Section, Name and ID#

Problem 2: Write a Java method ***public static double[] lin(double[] data)*** that takes as its argument an array of data points ***double[] data***, and returns a two-element array – the first element being the slope of the linear regression and the second element being the intercept. The linear regression approximates the data points by the linear formula

$$y = k\,x + b,$$

where the slope k and the intercept b are computed as

$$k = \frac{\overline{xy} - \bar{x}\,\bar{y}}{\overline{x^2} - \bar{x}^2}, b = \bar{y} - k\,\bar{x}$$

Here $\bar{x}$ is the mean of the x coordinates, $\bar{y}$ is the mean of the y coordinates, $\overline{x^2}$ is the mean of the squares of the x coordinates, and $\overline{xy}$ is the mean of the products of the x and y coordinates. Use the element indices of the array ***double[] data*** as x coordinates and the element values as y coordinates. You may assume and use the method double ***mean(double[] a)***.

```
public static double [] lin (double [] data)
double [] data = [2, 3, 4]
double [] lin = new [2]
double k =

for (int i=0, i<data.length, i++) {
{double x = i;
 double y = data[i];
}

for (int j=0, j< lin.length, i++) {
double xy1 = mean( [x, y] );
double x1 = mean(x)
double y1 = mean(y)
double x2 = mean(x, x)
double k = (xy1 - x1*y1)/(x2 - sqrt(x1))
double b = (y1 - k*x1)
double [] lin = [k, b]
```

[k, b]
x = (0, 1) ?
y = (x, 0) ?

Use the backside, if needed

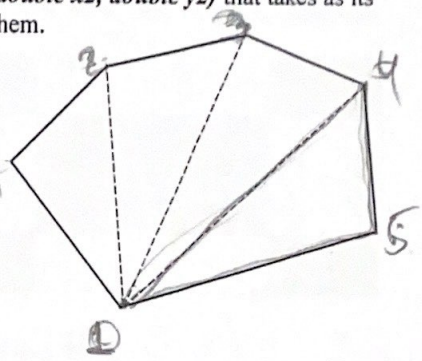
2

Section, Name and ID#

Problem 3: Write a Java function ***public static double area(double[][] vertex)*** that takes as its argument a ***2-by-n*** array of a convex polygon's vertex coordinates ***double[][] vertex*** – the ***x*** coordinates in the first row and ***y*** coordinates in the second row. It returns polygon's area as follows:

1. Divides the polygon into triangles by connecting the ***first*** vertex with the n^{th} and $(n+1)^{st}$ vertices;
2. Adds the areas of the constructed triangles using the formula $area = \sqrt{p(p-a)(p-b)(p-c)}$, where ***a***, ***b*** and ***c*** are the sides and $p = (a + b + c) / 2$.

You may assume and use a method ***double dist(double x1, double y1, double x2, double y2)*** that takes as its arguments coordinates of two points and returns the distance between them.




```
[2 - - - n]
[0 - - - n]
double[][] vertex = new...[2][n];
for(column=0, column
X=E
int[][]Point=[i][i]  for(int i=0, i<vertex.length, i++)
                       for(int j=0, j<vertex[i].length, j++
dist( double K= vertex.length
double p= K-1  for(int K=1, k≤n, K++)  ?
double a= dist(0,0, vertex[c]n] vertex[i], verte
double b= dist(0, 0, vertex[i+1], vertex[j+1])
double c dist( vertex[i], vertex[j], vertex[i+1], vertex[j
double p =(a+b+c)/2
double area = sqrroot(p*(p-a)*(p-b)*(p-c)
sum needed
double g= vertexE
```

1

Use the backside, if needed

OOP.MT-170317.L030

Section, Name and ID:

Problem 4: Write a Java method ***public static void magic4N(int[][] square)*** that creates a magic square of a *4N-by-4N* size using the following algorithm:

1. Creates an array of the same size as ***int[][] square*** and fills it forward with successive integers assigning *1* to the top-left element;
2. Creates anther array of the same size as ***int[][] square*** and fills it backward with successive integers assigning *1* to the bottom-right element;
3. Divides the original ***int[][] square*** into 16 blocks of the same size – 4 blocks per row and column. In the on-diagonal (shaded) blocks copies the elements from the first array, and in the off-diagonal blocks copies the elements from second array.

1	2					7	8
9	10					15	16
		19	20	21	22		
		27	28	29	30		
		35	36	37	38		
		43	44	45	46		
49	50					55	56
57	58					63	64

		62	61	60	59		
		54	53	52	51		
48	47					42	41
40	39					34	33
32	31					26	25
24	23					18	17
		14	13	12	11		
		6	5	4	3		

```
for (int i=0, i< magic4N.length, i++)
  for(int j=0, j<magic4N[i].length, j++)
int[0][0] number = 1
[i][j] + 1number
```

nice idea, wrong implementation

```
int[][] currentf= 1
currentf= currentf+1
```
} forward

```
int [][] currentb = 4N*4N
currentb = currentb- 1
```
(?) } backward

1

Use the backside, if needed

OOP.MT.170317.2030