Basis of Computer Programming (Java A)
Lab Exercise 7

[Experimental Objective]

- Review some intrigue materials we introduced before by illustrating Lab 5.6 and 6.5
- Reveal some hints for Homework Assignment 3 questions.

[Exercises]

1. Writing a program that prompts the user to enter n ($0<n<10^4$) integers ($0<integer<10^5$)) in ascending order. We want the average value of combinations of picking two numbers from these n integers should larger than the average value of all these integers, and count how many couples satisfy the condition. (Please try to design your program to accomplish this question as fast as possible) you can use current2-current1 to test how many time your program execute this algorithm.

```
long current1=System.currentTimeMillis();
/* your algorithm */
long current2=System.currentTimeMillis();
System.out.printf("your program using %.3f
second",(current2-current1)/1000.0d);
```

```
Enter how many numbers: 5
Enter 5 numbers:
1 2 3 4 5
average=3.0
The number of these couple is 4
your program using 0.004 second

Enter how many numbers: 30
Enter 30 numbers:
2 3 5 6 9 10 12 13 15 16 23 55 66 77 89
101 220 221 222 255 277 280 290 300 303
400 420 455 500 520
average=172.16666666666666
The number of these couple is 194
your program using 0.004 second
```

A possible solution:
Step 1: before we program, we need to sort our thoughts on solving such problem, including the algorithm and data structure. Since we need to calculate the averages of the input numbers, we should design a data structure to store them first. And nothing could be better than an array for completing this task. The algorithm is as follows: (1) we need to calculate the average of all the input numbers; (2) we also need to calculate the averages of any pair of the input numbers and finds out whether it is larger than the average of all. To deal with (1), it is essential to conduct a for loop.

```java
import java.util.Scanner;

public class Lab5P6 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in); // Create a Scanner

        // Prompt the user to enter how many integers
        System.out.print("Enter how many numbers: ");
        int n=input.nextInt();
        int[] numbers = new int[n]; // Create an array of length ten

        System.out.printf("Enter %d numbers: ",n);
        // calculate the average value
        int total=0;
        for (int i = 0; i < n; i++){
            numbers[i] = input.nextInt();
            total+=numbers[i];
        }
        long current1=System.currentTimeMillis();
        double avg=total/(double)n;
        System.out.println("average="+avg);
        avg*=2;
        int count=0;
```

Please pay attention to why we double the “avg” here (it would be useful for the subsequent code snippets).

Step 2: To deal with the algorithm part (2), it is essential to iteratively compute the averages of the paired input numbers under the entire input array that implies a nested for loop, i.e., the outer loop for the entire input array and the inner loop for the sub-arrays where the the averages of all the possible pairs can be iteratively computed.

```java
        // using 2-layer-for loop to find the pairs
        // simply search for each possible combination
        // not so fast, but still acceptable
        // time complexity O(n^2)
        for (int i = 0; i < numbers.length; i++) {
            for (int j = i+1; j < numbers.length; j++) {
                if ((numbers[i]+numbers[j])>avg){
                    count+=n-j;
                    break;
                }
            }
        }
        System.out.println("The number of these couple is "+ count);
        long current2=System.currentTimeMillis();
        System.out.printf("your program using %.3f second",(current2-current1)/1000.0d);
    }
}
```

This question seems to be correctly answered by the code above. However, is it a perfect answer?

Step 3 (bonus): When we design a program, we need to consider its computation efficiency, i.e., reduce the unnecessary computation in programs. Can you figure whether we have any redundant computation in the code above? If you can, how do we design our program to reduce the redundant computation? The following is a possible solution.

```
        // integrate the idea of binary search
        // rather faster than solution 1
        // time complexity O(nlogn)
        int left,right,mid,smallest;

        for(int i=0;i<numbers.length-1;i++){
            left=i+1;
            right=n-1;
            // aim: find the smallest number that satisifies
            // numbers[i]+numbers[smallest]>avg*2, numbers[i]+numbers[smallest-1]<avg*2
            smallest=n;

            while(left<=right){
                mid=(left+right)/2;
                if((numbers[i]+numbers[mid])>avg){
                    right=mid-1;
                    smallest=mid;
                }
                else{
                    left=mid+1;
                }
            }
            count+=(n-smallest);
        }
        System.out.println("The number of these couple is "+ count);
        long current2=System.currentTimeMillis();
        System.out.printf("your program using %.3f second",(current2-current1)/1000.0d);
    }
```

Can you figure how the code above improve the computational efficiency of step 2? Is it the perfect answer already? Can you do something more to improve it?

