ECE 481 Image Processing

Project 4 Report

Spring 2015

Ming Chen

A20314690

Finally, we reach to the last project of this semester. I have to say this one is much more challenging than the previous ones, especially the Problem 3. So let's start.

Problem 1.

(a) For this problem, there are two m functions as follows:

M-function code (recons.m):

```
function g = recons(f, M, K)
% output reconstruct image g
% input original image f
% M as mask,zig zag scanning order
% K as K coefficient

    fd = im2double(f);
    T = dctmtx(8);
    dct = @(block_struct) T*block_struct.data*T';
    B = blockproc(fd,[8 8],dct);

    m1=M;
    m1(m1<=K) = 1;
    m1(m1>K) = 0;

    B2 = blockproc(B,[8 8],@(block_struct) m1.*block_struct.data);
    invdct = @(block_struct) T'*block_struct.data*T;
    g = blockproc(B2,[8 8], invdct);
end
```

M-function code (p2n.m):

```
function peaksnr = p2n(f,g)
    fd = im2double(f);
    gd = im2double(g);
    [M,N] = size(f);

    error = fd-gd;
    MSE = sum(sum(error.*error))/(M*N);

    peaksnr = 10*log10((255^2)/MSE);

end
```

The testing script for ‘lena.tif’ is shown below:

```
f=imread('lena.tif');

M = [ 0 1 5 6 14 15 27 28
    2 4 7 13 16 26 29 42
    3 8 12 17 25 30 41 43
    9 11 18 24 31 40 44 53
    10 19 28 32 39 45 52 54
    20 22 33 38 46 51 55 60
    21 34 37 47 50 56 59 61
    35 36 48 49 57 58 62 63];

K = 32;

g = recons(f,M,K);

subplot(1,2,1);
imshow(f);

subplot(1,2,2);
g=im2uint8(g);
imshow(g);

ratio = p2n(f,g);
fprintf('The peak signal to noise ratio is %.4f\n',ratio);
```

Output image Comparison (lena.tif)

Origin lena.tif

Output lena4.tif (K=4)

	
Origin lena.tif	**Output lena8.tif (K=8)**
	
Origin lena.tif	**Output lena16.tif (K=16)**
	
Origin lena.tif	**Output lena32.tif (K=32)**

The peak signal to noise ratio: (lena)

