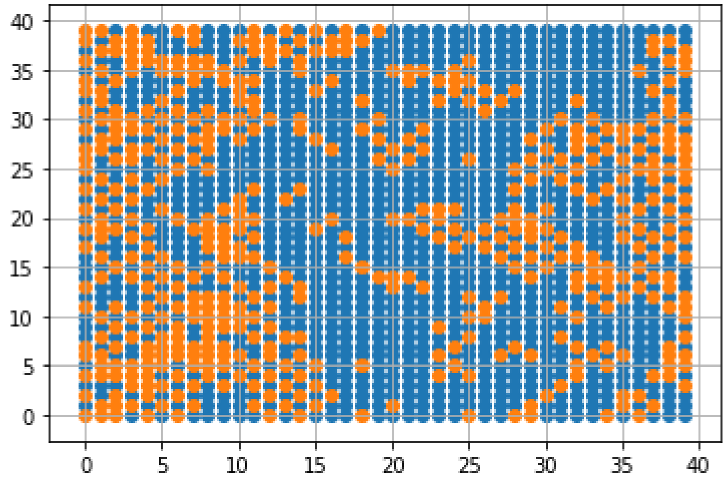
In [41]:
```
"""
Zac Cross
Corwin
PHYS 353

Problem Set 9
"""

import random as r
import numpy as np
import matplotlib.pyplot as plt
import math as m
```

In [53]:
```
# 8.26 Ising model

"""
This program will hopefully model a simple 2D Lattice Ferromagnet

I will have each lattice grid be a matrix of nested lists, with 1 being an
and -1 being a down.

"""

def lattice(size):
    "Returns a randomly generated lattice of ups and downs"
    matrix = []
    for i in range(size):
        col = []
        for j in range(size):
            col.append(r.choice([-1,1]))
        matrix.append(col)
    return matrix

def flip(current):
    """Random flip"""
    return current * -1


def delta_u(current,flip,  i, j, matrix, uB):

    """This function adds up the 4 nearest dipoles's energy, and returns th
    the current state's energy and the potential flip's energy. We have a p
    condition.
    """

    x = []
    y = []
    size = len(matrix)
    if i  == size-1: # These all account for periodic boundary
        x.append(0)
    else:
        x.append(i+1)

    if i == 0:
        x.append(-1)
    else:
        x.append(i-1)

    if j  == size-1:
        y.append(0)
    else:
        y.append(j+1)

    if j == 0:
        y.append(-1)
    else:
        y.append(j-1)

    cur_tot = 0
```

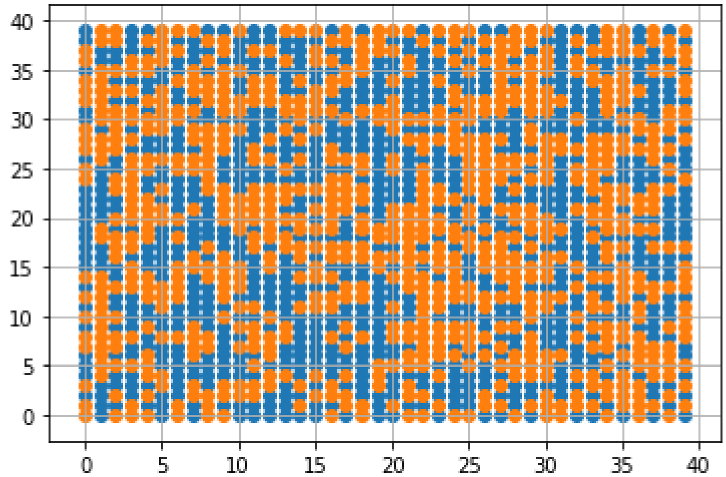
```
    flip_tot = 0
    neighbors = 0
    for t in range(2): #Go through and calculate the energy at eacb of the
        for z in range(2):
            neighbors += matrix[x[t]][y[z]]
    cur_tot = -1*((current*neighbors) + (current*uB))



    return flip_tot - cur_tot # delta U Flip - current


def prob_of_flip(current, flip, i, j, matrix, T, uB):
    """The probability of the dipole flipping. If D_U is negative,
    always flip. Otherwise flip with prob of bolt.
    factor."""

    delta = delta_u(current, flip,i, j, matrix, uB)
    if delta <= 0: # Negative always accepted
        return 1
    else:
        return np.exp(-2*delta/(T)) # from book

def change(current, flip, i, j, matrix, T, uB=0):
    """Flips dipole with the given prob above"""
    prob = prob_of_flip(current, flip, i, j, matrix, T, uB)
    if prob == 1: # negative energy, always accepted
        matrix[i][j] = flip
    else:
        matrix[i][j] = r.choices([current, flip], [1-prob, prob])[0]

def ising(size, iterations, T=1, uB=0):
    lat = lattice(size)
    for times in range(iterations ):
        row = r.randint(0, size-1)
        col = r.randint(0, size-1)
        new = flip(lat[row][col])
        change(lat[row][col], new, row, col, lat, T, uB)
    ups_x = []
    ups_y = []
    downs_x = []
    downs_y = []
    for row in range(size):
        for col in range(size):
            if lat[row][col] == 1:
                ups_y.append(row)
                ups_x.append(col)
            if lat[row][col] == -1:
                downs_x.append(col)
                downs_y.append(row)
    plt.scatter(ups_x, ups_y)
    plt.scatter(downs_x, downs_y)
    plt.grid()
    plt.show()
```