2. Lab6.5: Sudoku is a famous mathematical game in which players fill numbers 1-9 in a 9 × 9 square. The square satisfies that every row and every column contain 1-9 only once. Specially, the square is divided into 9 sub-squares (as shown below), and every sub-squares also contains 1-9 only once. Following is a valid Sudoku square (Notice that the sub-squares are separated by red lines).

2	9	3	7	1	5	4	8	6
8	6	1	2	4	9	5	3	7
7	4	5	8	6	3	1	9	2
6	7	8	9	2	1	3	4	5
1	3	9	5	7	4	2	6	8
4	5	2	6	3	8	7	1	9
9	2	4	3	8	7	6	5	1
3	8	6	1	5	2	9	7	4
5	1	7	4	9	6	8	2	3

Write a program to judge whether a 9×9 square is a Sudoku square.

1) Get a 9×9 square from console
2) If it is a Sudoku square, print **"Yes"**
3) If it is not a Sudoku square, print **"No"**

Sample:

```
2 9 3 7 1 5 4 8 6
8 6 1 2 4 9 5 3 7
7 4 5 8 6 3 1 9 2
6 7 8 9 2 1 3 4 5
1 3 9 5 7 4 2 6 8
4 5 2 6 3 8 7 1 9
9 2 4 3 8 7 6 5 1
3 8 6 1 5 2 9 7 4
5 1 7 4 9 6 8 2 3
Yes
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
No
1 2 3 1 2 3 1 2 3
4 5 6 4 5 6 4 5 6
7 8 9 7 8 9 7 8 9
1 2 3 1 2 3 1 2 3
4 5 6 4 5 6 4 5 6
7 8 9 7 8 9 7 8 9
1 2 3 1 2 3 1 2 3
4 5 6 4 5 6 4 5 6
7 8 9 7 8 9 7 8 9
No
```

A possible solution:

Step 1: We should initialize our variables for the entire program, i.e., by declaring “T” to store the inputting number of test cases (matrices) and “sudoku” to store each matrix

```
import java.util.Scanner;
public class Tutorial65 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int T = in.nextInt();    //get the number of test cases
        int[][] sudoku = new int[9][9];
```

Step 2: We need to check whether each matrix is a sudoku matrix. Accordingly, we can design an outer loop. To check each matrix, first, we need to find out whether the matrix is valid or not, i.e., whether the numbers in the matrix are between 1 and 9. Note that since matrix is essentially a two-dimensional array, we can conduct such checking process based on a nested for loop.

```
Outer:
while (T-- > 0) {
    boolean isValid = true;

    //Get input and check if there are invalid numbers
    for (int r = 0; r < 9; r++) {
        for (int c = 0; c < 9; c++) {
            sudoku[r][c] = in.nextInt();
            if(sudoku[r][c] > 9 || sudoku[r][c] < 1) {
                System.out.println("No");//isValid = false;
                continue Outer;
            }
        }
    }
```

Here, we have some specific tricks: (1) we tokenize the while loop as “Outer” and use that in the “continue” statement. You can try to think about what that can help with; (2) note that we declare a boolean variable “isValid” in the while loop. You can keep that in mind and dig what is used for in the following code snippets.

