Project 3: You will be implementing 2D 3x3 median filter and 2D

5x5 Gaussian filter.

What you have to do:

1) Project 3 will produce two result files, one for median, and one for Gaussian. Name them MedianResult and GaussResult, respectively.

2) Run your histogram program on MedianResult to get a histogram, name it MedianHist.

3) Run your BiMeanGauss program on MedianHist to get the best threshold value, name it MedianThr.

4) Apply your threshold program to the same input file of this project 3 using

MedianThr to get a binary image, call it MedianBinary.

5) Print MedianHist, MedianThr, and prettyPrint MedianBinary **as your project hard copies**.

6) Do the same for GaussResult: from 2) to 4) name them

GaussHist, GaussThr, and GaussBinary.

7) Print GaussHist, GaussThr, and prettyPrint GaussBinary **as your project hard copies.**

8) \*\*\* For you to see how the noise filter smooth the image,

Do the same for the original input image: from 2) to 4) name them

nonFilterHist, nonFilterThr, and prettyPrint nonFilterBinary **as your project hard copies.**

Note:

1) To make this project easier for you to implement, you do not need to frame the image, instead, for median filter, the operation begins at p(1, 1) and ends at p(numRows-2, numCols-2) and for Gaussian filter, the operations begins at p(2,2) and ends at p(numRows-3, numCols-3).

2) Follow the specs below, the project should be pretty easy to implement.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Language: Java

Due date: soft copy: 2/27/2019 Wednesday before midnight.

+2 pt for early submission: 2/25/2019 Monday before midnight.

-1 pt due: 2/28/2019 Thursday before midnight

After 2/28/2019 -12 pts for all students who did not submit soft copy

Due Date: Hard copy: 2/28/2019 Thursday in class,

-1 pt for late hard copy submission (place under door: A218).

All projects without hard copy will receive 0 pts even you have submit soft copy on time and even if it works.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

I. Input1 (args[0]): a txt file representing a grey-scale image with header.

Input2 (args[1]): a Gaussian template. Although we are using a 5x5 template,

it is more useful to allow a different Gaussian template, size and values.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

II. Output1(args[2]): The result of Median filter

Output2(args[3]): The result of Gaussian filter

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

III. Data structure:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

- an image class

- numRows (int)

- numCols (int)

- minVal (int)

- maxVal (int)

- newMin (int)

- newMax (int)

- TemplateRows (int)

- TemplateCols (int)

- totalWeight

- imgAry (int \*\*) // a 2D array of size numRows by numCols

// need to dynamically allocate at run time

- GaussTemplate (int \*\*) // a 2D array of size TemplateRows by

// TemplateCols, need to dynamically allocate at run time

- medianAry (int \*\*) // a 2D array, need to dynamically allocate

//at run time of size numRows by numCols

- GaussAry (int \*\*) a 2D array, need to dynamically allocate at run time

of size numRows by numCols

- neighborAry[9](int)

// 1 D array for storing the 3 x 3 neighbors of p(i,j) for sorting

- methods:

- constructor(s) // need to dynamically allocate all 2D arrays

- loadImage (...)

// read from inFile1 and load onto imgAry

// You should know how to do this method.

- loadTemplate (...)

// read from inFile2 and load onto templateAry

// You should know how to do this method.

- medianFilter(...) // See algorithm below

- loadNeighbors (i, j)

// load the 3 x 3 neighbors of given pixel(i,j)into neightAry

// You should know how to do this method.

- sort(neighborAry) // Use selection sort, up to 5th iterations

// You should know how to do this method.

- GaussianFilter(...) // Apply convolution with the Gaussian template

// for the entire image begins at p(2,2), ends p(numRows-3, numCols-3)

// See algorithm below.

- computeWeight (... )

// sum of the value in the template

// You should know how to do this method.

- int convolution (i, j) // apply convolution at pixel (i, j) with the given

// Gaussian template. The algorithm is given below.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

III. Main( )

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

step 0: inFile1 <-- open the input image file

inFile2 <-- open the template file

outFile1, outFile2 <-- open output files

numRows, numCols, minVal, maxVal <-- read from inFile1

TemplateRows, TemplateCols <-- read from inFile2

dynamically allocate all arrays, initialize all arrays to zero

step 1: loadImage (...)

step 2: loadTemplate (...)

step 3: medianFilter(...) // see algorithm below

step 4: output (numRows, numCols, newMin, newMax) to outFile1

step 5: output medianAry to outFile1

step 6: GaussianFilter(...) // see algorithm below

step 7: output (numRows, numCols, newMin, newMax) to outFile2

step 8: output GaussAry to outFile2

step 9: close all files

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IV. medianFilter(...)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

step 0: newMin <-- minVal

newMax <-- maxVal

step 1: process the imgAry, from left to right and top to bottom

using i, and j, begins at (1, 1) and ends at (numRows-2, numCols-2).

p(i,j) <-- next pixel

Step 2: loadNeighbors (i, j, neighborAry)

// load the 3 x 3 neighbors of p(i,j)into neighborAry

step 3: sort(neighborAry) // Use selection sort, up to 5th iterations

step 4: medianAry (i,j) <-- neighborAry[4]

if newMin > neighborAry[4]

newMin <-- neighborAry[4]

if newMax < neighborAry[4]

newMax <-- neighborAry[4]

step 5: repeat step 1 – step 4 until all pixels within bound are processed

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

V. GaussianFilter(...)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

step 0: newMin <-- minVal

newMax <-- maxVal

totalWeight <-- computeWeight (... )

// sum of all the values in the template

step 1: process the imgAry, from left to right and top to bottom

using i, and j, begins at (2, 2) and ends at (numRows-3, numCols-3).

p(i,j) <-- next pixel

Step 2: GaussAry (i,j) <-- convolution (i, j) / totalWeight  
  
        if newMin > GaussAry (i,j)  
            newMin <-- GaussAry (i,j)  
  
         if newMax < GaussAry (i,j)  
              newMax <-- GaussAry (i,j)

step 3: repeat step 1 – step 2 until all pixels within bound are processed

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

VI. int convolution (i, j)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

step 0: result <-- 0

iOffset <-- i – (TemplateRows / 2) // in our case 5/2 is 2)

jOffset <-- j - (TemplateCols / 2) // in our case 5/2 is 2)

Step 1: m <-- 0

step 2: n <-- 0

step 3: result += imgAry[iOffset + m, jOffset + n] \* GaussTemplate [m,n]

step 4: n++

step 5: repeat step 3 – step 4 while n < TemplateCols

step 6: result step 3 – step 5 while m < TemplateRows

step 7: return result