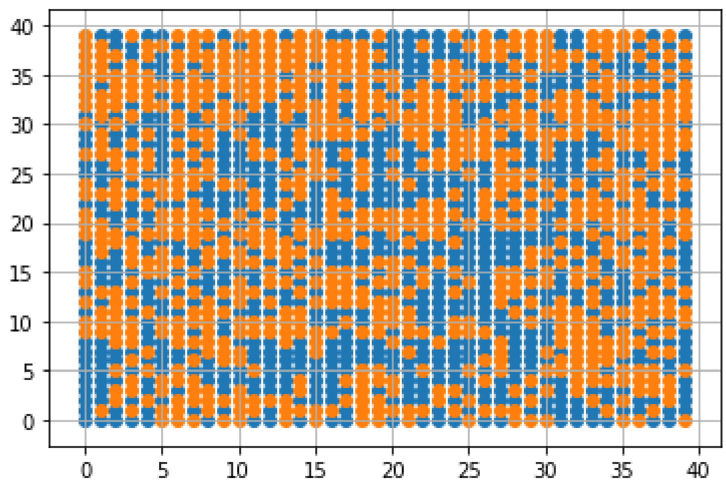
In [54]:
```
ising(40,100000, 2.5 )
```

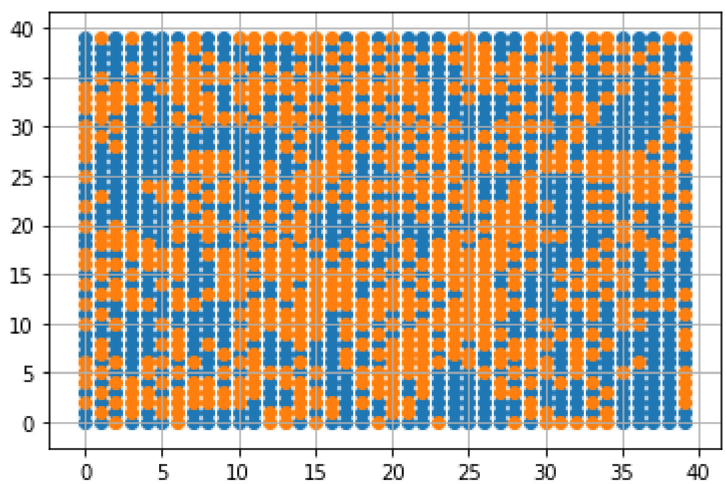


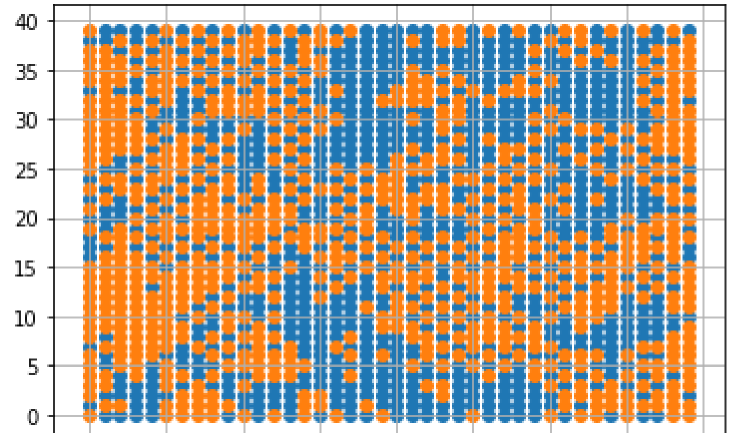
In [55]:
```
ts = [10, 5, 4, 3, 2.5]
```

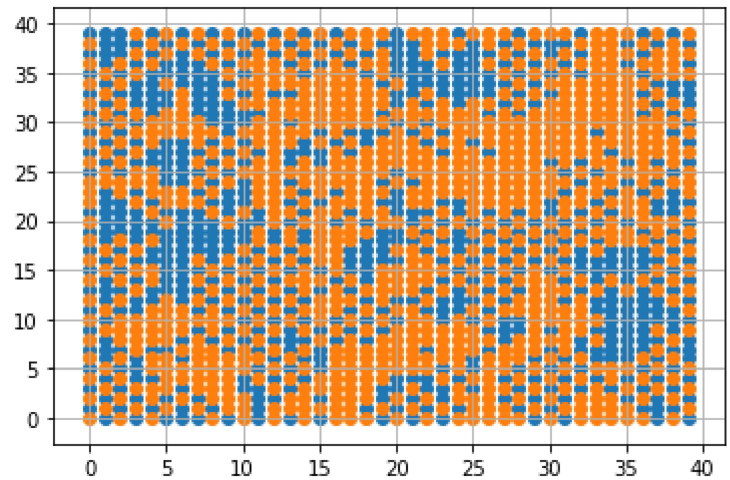
In [56]:
```
for t in ts:
    ising(40, 100000, t)
```