Step 3: We need to figure how to determine a matrix is a sudoku matrix. From the problem description, we can derive that a sudoku matrix has two features: (1) in each row and column, there should not be any repeated number; (2) in each 3*3 sub-square, there should not be any repeated number either. Therefore, we can figure that we might want to design two corresponding checking mechanisms, each of which should be designed in a loop manner. Moreover, the checking mechanisms can be designed as independent methods.

```
        //check if there are more than two same number in a row or a column
        for(int i = 0; i < 9; i++){
            if(!cheeckDuplicate(sudoku, i, 0) || !cheeckDuplicate(sudoku, i, 1)) {
                System.out.println("No");//isValid = false;
                continue Outer;
            }
        }

        //check if there are more than two same number in a sub-square
        for(int i=0;i<9;i=i+3){
            for(int j=0;j<9;j=j+3){
                if(!cheeckSub(sudoku,i,j)){
                    System.out.println("No");//isValid = false;
                    continue Outer;
                }
            }
        }

        System.out.println("Yes");
    }

}
```

Here, “checkDuplicate” and “checkSub” are the methods that help check the two features of sudoku matrix respectively.

Step 4: Design methods “checkDuplicate” and “checkSub”. The key for the design is to figure how to map your parameters to your method body and return values. In “checkDuplicate”, we want to check the validity of both the columns and rows. Therefore, it is essential to design a parameter that can be used as a switch of them. For each column or row, there are actually tons of ways to conduct the checking

process. We should try to design the most efficient ways, e.g., try to reduce the computation complexity.

```
/**
 * to check if rows or columns are duplicate
 *
 * @param sudoku sudoku you to be checked
 * @param k column or row index
 * @param rc 0 to check row, 1 to check column
 * @return if duplicate return false, else return true
 */
public static boolean cheeckDuplicate(int[][] sudoku, int k, int rc){
    int[] existNum = new int[9];
    for(int i = 0; i < 9; i++){
        int current = (rc == 0) ?  sudoku[k][i] : sudoku[i][k];
        existNum[current-1]++;
    }
    int product=1;
    for(int i = 0; i < 9; i++){
        product*=existNum[i];
    }
    if(product==1)
        return true;
    return false;
}
```

In “checkSub”, we should realize that a sub-matrix is also a two-dimensional array that implies we should still use a nested for loop to conduct the checking process.

```
    public static boolean cheeckSub(int[][] sudoku, int i, int j){
        int[] existNum = new int[9];
        for (int r = 0; r < 3; r++) {
            for (int c = 0; c < 3; c++) {
                existNum[sudoku[i+r][j+c]-1]++;
            }
        }
        int product=1;
        for(int k = 0; k < 9; k++){
            product*=existNum[k];
        }
        if(product==1)
            return true;
        return false;
    }
}
```

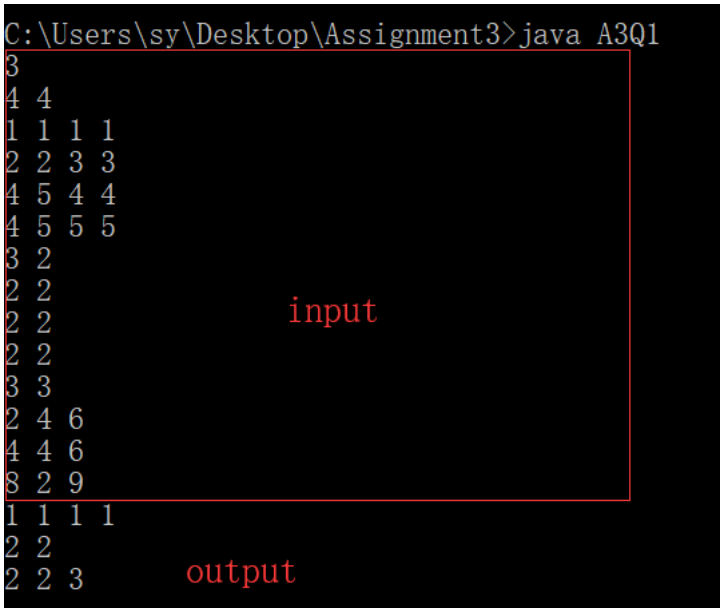
3. A3Q1: Greatest Common Divisor (GCD) [25 points]

Given an **m*****n** matrix (0<**m**<10^5, 0<**n**<10^5), and the elements of the matrix are all positive integers. Write a program to calculate and output **n** GCDs of all elements of **n** columns in the matrix.

