In [194]:
```
import scipy.stats as sts
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

В случае биномиального распределения m = 50, в случае нормального распределения с неизвестной дисперсией a = 3, в случае нормального распределения с неизвестным математическим ожиданием σ2 = 2.1. Второй параметр выберите случайно из распределения, предложенного в файле.

In [195]:
```
class mystat1(sts.rv_continuous):
    def _pdf(self, x):
        return 21.2238836827 * (x ** (4))
class mystat2(sts.rv_continuous):
    def _pdf(self, x):
        return 9.4033235903 * (x ** (7))
stat1 = mystat1(a=0.28, b=0.75)
stat2 = mystat2(a=0.026, b=0.98)
p = stat1.rvs(size=1)
sigma = stat2.rvs(size=1)
a = sts.uniform.rvs(size=1, loc=-38, scale=(28 + 38))
```

Сгенерируйте выборку X1, ..., XN , N = 1000, из распределений в теоретических задачах.

In [198]:
```
N = 1000
samples = [[] for i in range(4)]
samples[0] = sts.binom.rvs(size=N, n=50, p=p)
samples[1] = sts.expon.rvs(size=N, loc=0, scale=1)
samples[2] = sts.norm.rvs(size=N, loc=3, scale=np.sqrt(sigma))
samples[3] = sts.norm.rvs(size=N, loc=a, scale=np.sqrt(2.1))
```

Для всех □ <= N посчитайте значение эффективной оценки.

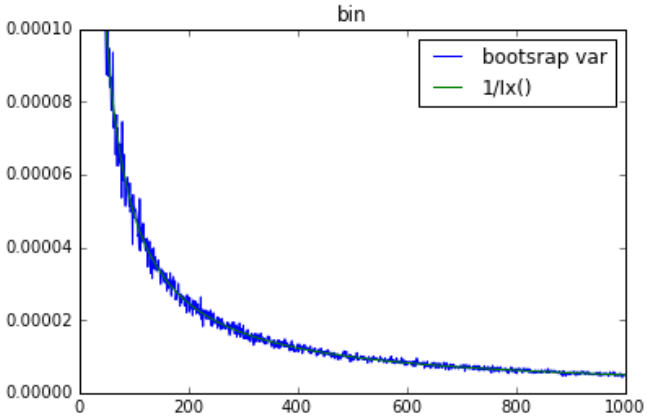
In [199]:
```
res = [[[] for n in range(N)] for i in range(4)]
samples2 = (samples[2] - 3.) ** 2
for n in range(N):
    res[0][n] = np.mean(samples[0][:(n + 1)]) / 50
    res[1][n] = np.mean(samples[1][:(n + 1)])
    res[2][n] = np.mean(samples2[:(n + 1)])
    res[3][n] = np.mean(samples[3][:(n + 1)])
```

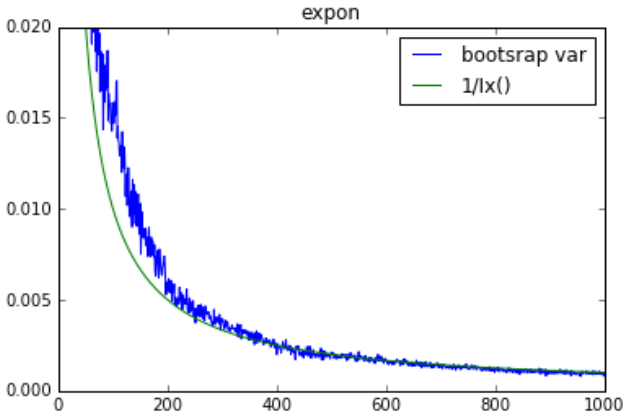
Для всех □ <= N посчитайте бутстрепную оценку дисперсии для эффективной оценки (параметрический бутстреп, количество бутстрепных выборок равно 500).

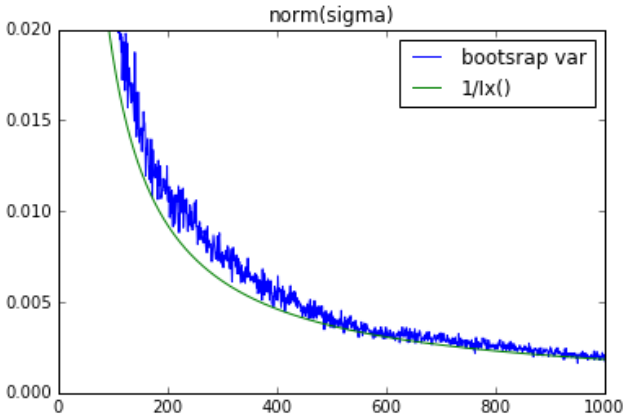
In [204]:
```
M = 500
bootstrap = [[[] for n in range(N)] for i in range(4)]
ix = [[[] for n in range(N)] for i in range(4)]
for n in range(N):
    boot = [[] for i in range(4)]
    for i in range(M):
        samples = sts.binom.rvs(size=(n + 1), n=50, p=res[0][n])
        boot[0].append(np.mean(samples) / 50)
        samples = sts.expon.rvs(size=(n + 1), loc=0, scale=(1. / res[1][n]))
        boot[1].append(np.mean(samples))
        samples = sts.norm.rvs(size=(n + 1), loc=3, scale=np.sqrt(res[2][n])
)
        boot[2].append(np.mean((samples - 3) ** 2))
        samples = sts.norm.rvs(size=(n + 1), loc=res[3][n], scale=np.sqrt(2.