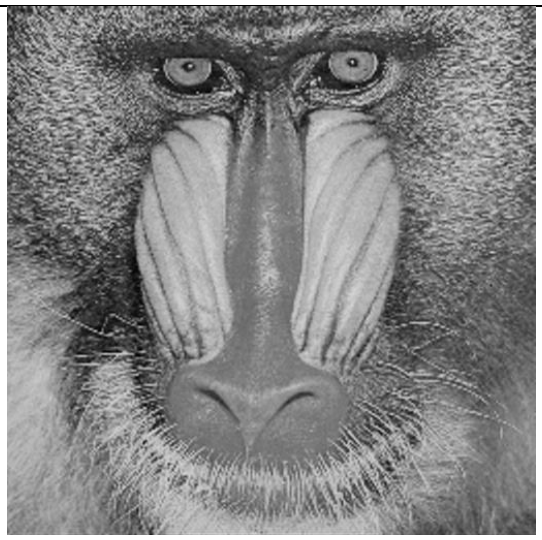
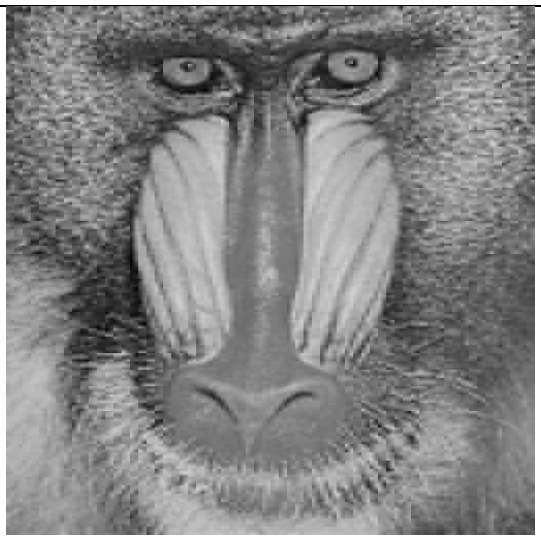
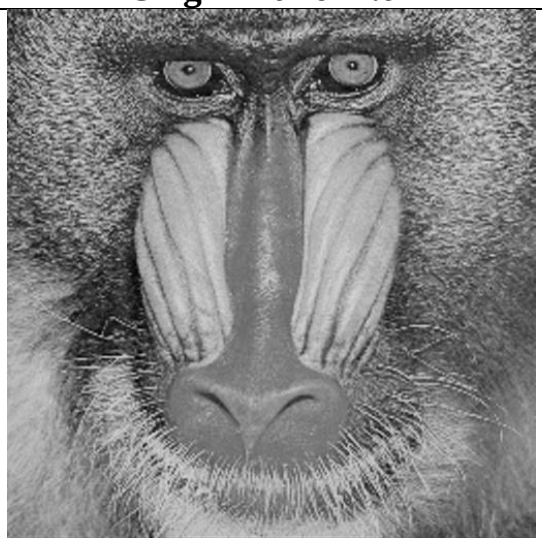
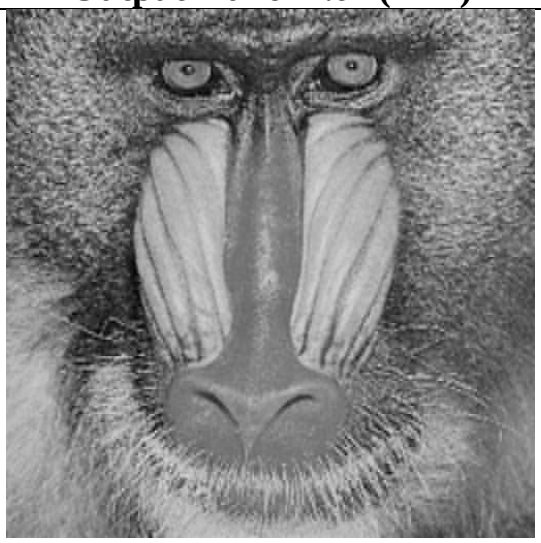
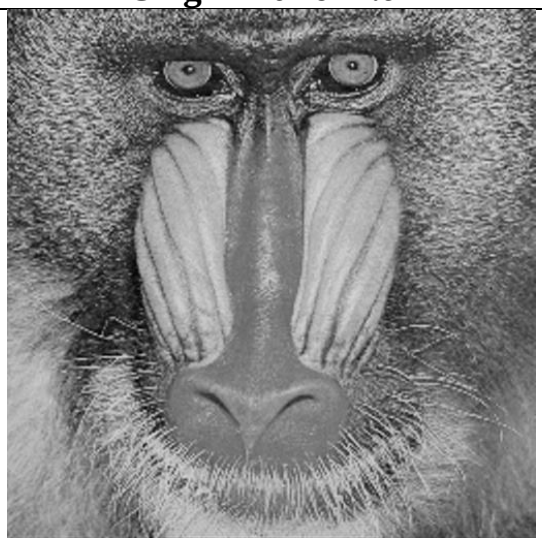
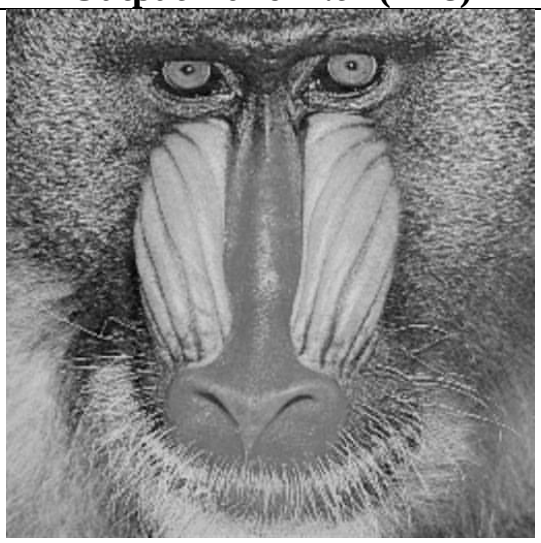
K=4	8	16	32
76.6423	79.9862	83.1108	88.2982

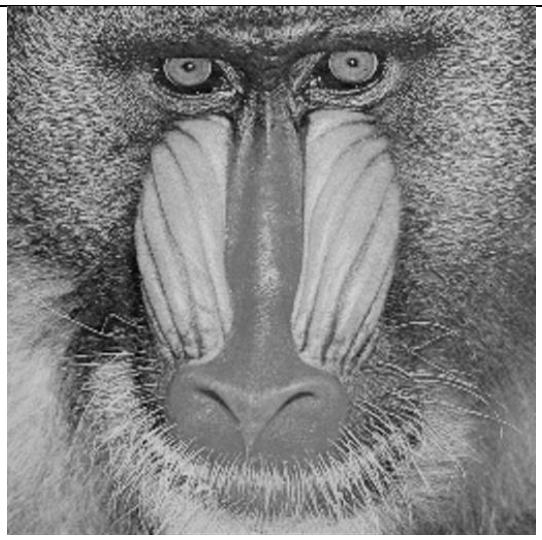
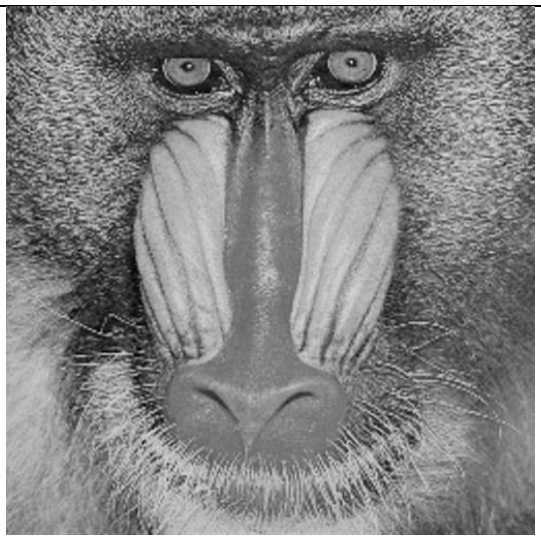
(b)