Input:
The first line of input is an integer **s** representing the number of matrices (0<**s**<100), followed by **s** matrixes. Each matrix input form is: The first line is the matrix's **m** and **n**, followed by the **m**-line input, with **n** elements in each line.

Output:
Please output the **n** GCDs of each matrix in order by line.

Sample:

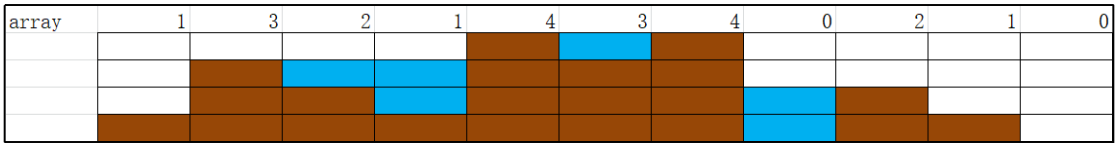


The hints:
Step 1: You should sort your thoughts in designing the corresponding algorithms and data structures. As the last question, you need to design a outer loop to deal with the total input matrices. Next, for each matrix, you need to figure out the GCD of them. Since we are dealing with matrix, it is essential to design two-dimensional arrays and the corresponding nested for loop to handle them. You need to understand the mechanism of how to derive GCD of matrices.

Step 2: Computing GCD is an independent and reusable process that can be totally designed as a method that is invoked by the main method. You should figure how to realize the logic of computing GCD, how to design the parameters and the returned values accordingly, and how to invoke it in your main method.

4. A3Q2: Water Storage [25 points]

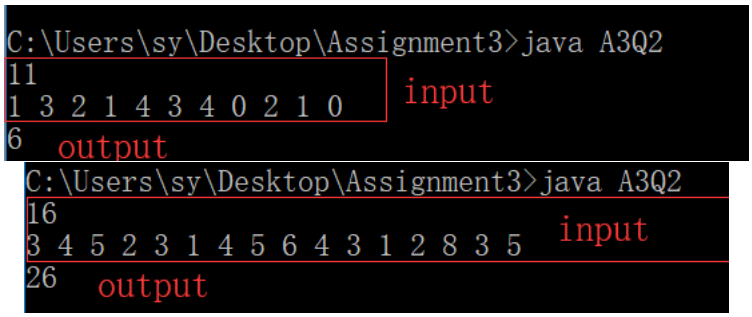
Given a one-dimensional array containing **n** positive integers (0<**n**<100), the number of the array represents the height of the column, and the depressed portion between the columns can store water. As the figure shown below, you can store 6 units of water. Write a program to calculate how many units of water an array can store.



Input:
The first line of input is an integer **n** representing the length of the array. The second line is **n** integers which are the elements of the array.

Output:
How many units of water the array can store.

Sample:



The hints:
Step 1: Sure you should sort your thoughts and find the essence of this question. An element of the array can store water when its left and right are both higher than itself. For instance, element 3 can store water because element 2 is 1 higher and element 4 is 3 higher.

Step 2: In addition, the exact water that one element can store also depends on some other factors. For instance, element 3 store 2 water because element 2 also stores 1 water.

Step 3: Therefore, for step 1, you might want to design your algorithm that can decide whether an element can store water by considering the impact from its left and right elements, e.g., iterating the array in both the forwarding and reversing order. For step 2, you can think about the possibility to design a variable that can represent the impact from the water stored by the adjacent elements and use that variable to help compute the water stored by the target element.

Step 4: As long as you are clear about the essence of this question, you do not have to be restricted in the steps above. Instead, you can totally combine them.

5. A3Q3: Spiral Matrix [25 points]

Given two integers **m**, **n** (0<**m**<10, 0<**n**<10), generate a **m***n** matrix filled with elements from 1 to **m***n** in anticlockwise spiral order, starting from the top right corner.

Input:
The input is two integers **m**, **n**.

Output:
Please output the **m***n** spiral matrix as the sample.