1))
        boot[3].append(np.mean(samples))
    for i in range (4):
        bootstrap[i][n] = np.var(np.array(boot[i]))
    ix[0][n] = p * (1 - p) / 50. / (n + 1)
    ix[1][n] = 1. / (n + 1)
    ix[2][n] = 2 * (sigma ** 2) / (n + 1)
    ix[3][n] = 2.1 / (n + 1)
```

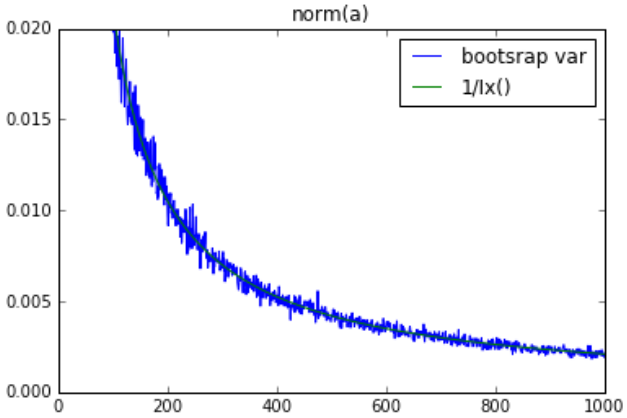
Постройте графики зависимости бутстрепных оценок дисперсий от размера выборки n. Для каждой бутстрепной оценки постройте на том же графике изобразите кривую зависимости $1/IX(\square)$ от n.

In [205]:
```
title = ["bin", "expon", "norm(sigma)", "norm(a)"]
for i in range(4):
    x = np.arange(N) + 1
    plt.plot(x, bootstrap[i], label='bootsrap var')
    plt.plot(x, ix[i], label = '1/Ix()')
    if i == 0: plt.ylim([0, 0.0001])
    else: plt.ylim([0, 0.02])
    plt.title(title[i])
    plt.legend()
    plt.show()
```


bin
bootsrap var
1/Ix()
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000
0
200
400
600
800
1000


expon
bootsrap var
1/Ix()
0.020
0.015
0.010
0.005
0.000
0
200
400
600
800
1000


norm(sigma)
bootsrap var
1/Ix()
0.020
0.015
0.010
0.005
0.000
0
200
400
600
800
1000


norm(a)
bootsrap var
1/Ix()
0.020
0.015
0.010
0.005
0.000
0
200
400
600
800
1000

In [ ]:
```
N = 1000
samples = [[] for i in range(4)]
samples[0] = sts.binom.rvs(size=N, n=50, p=p)
samples[1] = sts.expon.rvs(size=N, loc=0, scale=1)
samples[2] = sts.norm.rvs(size=N, loc=3, scale=np.sqrt(sigma))
samples[3] = sts.norm.rvs(size=N, loc=a, scale=np.sqrt(2.1))
res = [[[] for n in range(N)] for i in range(4)]
for n in range(N):
    res[0][n] = np.min(samples[0][:(n + 1)]) / 50.
    res[1][n] = 1. / 2. / np.mean(samples[1][:(n + 1)]) + (n + 1) / 2 / np.m
in(samples[1][:(n + 1)])
    res[2][n] = np.median(samples[2][:(n + 1)])
    res[3][n] = np.median(samples[3][:(n + 1)])
M = 500
bootstrap = [[[] for n in range(N)] for i in range(4)]
ix = [[[] for n in range(N)] for i in range(4)]
for n in range(N):
    boot = [[] for i in range(4)]
    for i in range(M):
        samples = sts.binom.rvs(size=(n + 1), n=50, p=res[0][n])
        boot[0].append(np.min(samples) / 50.)
        samples = sts.expon.rvs(size=(n + 1), loc=0, scale=(1. / res[1][n]))
        boot[1].append(1. / 2. / np.mean(samples) + (n + 1) / 2 / np.min(sam
ples) )
        samples = sts.norm.rvs(size=(n + 1), loc=3, scale=np.sqrt(res[2][n])
)
        boot[2].append(np.median(samples) )
        samples = sts.norm.rvs(size=(n + 1), loc=res[3][n], scale=np.sqrt(2.
1))
        boot[3].append(np.median(samples))
    for i in range (4):
        bootstrap[i][n] = np.var(np.array(boot[i]))
    ix[0][n] = p * (1 - p) / 50. / (n + 1)
    ix[1][n] = 1. / (n + 1)
    ix[2][n] = 2 * (sigma ** 2) / (n + 1)
    ix[3][n] = 2.1 / (n + 1)
title = ["bin", "expon", "norm(sigma)", "norm(a)"]
for i in range(4):
    x = np.arange(N) + 1
    plt.plot(x, bootstrap[i], label='bootsrap var')
    plt.plot(x, ix[i], label = '1/Ix()')
    plt.title(title[i])
    if i == 0: plt.ylim([0, 0.001])
    else:
        if i == 1: plt.ylim([0, 1])
        else: plt.ylim([0, 0.01])
    plt.legend()
    plt.show()
```

In [ ]:
```
```