Student: Ta Quoc Viet (299954)

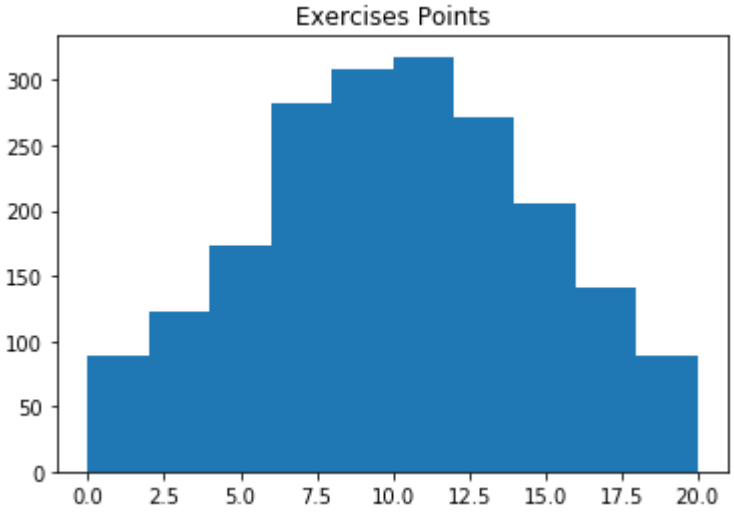
Task 1

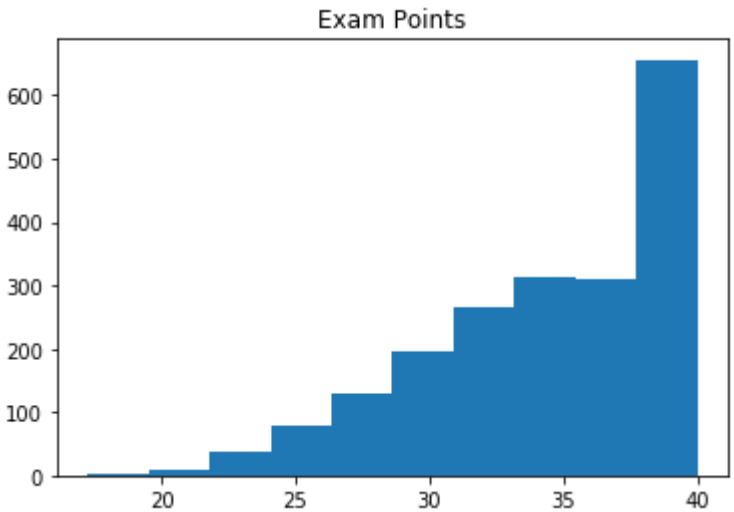
Load data file `bogus_student_data.txt`

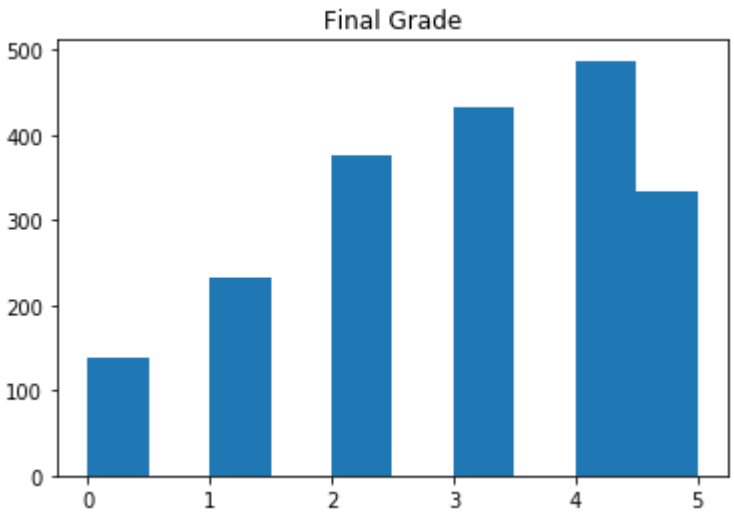
In [157]:
```
import numpy as np
import pandas as pd

def plot_scatter(x, y, xlabel, ylabel, title=None, show=True, axisEqual=False
):
    plt.scatter(x, y)
    if title==None:
        plt.title('{}//{}'.format(xlabel, ylabel))
    else:
        plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    if axisEqual:
        plt.axis('equal')
    if show:
        return plt.show()
    return plt.gca()

def plot_hist(data, title):
    plt.hist(data)
    plt.title(title)
    plt.show()

STUDENT_DATA_PATH = './bogus_student_data.txt'
data = np.genfromtxt(STUDENT_DATA_PATH, names=True)
```

