negative transformation:

```
int[][] result = new int[im[0].length][im.length];
for (int i = 0; i < im[0].length; i++) {
for (int j = 0; j < im.length; j++) {
result[i][j]=im[j][i];
} }
```

Image Rotation

```
int[][] result = new int[im.length][im[0].length];
for (int i = 100; i < im.length; i++) {
for (int j = 100; j < im[i].length; j++) {
result[i][j]=im[i-100][j-100];
} }
```

Image translation

```
int[][] im2 = readimage("C:\\Images\\img2.jpg");
int[][] im3 = readimage("C:\\Images\\img3.jpg");
int r = Math.min(im.length,Math.min(im2.length,im3.length));
int c = Math.min(im[0].length,Math.min(im2[0].length,im3[0].length));
int[][] result = new int[r][c];
for (int i = 0; i < r; i++) {
for (int j = 0; j < c; j++) {
result[i][j]=im[i][j]+im2[i][j]+im3[i][j];
} }
```

Arithmetic Mean Filter:

```
public static int[][] ArethmaticMeanFilter(int[][] im, int a,int b) {
int[][] newIm = new int[im.length][im[0].length];
for (int i = a / 2; i < im.length - a / 2; i++) {
for (int j = b / 2; j < im[i].length - b / 2; j++) {
int sum = 0;
for (int k = i - a / 2; k < i + 1 + a / 2; k++) {
for (int l = j - b / 2; l < j + 1 + b / 2; l++) {
sum += im[k][l];
} }
newIm[i][j] = sum / (a * b);
} }
return newIm; }
```

negative trasformation:

```
public static int[][] TransformNegative(int[][] im) {
int[][] new_im = new int[im.length][im[0].length];
int max = Integer.MIN_VALUE;
int min = Integer.MAX_VALUE;
for (int i = 0; i < im.length; i++) {
for (int j = 0; j < im[i].length; j++) {
new_im[i][j] = Negative(im[i][j]);
max = Math.max(max, new_im[i][j]);
min = Math.min(min, new_im[i][j]);
} }

for (int i = 0; i < new_im.length; i++) {
for (int j = 0; j < new_im[0].length; j++) {
new_im[i][j] = 255 * (new_im[i][j] - min) / (max - min);
} }
return new_im; }
```

Geometric Mean Filter:

```
public static int[][] geometricMean(int[][] im, int a, int b) {
int[][] newIm = new int[im.length][im[0].length];
double square = a * b;
for (int i = a / 2; i < im.length - a / 2; i++) {
for (int j = b; j < im[i].length - b / 2; j++) {
double product = 1;
for (int k = i - a / 2; k <= i + a / 2; k++) {
for (int l = j - b / 2; l <= j + b / 2; l++) {
product *= Math.pow(im[k][l], 1 / square);
} }
newIm[i][j] = (int) product;
} }
return newIm; }
```

Median Filter:

```
public static int[][] medianFilter(int[][] im, int a, int b) {
int[][] newIm = new int[im.length][im[0].length];
for (int i = a / 2; i < im.length - a / 2; i++) {
for (int j = b / 2; j < im[i].length - b / 2; j++) {
int[] arr = new int[a * b];
int c = 0;
for (int k = i - a / 2; k < i + 1 + a / 2; k++) {
for (int l = j - b / 2; l < j + 1 + b / 2; l++) {
arr[c] = im[k][l];
c++;
} }
Arrays.sort(arr);
newIm[i][j] = arr[a * b / 2];
} }
return newIm; }
```

transformation power:

```
public static int[][] TransformPower(int[][] im, double c,double gamma) {
int[][] new_im = new int[im.length][im[0].length];
int max = Integer.MIN_VALUE;
int min = Integer.MAX_VALUE;
for (int i = 0; i < im.length; i++) {
for (int j = 0; j < im[i].length; j++) {
new_im[i][j] = Power(im[i][j],c,gamma);
max = Math.max(max, new_im[i][j]);
min = Math.min(min, new_im[i][j]);
} }
for (int i = 0; i < new_im.length; i++) {
for (int j = 0; j < new_im[0].length; j++) {
new_im[i][j] = 255 * (new_im[i][j] - min) / (max - min);
} }
return new_im; }
```

