begin{bmatrix} -9.31 & 8.19 & 2.23 \\ -2.87 & 7.49 & 1.03 \\ 9.20 & -5.98 & 7.96 \end{bmatrix}$ | $\begin{bmatrix} -1.38 & -4.24 & -7.42 \\ 0.23 & 1.28 & 9.99 \\ 8.33 & 5.48 & 6.01 \end{bmatrix}$ | $\begin{bmatrix} 33.31 & 62.18 & 164.30 \\ 14.26 & 27.40 & 102.31 \\ 52.24 & -3.04 & -80.16 \end{bmatrix}$ | $\begin{bmatrix} 33.29 & 62.25 & 164.22 \\ 14.25 & 27.42 & 102.29 \\ 52.21 & -3.09 & -80.15 \end{bmatrix}$ |
| $\begin{bmatrix} -6.14 & 7.85 & 6.00 \\ -8.71 & 3.36 & -1.96 \\ -4.86 & 8.48 & 4.84 \end{bmatrix}$ | $\begin{bmatrix} -0.47 & -2.13 & 3.90 \\ -4.35 & -5.06 & 6.65 \\ -3.94 & -2.29 & -8.39 \end{bmatrix}$ | $\begin{bmatrix} -54.90 & -40.38 & -22.08 \\ -2.80 & 6.04 & 4.82 \\ -53.67 & -43.64 & -3.17 \end{bmatrix}$ | $\begin{bmatrix} -54.91 & -40.38 & -22.12 \\ -2.77 & 6.07 & 4.74 \\ -53.71 & -43.67 & -3.19 \end{bmatrix}$ |
| $\begin{bmatrix} -4.13 & 4.79 & -8.23 \\ -6.02 & 0.78 & -1.33 \\ -5.98 & -2.59 & 5.39 \end{bmatrix}$ | $\begin{bmatrix} 6.72 & 5.18 & 0.65 \\ 7.19 & 6.84 & 2.82 \\ 6.72 & 8.95 & -2.87 \end{bmatrix}$ | $\begin{bmatrix} -48.62 & -62.29 & 34.44 \\ -43.78 & -37.75 & 2.10 \\ -22.59 & -0.45 & -26.66 \end{bmatrix}$ | $\begin{bmatrix} -48.66 & -62.29 & 34.42 \\ -43.76 & -37.74 & 2.07 \\ -22.59 & -0.51 & -26.67 \end{bmatrix}$ |

## 6.1 The source code for Task 6

```
/**
 * @author Mark Said Camilleri
 * @version 20160517
 */
public class Question6 {

    /**
     * Stores the size of the matricie
     */
    public static final int SIZE = 3;

    public static void main(String[] args) {

        /*============================ WELCOME USER
    ================================*/
        System.out.println("
    ------------------------------------------------------");
        System.out.println("|    CSA 1017 - Data Structures and
    Algorithms 1    |");
        System.out.println("
    |-----------------------------------------------|");
        System.out.println("|         Submission by Mark Said Camilleri
            |");
        System.out.println("|           Task 6: Matrix Multiplication
            |");
        System.out.println("
    |-----------------------------------------------|");

        /*
         * The 2 matricies. Matrix A will be multuplied with matrix B
         * to from matrix AB.
         */
        double[][] matrixA = new double[SIZE][SIZE];
        double[][] matrixB = new double[SIZE][SIZE];

        //Put random values from 1 to 10 in matrixA
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                matrixA[i][j] = Math.random() * 20 - 10;
            }
        }

        //Put random values from 1 to 10 in matrixB
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                matrixB[i][j] = Math.random() * 20 - 10;
            }
        }

        //Multiplied matrix
        double[][] result = new double[SIZE][SIZE];


       /*
        * Transposes MatrixB. This is done to make the multiplication
```

```
49          * easier since the method would accept 2 arrays, one for the
50          * row and the other for the column.
51          */
52          double[][] matBtranspose = matTranspose(matrixB);
53
54         /*
55          * Multiplies each row of Matrix A with each column of Matrix B
56          * and adds them up to get the value of each cell.
57          */
58
59          for (int i = 0; i < SIZE; i++) {
60              for (int j = 0; j < SIZE; j++) {
61                  result[i][j] = cellMult(matrixA[i], matBtranspose[j]);
62              }
63          }
64          //Outputs Matrix A
65          System.out.print("Matrix A = ");
66          displayMatrix(matrixA);
67
68
69          System.out.println();
70          System.out.print("Matrix B = ");
71          displayMatrix(matrixB);
72
73          System.out.println();
74          System.out.print("  A * B =  ");
75          displayMatrix(result);
76
77      }
78
79      /**
80       * Performs a matrix multiplication for 1 cell only.
81       *
82       * @param row The row of the first matrix to be multiplied
83       * @param col The colum of the second matrix to be multiplied
84       * @return The value of the cell that resides in both the row and
    column that were inputted, for the resultant matrix.
85       */
86      public static double cellMult(double[] row, double[] col) {
87          //Result to be returned when finished.
88          double result = 0;
89
90          for (int i = 0; i < SIZE; i++) {
91              result += (row[i] * col[i]);
92          }
93
94          return result;
95      }
96
97      /**
98       * Transposes a square matrix of n x n size
99       *
100      * @param matrix The matrix to be transposed
101      * @return matrix transposed
102      */
103     public static double[][] matTranspose(double[][] matrix) {
104
105         double[][] transposed = new double[matrix.length][matrix[0].
```

```
    length];

        for (int i = 0; i < transposed.length; i++) {
            for (int j = 0; j < transposed[0].length; j++) {
                transposed[i][j] = matrix[j][i];
            }
        }

        return transposed;
    }

    /**
     * Displays the matrix to the user.
     * @param matrix The matrix to be displayed to the user.
     */
    public static void displayMatrix(double[][] matrix){
        for (int i = 0; i < matrix.length; i++) {

            for (int j = 0; j < matrix[i].length; j++) {
                System.out.printf("%7.2f ", matrix[i][j]);
            }
            System.out.println();
            System.out.print("           ");
        }
    }
}
```

# Task 7

# Task 8

# Task 9