Visualize the data

In [158]:
```
import matplotlib.pyplot as plt
%matplotlib inline

plot_hist(data['exercise_points'], 'Exercises Points')
plot_hist(data['exam_points'], 'Exam Points')
plot_hist(data['grades'], 'Final Grade')
```

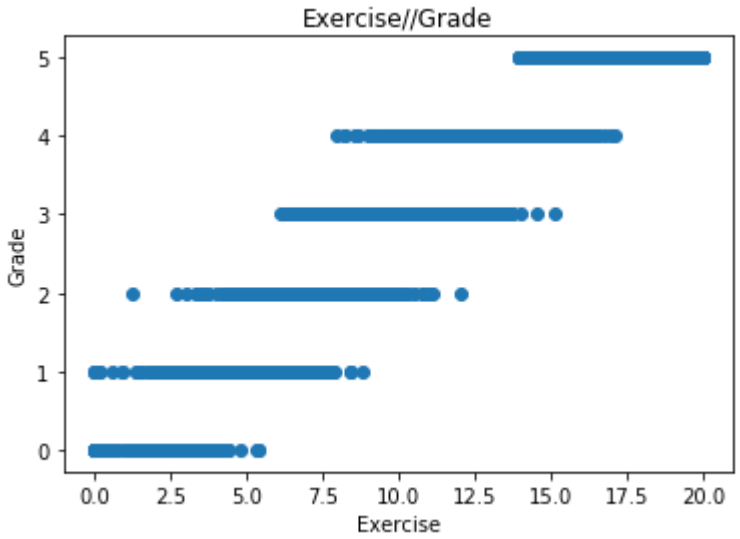


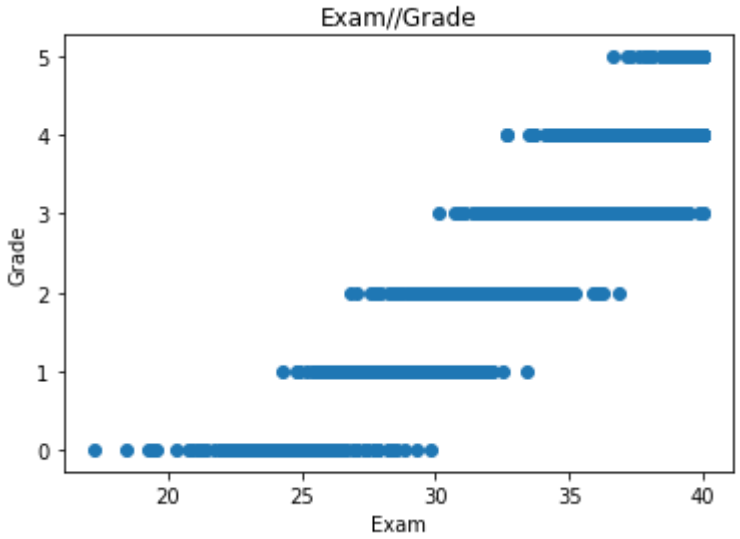


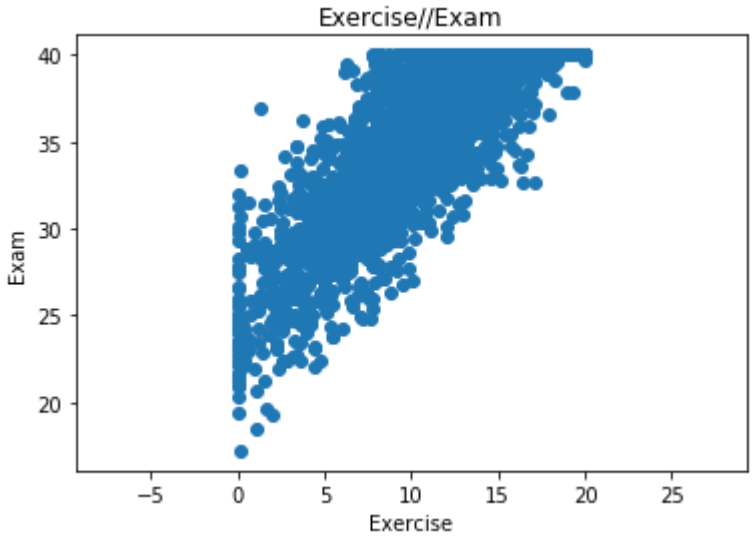


- We can't really infer so much from the histograms of the data other than the individual elements' distributions.