Testing script for 'mandril.tif'

```
f=imread('mandril.tif');

M = [ 0 1 5 6 14 15 27 28
    2 4 7 13 16 26 29 42
    3 8 12 17 25 30 41 43
    9 11 18 24 31 40 44 53
    10 19 28 32 39 45 52 54
    20 22 33 38 46 51 55 60
    21 34 37 47 50 56 59 61
    35 36 48 49 57 58 62 63];

K = 2;

g = recons(f,M,K);

subplot(1,2,1);
imshow(f);

subplot(1,2,2);
g=im2uint8(g);
imshow(g);

ratio = p2n(f,g);
fprintf('The peak signal to noise ratio is %.4f\n',ratio);
```

Output image Comparison (mandril.tif)

	
Origin mandril.tif	**Output mandril.tif (K=4)**
	
Origin mandril.tif	**Output mandril.tif (K=8)**
	
Origin mandril.tif	**Output mandril.tif (K=16)**

	
Origin mandril.tif	**Output mandril.tif (K=32)**

The peak signal to noise ratio: (mandril)

K=4	**8**	**16**	**32**
72.2054	**74.2164**	**78.2700**	**88.5844**

(c)

As shown in the output comparison for ‘lena’ and ‘mandril’, it is very hard to distinguish the compressed image when K=16 from the one when K=32.

I would choose K=16 as the necessary coefficient to reconstruct a good image.

Problem 2.

(a) For this problem, there are two m functions as follows:

M-function code (recons.m):

```
function g = reconsQ(f, Q, S)
% output reconstruct image g
% input original image f
% Q as original Quantizaion table
% S as scalling factor

    fd = im2double(f);
    fd = fd.*255;
    T = dctmtx(8);
    dct = @(block_struct) T*block_struct.data*T';
    B = blockproc(fd,[8 8],dct);

    %qFun = @(block_struct) floor((block_struct.data+Q*S)./(Q*S)).*(Q*S);
    qFun = @(block_struct) floor(block_struct.data./(Q*S)+0.5);
    B2 = blockproc(B,[8 8],qFun);
    invdct = @(block_struct) T'*block_struct.data*T;
    g = blockproc(B2,[8 8], invdct);
end
```

M-function code (p2n.m):

```
function peaksnr = p2n(f,g)
    fd = im2double(f);
    gd = im2double(g);
    [M,N] = size(f);

    error = fd-gd;
    MSE = sum(sum(error.*error))/(M*N);

    peaksnr = 10*log10((255^2)/MSE);

end
```

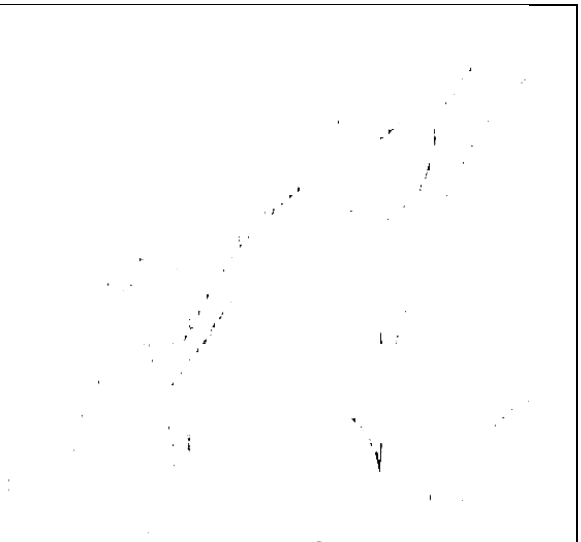
The testing script for ‘lena.tif’ is shown below:

```
f=imread('lena.tif');

Q = [16 11 10 16 24 40 51 61
    12 12 14 19 26 58 60 55
    14 13 16 24 40 57 69 56
```

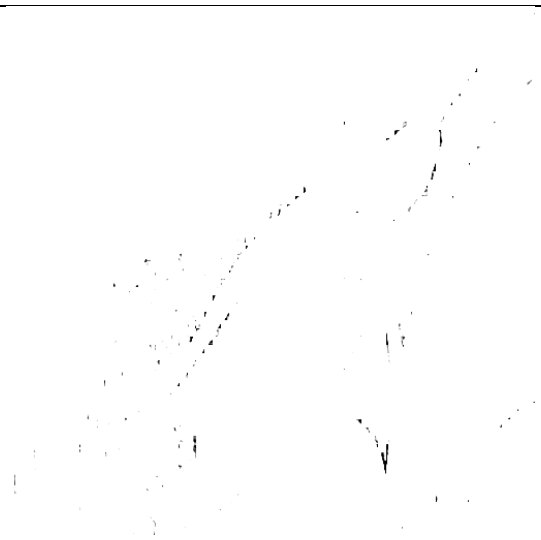
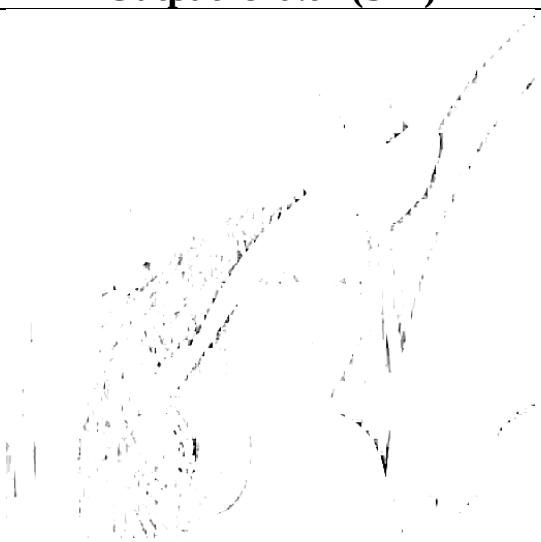
```
    14 17 22 29 51 87 80 62
    18 22 37 56 68 109 103 77
    24 35 55 64 81 104 113 92
    49 64 78 87 103 121 120 101
    72 92 95 98 112 100 103 99];

S = 8;

g = reconsQ(f,Q,S);

subplot(1,2,1);
imshow(f);

subplot(1,2,2);
g=im2uint8(g);
imshow(g);

psnr = p2n(f,g);
fprintf('The peak signal to noise ratio is %0.4f\n',psnr);
```

