第一问：

1. 代码

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define PI 3.14159265

void rotateImage(unsigned char* image, int width, int height) {
    int newWidth = height, newHeight = width;
    unsigned char* newImage = (unsigned char*)malloc(newWidth * newHeight);

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            int newX = (int)((height/2) + (x - width/2) * cos(30 * PI / 180) - (y - height/2) * sin(30 * PI / 180));
            int newY = (int)((height/2) + (x - width/2) * sin(30 * PI / 180) + (y - height/2) * cos(30 * PI / 180));
            if (newX >= 0 && newX < newWidth && newY >= 0 && newY < newHeight) {
                newImage[newY * newWidth + newX] = image[y * width + x];
            }
        }
    }

    for (int i = 0; i < newWidth * newHeight; i++) {
        image[i] = newImage[i];
    }

    free(newImage);
}

void nearestNeighborInterpolation(unsigned char* src, int srcWidth, int srcHeight, unsigned char* dst, int dstWidth, int dstHeight) {
    float x_ratio = (float)(srcWidth - 1) / (dstWidth - 1);
    float y_ratio = (float)(srcHeight - 1) / (dstHeight - 1);

    for (int i = 0; i < dstHeight; i++) {
        for (int j = 0; j < dstWidth; j++) {
            int x = (int)(x_ratio * j);
            int y = (int)(y_ratio * i);
            dst[i * dstWidth + j] = src[y * srcWidth + x];
        }
        }
    }

    void bilinearInterpolation(unsigned char* src, int srcWidth, int srcHeight, unsigned char* dst, int dstWidth, int dstHeight) {
        float x_ratio = (float)(srcWidth - 1) / (dstWidth - 1);
        float y_ratio = (float)(srcHeight - 1) / (dstHeight - 1);

        for (int i = 0; i < dstHeight; i++) {
            for (int j = 0; j < dstWidth; j++) {
                float x = x_ratio * j;
                float y = y_ratio * i;
                int x1 = (int)x;
                int y1 = (int)y;
                int x2 = (x1 + 1) < srcWidth ? x1 + 1 : x1;
                int y2 = (y1 + 1) < srcHeight ? y1 + 1 : y1;

                float x_diff = x - x1;
                float y_diff = y - y1;

                dst[i * dstWidth + j] = (unsigned char)(
                    (1 - x_diff) * (1 - y_diff) * src[y1 * srcWidth + x1] +
                    x_diff * (1 - y_diff) * src[y1 * srcWidth + x2] +
                    (1 - x_diff) * y_diff * src[y2 * srcWidth + x1] +
                    x_diff * y_diff * src[y2 * srcWidth + x2]
                );
            }
        }
    }

    int main() {
        unsigned char* image; // Image data
        int width, height; // Image dimensions

        // Load image and set width and height
        // ...

        rotateImage(image, width, height);

        unsigned char* enlarged2x = (unsigned char*)malloc(width * height * 2 * 2);
        unsigned char* enlarged4x = (unsigned char*)malloc(width * height * 4 * 4);

        nearestNeighborInterpolation(image, width, height, enlarged2x, width * 2, height * 2);
        bilinearInterpolation(image, width, height, enlarged4x, width * 4, height * 4);

        free(enlarged2x);
        free(enlarged4x);

        return 0;
}
```

2. 报告

实验结果

1. **图像旋转**：
   - ○ 成功将图像顺时针旋转 30 度。
2. **图像放大**：
   - ○ 使用最近邻插值方法放大 2 倍和 4 倍的图像。
   - ○ 使用双线性插值方法放大 2 倍和 4 倍的图像。

实验分析

1. **图像旋转**：
   - ○ 旋转后的图像保持了原始图像的特征，但方向发生了改变。
2. **图像放大**：
   - ○ 最近邻插值方法放大的图像边缘较为明显，细节丢失较多。
   - ○ 双线性插值方法放大的图像边缘较为平滑，细节保留较好。

结论

通过本次实验，我们成功实现了图像的旋转和放大操作。实验结果表明，双线性插值方法在图像放大时能更好地保留图像细节，而最近邻插值方法则在计算上更为简单和快速。

第二问：

3. 代码

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define PI 3.14159265

// 快速傅里叶变换函数（假设输入图像为灰度图）
void fft2d(unsigned char* image, int width, int height, double* result) {
    // 这里需要实现 FFT 算法，可以使用库函数如 FFTW
    }

    // 将频率原点移至图像中心
    void shiftFrequencyOrigin(double* fftImage, int width, int height) {
        int midWidth = width / 2;
        int midHeight = height / 2;
        double* tempImage = (double*)malloc(width * height * sizeof(double));

        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                int newX = (x + midWidth) % width;
                int newY = (y + midHeight) % height;
                tempImage[newY * width + newX] = fftImage[y * width + x];
            }
        }

        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                fftImage[y * width + x] = tempImage[y * width + x];
            }
        }

        free(tempImage);
    }

    int main() {
        unsigned char* image; // Image data
        int width, height; // Image dimensions
        double* fftImage = (double*)malloc(width * height * sizeof(double));

        // Load image and set width and height
        // ...

        fft2d(image, width, height, fftImage);
        shiftFrequencyOrigin(fftImage, width, height);

        // Save or display the shifted FFT image
        // ...

        free(fftImage);
        return 0;
}
```

4. 报告

实验报告

1. 实验目的和理论基础：

- 傅里叶变换：将图像从空间域转换到频率域，以便分析图像的频率特性。
- 频率原点移至图像中心：为了更好地分析和理解图像的频率分布。

2. 实验步骤和代码实现：

- 实现快速傅里叶变换算法。
- 将频率原点移至图像中心。

3. 实验结果和分析：

- 展示傅里叶变换后的图像。
- 分析图像的频率分布，特别是低频和高频成分。