In [159]:
```
plot_scatter(data['exercise_points'], data['grades'], 'Exercise', 'Grade')
plot_scatter(data['exam_points'], data['grades'], 'Exam', 'Grade')
plot_scatter(data['exercise_points'], data['exam_points'], 'Exercise', 'Exam',
axisEqual=True)
```







We can now see some patterns here:

- The exercise points and grade points are proportional, i.e. the higher the exercise points are, the higher the grade points will be.
- The same themes apply to the relations between exam points and grade points, and exercise points and exam points ## Statistics
- Mean and std of the Exercise Points

In [160]:
```
print('Mean: ', np.mean(data['exercise_points']))
print('Std: ', np.std(data['exercise_points']))
```

```
Mean:  10.10813
Std:  4.659659580173212
```

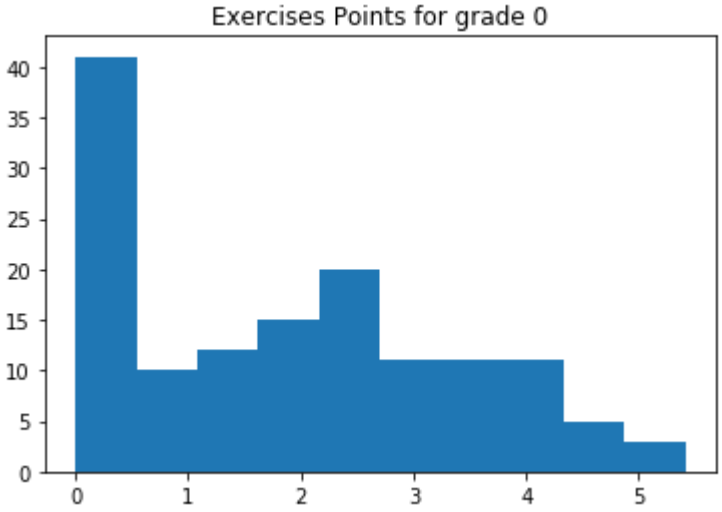
- Mean and std of the Exercise Points for each grade

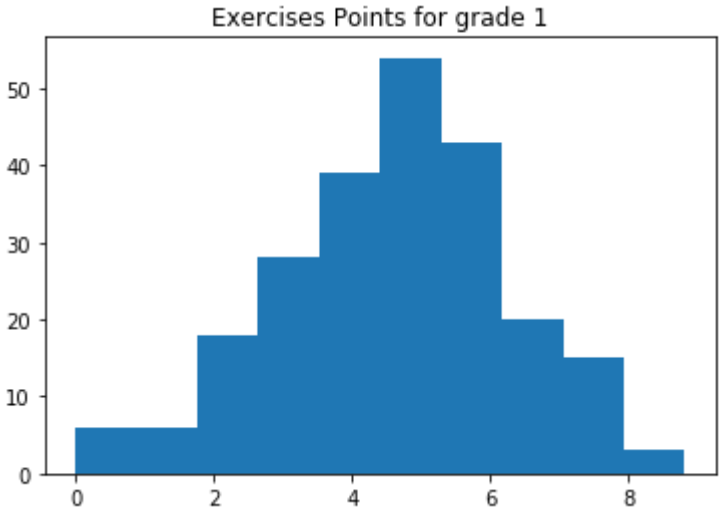
In [161]:
```
for g in range(0, 6):
    print('Mean and std for Grade == ', g)
    currentGrade = data[np.where(data['grades'] == g)]
    print('Mean: ', np.mean(currentGrade['exercise_points']))
    print('Std: ', np.std(currentGrade['exercise_points']))
```