Output image Comparison (lena.tif)

Origin lena.tif



Output lena.tif (S=0.5)

 **Origin lena.tif**	 **Output lena.tif (S=1)**
 **Origin lena.tif**	 **Output lena.tif (S=2)**
 **Origin lena.tif**	 **Output lena.tif (S=4)**

	
Origin lena.tif	**Output lena.tif (S=8)**

The peak signal to noise ratio: (lena)

S=0.5	1	2	4	8
53.8227	53.8394	53.9098	54.2828	56.4722

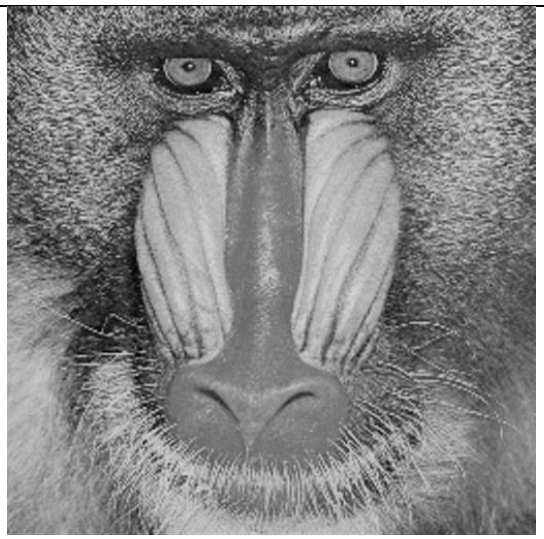
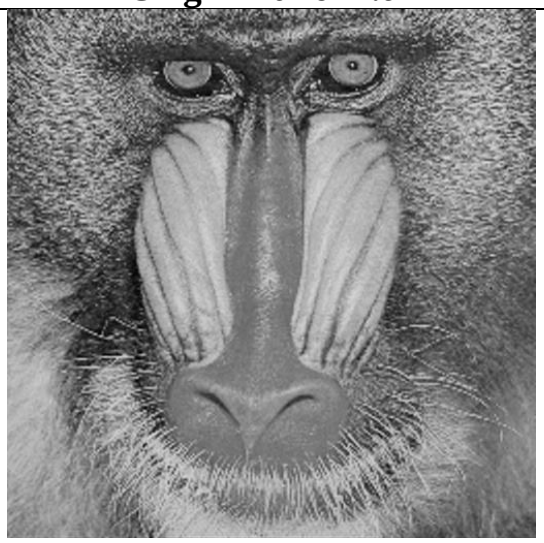
(b)