Sample:

```
C:\Users\sy\Desktop\Assignment3>java A3Q3
4 4  input
4  3  2  1
5  14 13 12
6  15 16 11      output
7  8  9  10

C:\Users\sy\Desktop\Assignment3>java A3Q3
5 3
3  2  1
4  13 12
5  14 11
6  15 10
7  8  9

C:\Users\sy\Desktop\Assignment3>java A3Q3
4 6
6  5  4  3  2  1
7  20 19 18 17 16
8  21 22 23 24 15
9  10 11 12 13 14
```

The hints:
Step 1: Well, it seems to be a harsh question, but as long as you can understand the patterns of the number movement in the matrix, it is just as easy as snapping your finger. So, having confidence and faith is our first suggestion. :)

Step 2: Now, observe the number movement. First, you place 1 in the right-up corner and determine the numbers place from right to left until they hit the row's bound. Next, you keep placing the numbers in a vertical manner until they hit the column's bound. Then, you keep placing the numbers in a horizontal manner until they hit the row's bound again. At last, you again place the numbers in a vertical manner until they meet 1. And this process iterates for the inner matrix. Therefore, you should realize this is a nested combination of loop and if statements.

Step 3: While it is easy to figure out how to start, you need to be cautious to find out when it should terminate. Eventually you should be able to nail it.

6. A3Q4: Sum of Sub-matrix [25 points]

Given an integer matrix of **m*n** (0<**m**<100, 0<**n**<100), and **T** queries. Each query includes two sets of coordinates: the upper left coordinate (**x1**, **y1**) and the lower right coordinate (**x2**, **y2**) (**x1** <= **x2**, **y1** <= **y2**). Please write a program to calculate the sum of all the integers in the sub-matrix determined by each set of coordinates.
As shown in the figure below, the coordinates (2, 1) (3, 3) determine the red sub-matrix. The sum of all the elements in this sub-matrix is 93

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Input:
The first line of input is two integers **m**, **n**, followed by m lines with **n** integers which represents the elements of the matrix. Then, there will be **T** lines of queries, and each line includes four integers **x1**, **y1**, **x2**, **y2**.

Output:
Please output **T** lines of result of the queries in order.

Sample:

```
C:\Users\sy\Desktop\Assignment3>java A3Q4
5 5
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25
5                    input
2 1 3 3
1 1 2 2
2 1 2 4
2 2 3 2
1 1 1 1
93
40
54
31
7
```

The hints:
Step 1: Now that you have taken the previous training, in this question you should be able to design the corresponding data structures of storing the inputs and the associated checking process in the beginning of your code.

Step 2: First and foremost, you should be able to tell the locations of the four corners by the input queries. The tricky part here is how to derive the right-up and left-down locations based on the given queries.

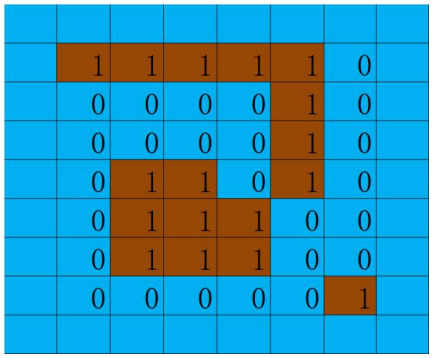
Step 3: Once you figure that out, the rest is routine. However, can you design the computing process as one or more methods such that they can be potentially reused in other scenarios?

7. A3Q5: The Biggest Island [50 points]

Given a matrix of **m*n** (0<**m**<100, 0<**n**<100). The matrix contains only **0** and **1**, where **0** represents the sea and **1** represents the land. Assuming that the outside of the matrix is all sea, please write a program to calculate the area of the largest island (i.e. how many **1**s are included).

Input:
The first line of input is two integers **m**, **n**, followed by m lines with **n 0/1**s which represents the elements of the matrix.

Output:
Please output the area of the largest island. As shown in the figure below, area of the biggest island is 8.