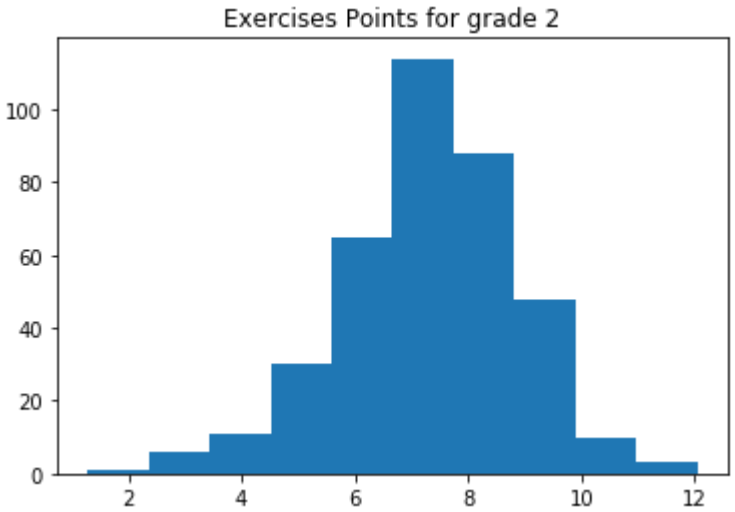
```
Mean and std for Grade ==  0
Mean:  1.8310791366906474
Std:  1.51760208936699
Mean and std for Grade ==  1
Mean:  4.599181034482759
Std:  1.7097503654200816
Mean and std for Grade ==  2
Mean:  7.3363297872340425
Std:  1.5564404075659979
Mean and std for Grade ==  3
Mean:  10.006388888888889
Std:  1.6470519164349673
Mean and std for Grade ==  4
Mean:  12.685030800821357
Std:  1.6404048522037007
Mean and std for Grade ==  5
Mean:  16.87395209580838
Std:  1.7757170966361806
```

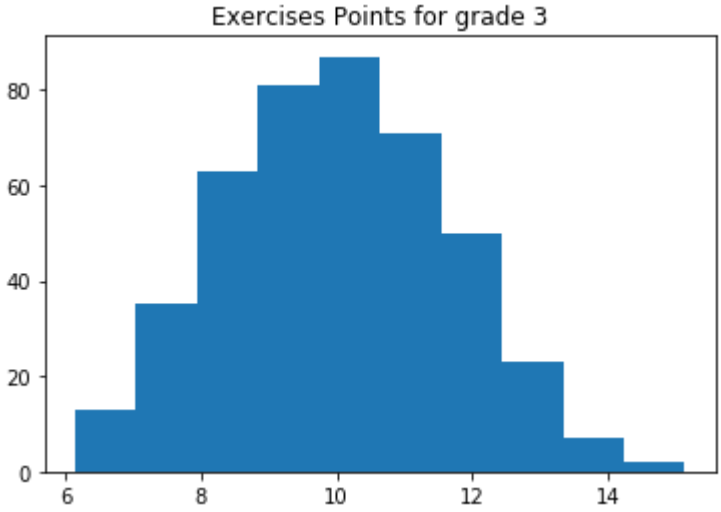
Inference of the grade based on exercises done

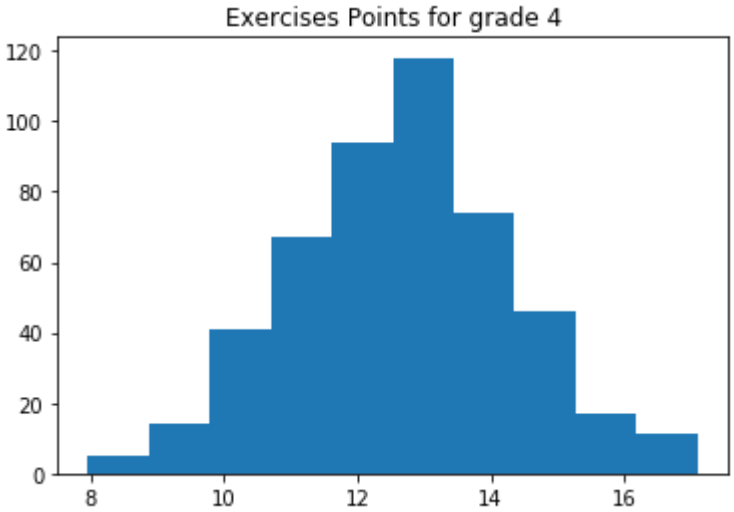
In [162]:
```
for g in range(0, 6):
    currentGrade = data[np.where(data['grades'] == g)]
    plot_hist(currentGrade['exercise_points'], 'Exercises Points for grade {}'
.format(g))
```