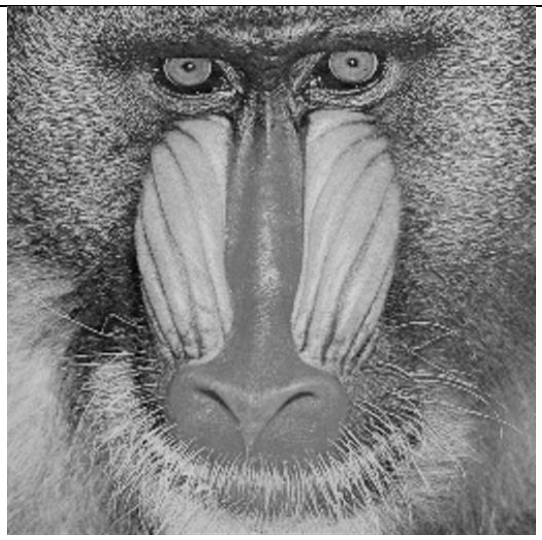
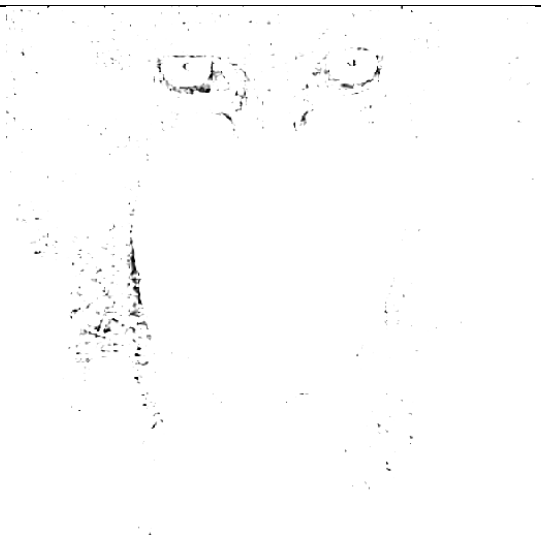
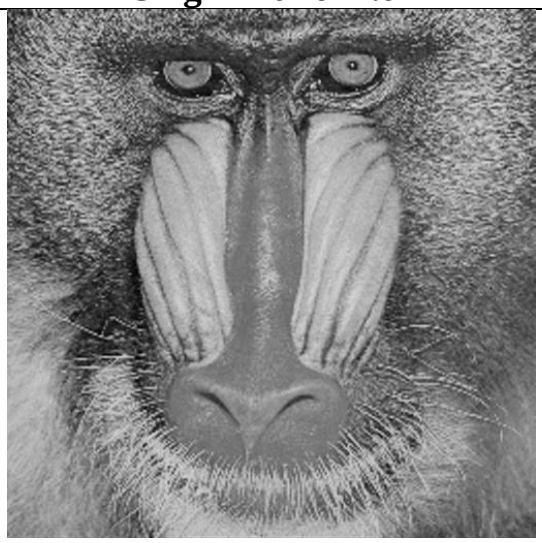
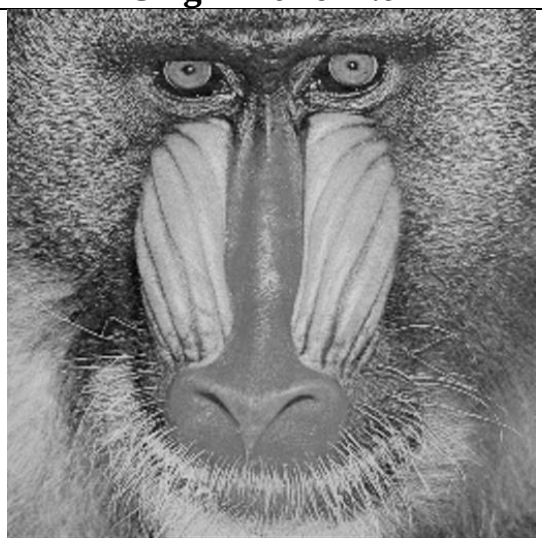
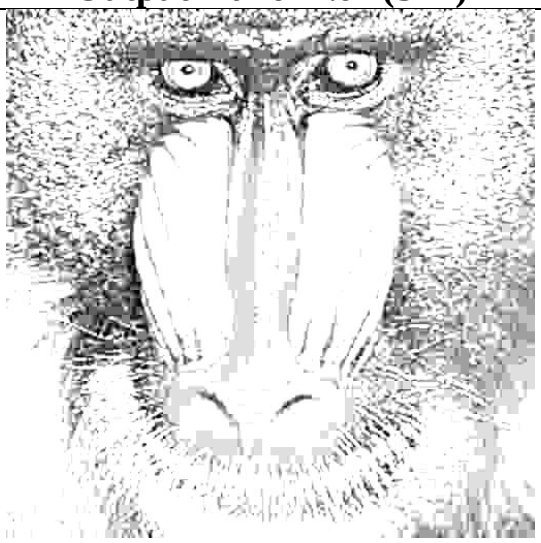
Testing script for 'mandril.tif'

```
f=imread('mandril.tif');

Q = [16 11 10 16 24 40 51 61
    12 12 14 19 26 58 60 55
    14 13 16 24 40 57 69 56
    14 17 22 29 51 87 80 62
    18 22 37 56 68 109 103 77
    24 35 55 64 81 104 113 92
    49 64 78 87 103 121 120 101
    72 92 95 98 112 100 103 99];

S = 0.5;

g = reconsQ(f,Q,S);

subplot(1,2,1);
imshow(f);

subplot(1,2,2);
g=im2uint8(g);
imshow(g);
```

```
psnr = p2n(f,g);
fprintf('The peak signal to noise ratio is %0.4f\n',psnr);
```

Output image Comparison (mandril.tif)

	
Origin mandril.tif	**Output mandril.tif (S=0.5)**
	
Origin mandril.tif	**Output mandril.tif (S=1)**

	
Origin mandril.tif	**Output mandril.tif (S=2)**
	
Origin mandril.tif	**Output mandril.tif (S=4)**
	
Origin mandril.tif	**Output mandril.tif (S=8)**

The peak signal to noise ratio: (mandril)

S=0.5	1	2	4	8
53.3899	53.4019	53.4721	54.2999	56.9715

(c)

As shown in the output comparison for 'lena' and 'mandril', I know there must have something wrong with my equation in reconsQ.m. But I followed exactly the same as the lecture slides, so I don't know what goes wrong. I assume that a quick fix will be on that equation. If I have the right equation, my program should work very well.