For every one add :

```
public static int Negative(int i){
return 255-i; }
public static int Log(int i,double c){
return (int)(c*Math.log10(i+1)); }
public static int Power(int i,double c,double gamma){
return (int) (c*Math.pow(i, gamma)); }
```

Adaptive Filter:

```
public static int[][] adaptive(int[][] im, int maxLevel)
{
int[][] newIm = new int[im.length][im[0].length];
maxLevel = maxLevel * 2 + 1;
// adaptive median filter
for (int i = maxLevel / 2; i < im.length - maxLevel / 2;
i++) {
for (int j = maxLevel / 2; j < im[i].length - maxLevel /
2; j++) {
int level = 3;
for (int lvl = level; lvl <= maxLevel; lvl += 2) {

int[] arr = new int[lvl * lvl];
int c = 0;
for (int k = i - lvl / 2; k < i + 1 + lvl / 2; k++) {
for (int l = j - lvl / 2; l < j + 1 + lvl / 2; l++) {
arr[c] = im[k][l];
c++;
} }
Arrays.sort(arr);
int min = arr[0];
int median = arr[lvl * lvl / 2];
int max = arr[arr.length - 1];
if (min < median && median < max) {
if (min < im[i][j] && im[i][j] < max) {
newIm[i][j] = im[i][j];
break;
} else {
newIm[i][j] = median;
}
} else if(lvl == maxLevel) {
newIm[i][j] = median;
} } } }
return newIm; }
```

Alpha Filter:

```
public static int[][] alphaFilter(int[][] im, int a, int b, int alpha) {
int[][] newIm = new int[im.length][im[0].length];
if (alpha > a * b - 1) {
return newIm; }
for (int i = a / 2; i < im.length - a / 2; i++) {
for (int j = b / 2; j < im[i].length - b / 2; j++) {
int[] arr = new int[a * a];
int c = 0;
for (int k = i - a / 2; k < i + 1 + a / 2; k++) {
for (int l = j - b / 2; l < j + 1 + b / 2; l++) {
arr[c] = im[k][l];
c++;
} }
Arrays.sort(arr);
int sum = 0;
for (int k = alpha; k < arr.length - alpha; k++) {
sum += arr[k]; }
newIm[i][j] = sum / (arr.length - 2 * alpha);
} }
return newIm; }
```

Midpoint Filter:

```
public static int[][] midPointFilter(int[][] im, int a, int b) {
int[][] newIm = new int[im.length][im[0].length];
for (int i = a / 2; i < im.length - a / 2; i++) {
for (int j = b; j < im[i].length - b / 2; j++) {
int min = 256;
int max = -1;
for (int k = i - a / 2; k <= i + a / 2; k++) {
for (int l = j - b / 2; l <= j + b / 2; l++) {
min = Math.min(min, im[k][l]);
max = Math.max(max, im[k][l]);
} }
newIm[i][j] = (min + max) / 2;
} }
return newIm; }
```

Harmonic Mean Filter:

```
public static int[][] harmonicMean(int[][] im, int a, int b) {
int[][] newIm = new int[im.length][im[0].length];
double square = a * b;
for (int i = a / 2; i < im.length - a / 2; i++) {
for (int j = b; j < im[i].length - b / 2; j++) {
double product = 0;
for (int k = i - a / 2; k <= i + a / 2; k++) {
for (int l = j - b / 2; l <= j + b / 2; l++) {
product += 1.0 / im[k][l];
} }
newIm[i][j] = (int) (square / product);
} }
return newIm; }
```

Contra Harmonic Filter:

```
public static int[][] contraHarmonic(int[][] im, int a, int b, int Q) {
int[][] newIm = new int[im.length][im[0].length];
for (int i = a / 2; i < im.length - a / 2; i++) {
for (int j = b; j < im[i].length - b / 2; j++) {
double q = 0;
double qPlus1 = 0;
for (int k = i - a / 2; k <= i + a / 2; k++) {
for (int l = j - b / 2; l <= j + b / 2; l++) {
q += Math.pow(im[k][l], Q);
qPlus1 += Math.pow(im[k][l], Q + 1);
} }
newIm[i][j] = (int) (qPlus1 / q);
} }
return newIm; }
```

ekhwete ahla
aalam bl
aalam ento