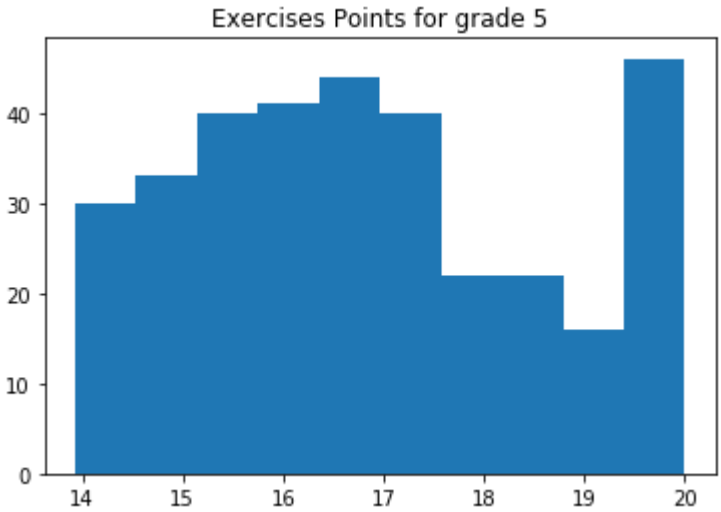

Exercises Points for grade 0
40
35
30
25
20
15
10
5
0
0
1
2
3
4
5


Exercises Points for grade 1
50
40
30
20
10
0
0
2
4
6
8


Exercises Points for grade 2
100
80
60
40
20
0
2
4
6
8
10
12





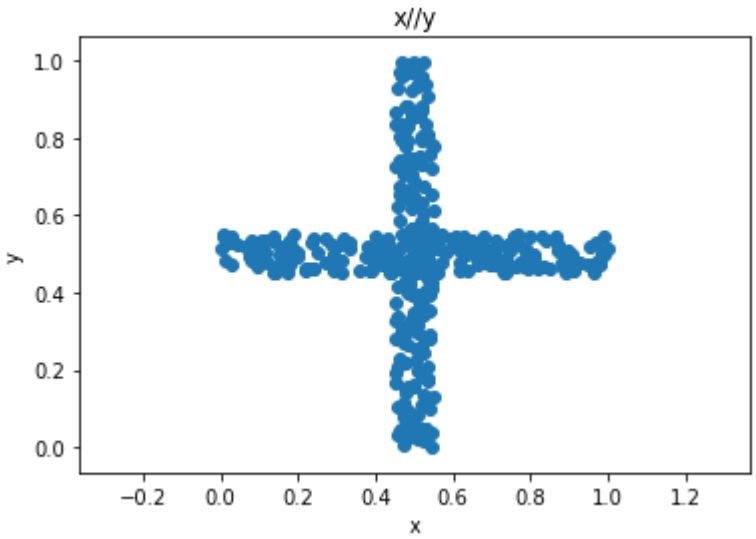


By looking at the histograms and the statistics of the exercise points for each grade, we can infer their final grade at some level:

- If the exercise point is ≤ 3 then there's a high chance that the final grade will be 0
- If the exercise point is ≥ 15 then there's a high chance that the final grade will be 5
- If the exercise point $\in [11, 14]$ then there's a high chance that the final grade will be 4

Task 2

In [163]:
```
import scipy.io as spio

NORMAL_DATA_PATH = './Normal_Data.mat'
# Load data
normal_data = spio.loadmat(NORMAL_DATA_PATH)['Normal_Data']
no_of_points = normal_data.shape[1]
# print(no_of_points)
plot_scatter(normal_data[0, :], normal_data[1, :], 'x', 'y', axisEqual=True)
```



NSA algorithm

In [176]:
```
def generate_detector(min, max):
    r = np.random.random() * (max-min) + min
    xy = np.random.random_sample(2) * (max-min) + min
    return [r, xy[0], xy[1]]


def detect(detector, point):
    r, xd, yd = detector[0], detector[1], detector[2]
    xp, yp = point[0], point[1]
    d = np.sqrt(np.square(xp-xd) + np.square(yp-yd))
    return d < r

# for i in range(10):
#     detector = generate_detector(0, 1)


detectors = []
while len(detectors) < 10:
    detector = generate_detector(0, 1)
    match = False
    index = 0
    while not match and index < no_of_points:
        point = normal_data[:, index]
        match = detect(detector, point)
        index += 1
    if not match:
        detectors.append(detector)

# Draw the data and the detectors on the same plot
ax = plot_scatter(normal_data[0, :], normal_data[1, :], 'x', 'y', show=False,
axisEqual=True)
for detector in detectors:
    circle = plt.Circle(detector[-2:],detector[0],color='r', fill=False)
    ax.add_artist(circle)
```