I would choose S=8 as the necessary coefficient to reconstruct a good image.

Problem 3.

Ok, the best part for this project. Like all of the other students, I searched the internet and tried to find a good implementation for this Huffman encoding function, but they are all using greedy approach and construct actual trees and nodes. I can build a similar tree in a minute using C, Java and Python, but I am not familiar with the 'struct' data structure in Matlab. Then I decided to use the similar idea but using only array and cell to achieve the virtual tree.

In order for you to understand my idea and my code, I will present you a small picture to illustrate my idea.

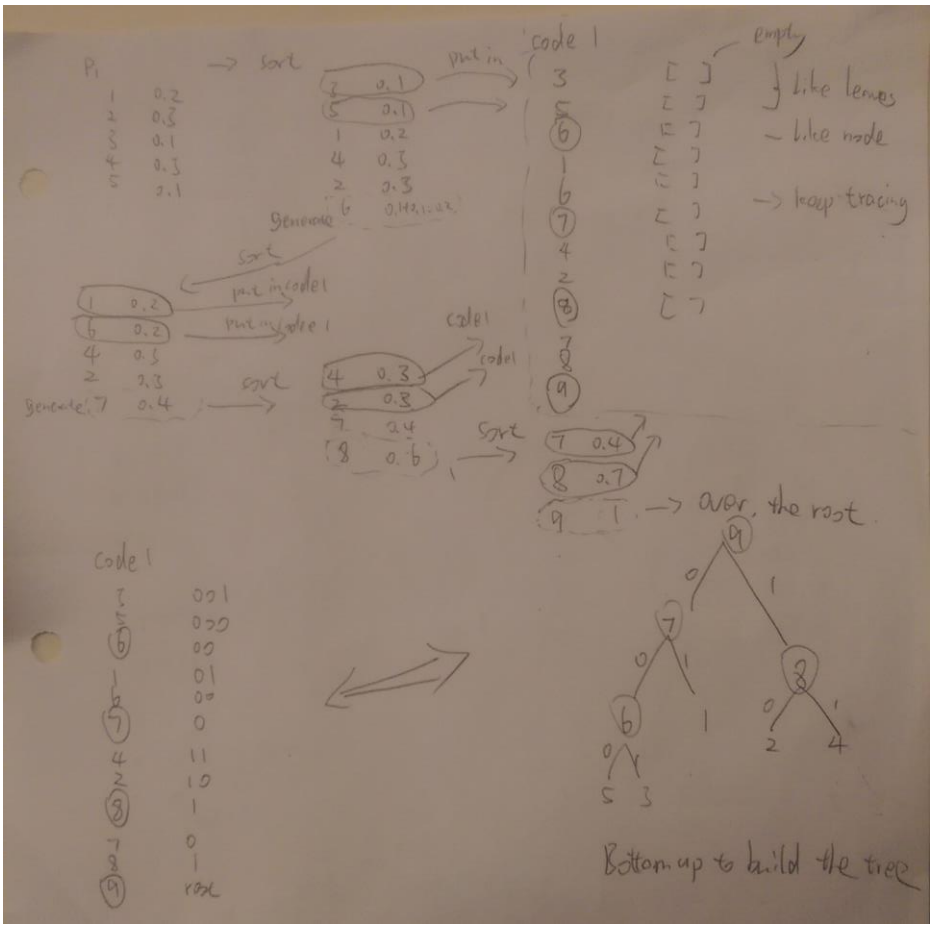
For this problem, there is a M-function called Huffman.m. It takes a dictionary consists of symbols and normalized histogram. Please look at the simplified version of testing data.

1	0.2000
2	0.3000
3	0.1000
4	0.3000
5	0.1000

Let's say we can this cell structure. The first column is the symbols assigned to each normalized histogram. They should be unique and cover all the probabilities. It is called a dictionary.

Then we pass this to the 'huffman(p)' function and let it do the rest of the job.

The idea in the function can be illustrated as following:



