Programming Practice for Exam 2

Write a complete program that uses one and two dimensional arrays to store gradebook values for 10 students. Your program should find the average for each student,the average for each test and the average for the class.

I have provided you with the main method, that includes the method calls to produce the gradebook table.

import java.util.Arrays;

public class Gradebook{

public static void main(String [] args){

int [][] gradesArray = {{87,96,70},

{68,87,90},

{94,100,90},

{100,81,82},

{83,65,85},

{78,87,65},

{85,75,83},

{91,94,100},

{76,72,84},

{87,93,73}};

double [] averageRowArray = averageStudent(gradesArray);

double [] averageColArray = averageTestScore(gradesArray);

double totalAverage = getClassAverage(gradesArray);

printGrades(gradesArray,averageRowArray,averageColArray,totalAverage);

}

averageStudent will return a one-dimensional array with the average value for each student.

averageTestScore will return a one-dimensional array with the average for each exam.

getClassAverage will return a double that represents the class average.

printGrades will print the Final result table as seen below.

Try to format the output by using the printf command and the “\t” notation that represents tabs.

Final Result Table:

Test 1 Test 2 Test 3 Average

Student 1 87 96 70 84.33

Student 2 68 87 90 81.67

Student 3 94 100 90 94.67

Student 4 100 81 82 87.67

Student 5 83 65 85 77.67

Student 6 78 87 65 76.67

Student 7 85 75 83 81.00

Student 8 91 94 100 95.00

Student 9 76 72 84 77.33

Student 10 87 93 73 84.33

Average 84.90 85.00 82.20 84.03

Comprehension 1:

Using the two-dimensional array in the gradebook program, write code that will print the memory reference of each row.

Comprehension 2:

Using the gradebook program, show the stack call and heap memory after the main method is invoked.

Comprehension 3:

Using the gradesArray defined in the main method of the GradeBook program. Write a method that takes in the gradesArray and sorts each row in ascending order using the selection sort algorithm.

Output:

70 87 96

68 87 90

90 94 100

Etc.

Comprehension 4:

Using the sorted array below, state the number of comparisons made when using the binary search to determine if the value was found or not found. To get partial credit if your answer is incorrect, outline how you determined the number of comparisons. **Do not submit code.**

**Target: -25, return -1 because not in it**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| -5 | -15 | -28 | -35 | -42 | -79 |

Comprehension 5:

Show the state of the array with each iteration (comparison) of insertion sort to sort the array in ascending order. Note, you should begin before the first iteration, and state what the comparison was that was made and what the array will look like based on said comparison. **Do not submit code.**

**(insertion sort, swap each element with each other in order)**

**2 0 1 3 9**

**0 2 1 3 9**

**0 1 2 3 9**

    {

**int** n = arr.length;

**for** (**int** i = 1; i < n; ++i) {

**int** key = arr[i];

**int** j = i - 1;

            /\* Move elements of arr[0..i-1], that are

               greater than key, to one position ahead

               of their current position \*/

**while** (j >= 0 && arr[j] > key) {

                arr[j + 1] = arr[j];

                j = j - 1;

            }

            arr[j + 1] = key;

        }

    }

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 90 | 100 | 40 | 5 | 0 | 30 |

Compare 100 to 90

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 90 | 100 | 40 | 5 | 0 | 30 |

Compare 40 to 100

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 90 | 40 | 100 | 5 | 0 | 30 |

Compare 40 to 90

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 40 | 90 | 100 | 5 | 0 | 30 |

Compare 100 to 90

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 40 | 90 | 5 | 100 | 0 | 30 |

Compare 100 to 5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 40 | 5 | 90 | 100 | 0 | 30 |

compare 90 to 5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | 40 | 90 | 100 | 0 | 30 |

Compare 40 to 5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | 40 | 90 | 0 | 100 | 30 |

Compare 100 to 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | 40 | 0 | 90 | 100 | 30 |

Compare 90 to 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | 0 | 40 | 90 | 100 | 30 |

Compare 40 to 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 5 | 40 | 90 | 100 | 30 |

Compare 5 to 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 5 | 40 | 90 | 30 | 100 |

compare 100 to 30

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 5 | 40 | 30 | 90 | 100 |

Compare 90 to 30

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 5 | 30 | 40 | 90 | 100 |

Compare 40 to 30

Count: 14

Comprehension 6:

Show the state of the array with each iteration of the selection sort. Use the example above when showing the state of the array after each comparison. **Do not submit code.**

**Selection sort find smallest num, swap with that index**

**3 9 0 2 1 5**

**0 9 3 2 1 5**

**0 1 3 2 9 5**

**0 1 2 3 9 5**

**0 1 2 3 5 9**

    {

**int** n = arr.length;

        // One by one move boundary of unsorted subarray

**for** (**int** i = 0; i < n-1; i++)

        {

            // Find the minimum element in unsorted array

**int** min\_idx = i;

**for** (**int** j = i+1; j < n; j++)

**if** (arr[j] < arr[min\_idx])

                    min\_idx = j;

            // Swap the found minimum element with the first

            // element

**int** temp = arr[min\_idx];

            arr[min\_idx] = arr[i];

            arr[i] = temp;

        }

    }

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 90 | 100 | 40 | 5 | 0 | 30 |
| 0 | 100 | 40 | 5 | 90 | 30 |
| 0 | 5 | 40 | 100 | 90 | 30 |
| 0 | 5 | 30 | 40 | 90 | 100 |