Sample:

```
C:\Users\sy\Desktop\Assignment3>java A3Q5
6 6
1 1 1 1 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 1 1 1 0 0
0 1 1 1 0 0
0 1 1 1 0 0
9 output

C:\Users\sy\Desktop\Assignment3>java A3Q5

6 6
1 1 1 1 1 0
0 0 0 0 1 0
1 1 1 1 1 0
1 0 0 0 0 0
1 1 1 1 1 1
0 0 0 0 0 0
18
```

input

```
C:\Users\sy\Desktop\Assignment3>java A3Q5
6 6
1 0 1 1 1 0
1 0 1 0 1 0
1 0 1 0 1 0
1 0 1 0 1 0
1 1 1 0 1 1
0 0 0 0 0 0
18

C:\Users\sy\Desktop\Assignment3>java A3Q5
6 6
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0
```

The hints:
Step 1: Still, you need to figure the exact pattern of how the numbers in the matrix are counted. The essence is that if an element is “1”, then you should look for whether its adjacent elements (row-wise and column-wise) are also “1”s. If you can find one, then based on that one, you should repeat the process of finding the adjacent “1”s until you cannot find any more.

Step 2: Now here comes the problem, how do you present such repetition of finding "1"s. It is a both hard and easy question. It is hard because we need to do it in a deterministic manner which we have not learned yet. It is easy because once you know that manner, you will soon finish your work, just like snapping your finger. The manner is called "recursion". Go to learn that by yourself given this is a bonus question.

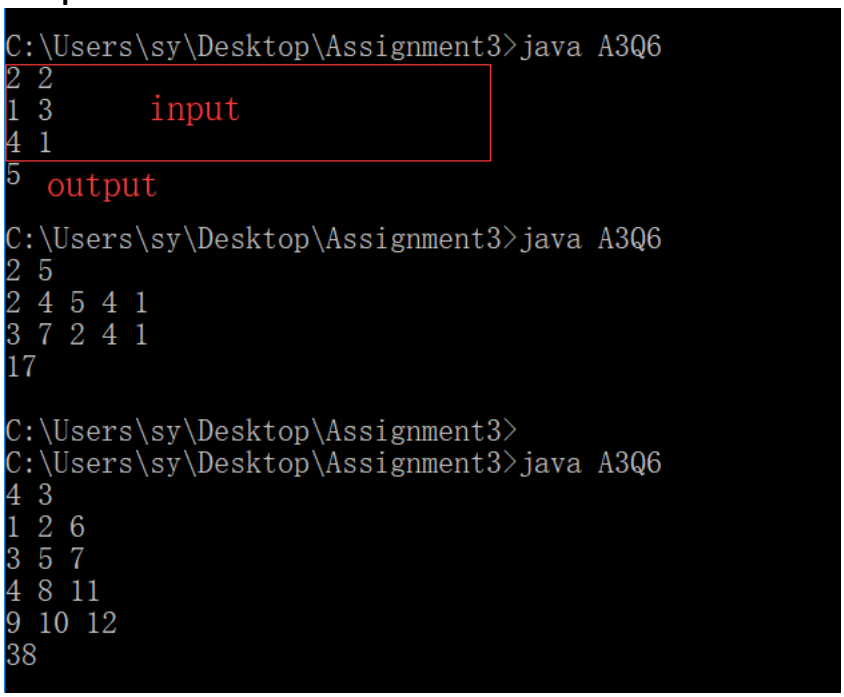
8. A3Q6: Shortest Path [50 points]

Given an integer matrix of **m*n** (0<**m**<10, 0<**n**<10). You need to go from the top left corner of the matrix to the bottom right corner, and you can only go right or down. The value of each point is the distance you need to complete if you go through this point. Write a program to find the shortest path and output the total distance of this path.

Input:
The first line of input is two integers **m**, **n**, followed by m lines with **n** integers which represents the elements of the matrix.

Output:
Please output the length of the shortest path.

Sample:



The hints:
Step 1: This question is somehow similar to the last one. Essentially, each time when you hit an element, you need to look either right or down at the adjacent elements and add the value to the path scores towards your target. You need to repeat this process for every possible elements in between and select the smallest path score.

Step 2: Here comes the same problem as the last one. How do you present such repetition? Is “recursion” still the solution you are looking for? Think about it.

P.S., Please refer to “Dijkstra algorithm” for a general case relevant to this question.