M-function code (huffman.m):

```
function H_CODE = huffman (p)

%create a duplicate dictionary to do sorting
p1=p;

%create a cell array to store tree nodes and leaves
%first column to store indeies,
%second one for assigning binary coding later
code1=cell(size(p,1),2);

n=size(p,1); %original size
n1=n;  %increasing index
i=1;        %record code1 increasing size

%at least two elements remaining in the dictionary
while size(p1,1)>= 2
    newprob=0;
    n1=n1+1;   %parent index +1

    p1=sortrows(p1,2);
    code1{i,1}=p1{1,1};
    newprob=p1{1,2};
```

```
    i=i+1;
    p1(1,:)=[];     %remove first row

    code1{i,1}=p1{1,1};
    newprob=newprob+p1{1,2};
    i=i+1;
    p1(1,:)=[];     %remove second row

    p1{size(p1,1)+1,1}=n1;  %insert a new row, a new parent
    p1{size(p1,1),2}=newprob;
    code1{i,1}=n1;          %create a node in our "tree"
    i=i+1;
end

%at this point, all indexs in code1 is set
%like a virtual tree, but we only record the sequence of the node and
%leaves names

%now we are ready to tackle the last problem
%construct binary strings for leaves
len_code1=size(code1,1);

for i=len_code1:-3:3
    if i==len_code1 %base case
        if code1{i-1,1}>n       %just a node on rihgt
            inode=find([code1{1:i-3,1}]==code1{i-1,1});  %find the index of
this node above
            code1{inode,2}(end+1)=1;    %assign 1 to the bigger node
        else                    %one of the symbols, one of the leaves
            code1{i-1,2}(end+1)=1;      %assign 1 to the bigger leave
        end

        if code1{i-2,1}>n       %just a node on left
            inode=find([code1{1:i-3,1}]==code1{i-2,1});  %find the index of
this node above
            code1{inode,2}(end+1)=0;    %assign 0 to the smaller node
        else                    %one of the symbols, one of the leaves
            code1{i-2,2}(end+1)=0;      %assign 0 to the smaller leave
        end
    else
        if code1{i-1,1}>n       %just a node on rihgt
            inode=find([code1{1:i-3,1}]==code1{i-1,1});  %find the index of
this node above
            code1{inode,2}=cat(2,code1{inode,2},code1{i,2});
            code1{inode,2}(end+1)=1;    %assign 1 to the bigger node
        else                    %one of the symbols, one of the leaves
            code1{i-1,2}=cat(2,code1{i-1,2},code1{i,2});     %assign the
parent bits
            code1{i-1,2}(end+1)=1;              %assign 1 to the bigger
leave
        end

        if code1{i-2,1}>n       %just a node on left
            inode=find([code1{1:i-3,1}]==code1{i-2,1});  %find the index of
this node above
```

```
            code1{inode,2}=cat(2,code1{inode,2},code1{i,2});
            code1{inode,2}(end+1)=0;     %assign 0 to the smaller node
        else                        %one of the symbols, one of the leaves
            code1{i-2,2}=cat(2,code1{i-2,2},code1{i,2});     %assign the
parent bits
            code1{i-2,2}(end+1)=0;      %assign 0 to the smaller leave
        end
    end
end

%remove nodes and root, only keep leaves;
code1=code1([code1{:,1}]<=n,:);

%sort it in terms of symbols
H_CODE = sortrows(code1,1);

end
```

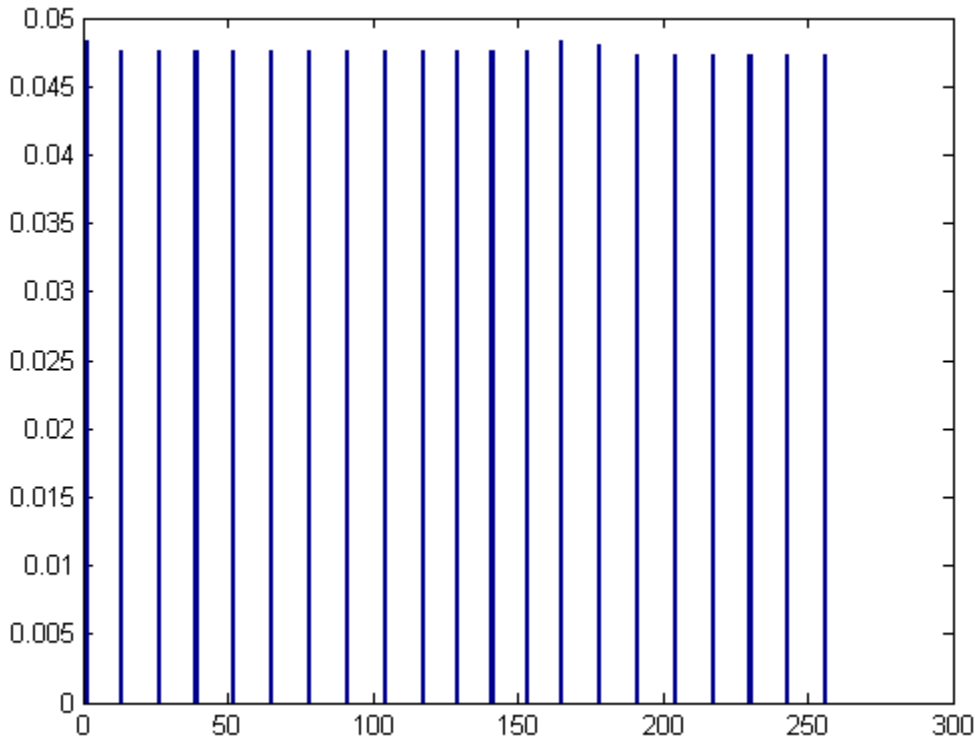
(a) all about 'test1.tif'

Testing script for 'test1'.tif:

```
f=imread('test1.tiff');
h=imhist(f);
[m,n]=size(f);
P=h./(n*m); %normalized histogram

%h=h(h~=0);
horz = 1:1:256;
P1=P(1:1:256);
bar(horz,P1);

P=P(P~=0);
E=sum(-(P.*log2(P))); %entropy
fprintf('The entropy is %.4f\n',E);

%crate a dictionary
h_length=size(P,1);
symbols=1:h_length;
dict=cell(h_length,2);
for i=1:h_length
    dict{i,1}=symbols(i);
    dict{i,2}=P(i);
end



codes = huffman(dict);

%calculate the avg string length
stringTotal=0;
for i=1:h_length
   stringTotal = stringTotal+size(codes{i,2},2);
end
```

```
avgL=stringTotal/h_length;
fprintf('The average string length is %.2f\n',avgL);
```

(a.1) The normalized histogram of the image



(a.2) Calculate the entropy

```
The entropy is 4.3923
```

(a.3) The Huffman Code

1	[1,0,0,1]
2	[1,1,1,0,0]
3	[1,1,1,0,1]
4	[1,1,1,1,0]
5	[1,1,1,1,1]
6	[0,0,0,0]
7	[0,0,0,1]
8	[0,0,1,0]
9	[0,0,1,1]
10	[0,1,0,0]
11	[0,1,0,1]
12	[0,1,1,0]
13	[0,1,1,1]
14	[1,0,1,0]
15	[1,0,0,0]
16	[1,0,1,1,0]
17	[1,0,1,1,1]
18	[1,1,0,0,0]
19	[1,1,0,0,1]
20	[1,1,0,1,0]
21	[1,1,0,1,1]

(a.4) Average number of bits

```
The average string length is 4.48
```

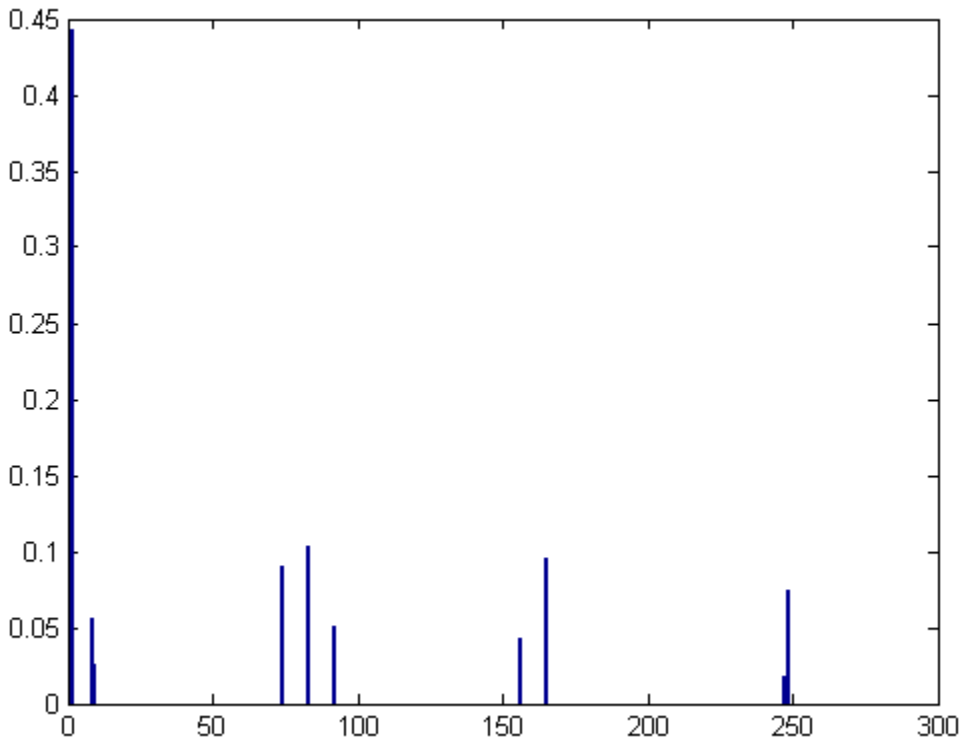
(b) All about ‘test2.tif’

Testing script for ‘test2’.tif:

```
f=imread('test2.bmp');
h=imhist(f);
[m,n]=size(f);
P=h./(n*m); %normalized histogram

%h=h(h~=0);
horz = 1:1:256;
P1=P(1:1:256);
bar(horz,P1);

P=P(P~=0);
E=sum(-(P.*log2(P))); %entropy
fprintf('The entropy is %.4f\n',E);

%crate a dictionary
h_length=size(P,1);
symbols=1:h_length;
dict=cell(h_length,2);
for i=1:h_length
    dict{i,1}=symbols(i);
```

```
    dict{i,2}=P(i);
end


codes = huffman(dict);


%calculate the avg string length
stringTotal=0;
for i=1:h_length
   stringTotal = stringTotal+size(codes{i,2},2);
end

avgL=stringTotal/h_length;
fprintf('The average string length is %.2f\n',avgL);
```

(b.1) The normalized histogram of the image



(b.2) Calculate the entropy

```
The entropy is 2.6614
```

(b.3) The Huffman Code

1	0
2	[1,0,1,1]
3	[1,1,0,1,1,1]
4	[1,1,1,0]
5	[1,0,0]
6	[1,0,1,0]
7	[1,1,0,1,0]
8	[1,1,1,1]
9	[1,1,0,1,1,0]
10	[1,1,0,0]

(b.4) Average number of bits

```
The average string length is 4.10
```

References:

1. http://www.mathworks.com/help/images/discrete-cosine-transform.html

2. https://www.cs.auckland.ac.nz/software/AlgAnim/huffman.html