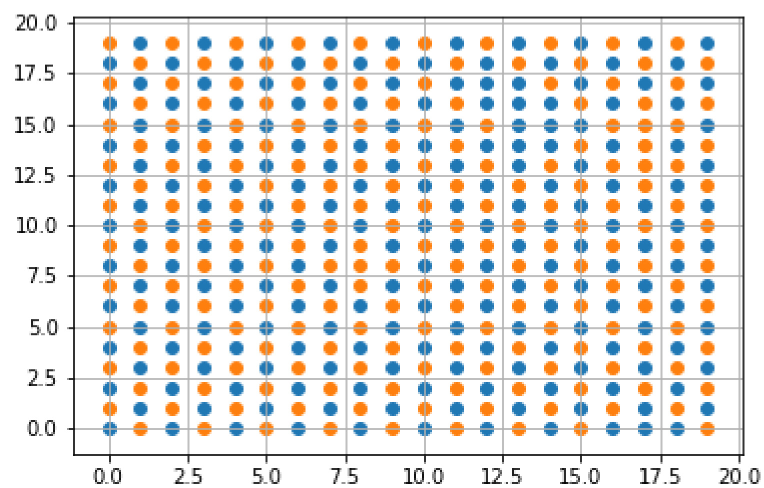




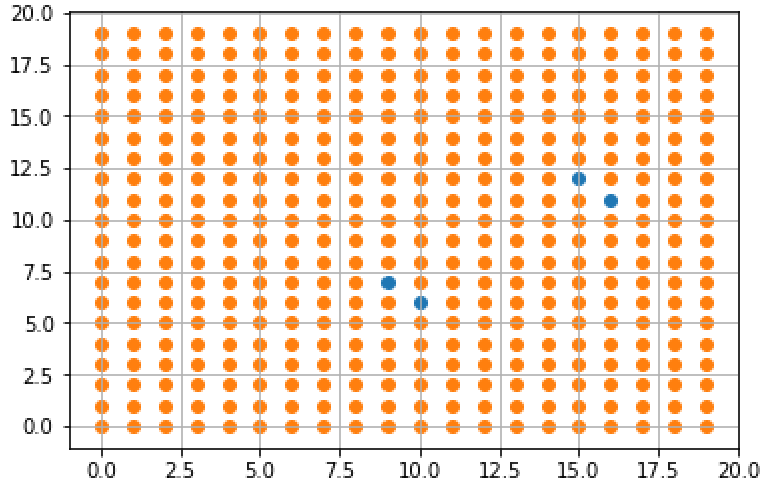




In [57]:
```
# 8.26 c

ts = [2, 1.5, 1]

ising(20, 100000, 2)
```



In [58]:
```
ising(20, 100000, 1.5)
```



In [59]:
```
ising(20, 100000)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-59-5dddf2130e7c> in <module>
----> 1 ising(20, 100000)

<ipython-input-53-01662569a31e> in ising(size, iterations, T, uB)
     92         col = r.randint(0, size-1)
     93         new = flip(lat[row][col])
---> 94         change(lat[row][col], new, row, col, lat, T, uB)
     95     ups_x = []
     96     ups_y = []

<ipython-input-53-01662569a31e> in change(current, flip, i, j, matrix, T, uB)
     84         matrix[i][j] = flip
     85     else:
---> 86         matrix[i][j] = r.choices([current, flip], [1-prob, prob])[0]
     87 
     88 def ising(size, iterations, T=1, uB=0):

~/opt/anaconda3/lib/python3.7/random.py in choices(self, population, weights, cum_weights, k)
    364         hi = len(cum_weights) - 1
    365         return [population[bisect(cum_weights, random() * total, 0, hi)]
--> 366                 for i in range(k)]
    367 
    368 ## -------------------- real-valued distributions  -------------------

~/opt/anaconda3/lib/python3.7/random.py in <listcomp>(.0)
    363         total = cum_weights[-1]
    364         hi = len(cum_weights) - 1
--> 365         return [population[bisect(cum_weights, random() * total, 0, hi)]
    366                 for i in range(k)]
    367 

KeyboardInterrupt: 
```

In [ ]:
```
# 8.26 d

ising(10, 100000, 2.5)
```

In [ ]:
```
#8.26 e

lats = [10,20,30,40,50]
ts = [2.5,2.4,2.3,2.27]
for t in ts:
    for lat in lats:
        ising(lat, 100000, t)
```

In [106]:

```
# 8.27
""" To calculate avergae energy, I will use the derivative of the ln(Z). So
I will have to keep track of the z's for each dipole, add them up, and then
I will have to retool my ising model and functions to include all of this e


def s_bar(dipole, lattice, i, j, T, uB=0):
    """Calculate average spin"""
    x = []
    y = []
    size = len(lattice)
    if i  == size-1: # These all account for periodic boundary
        x.append(0)
    else:
        x.append(i+1)

    if i == 0:
        x.append(-1)
    else:
        x.append(i-1)

    if j  == size-1:
        y.append(0)
    else:
        y.append(j+1)

    if j == 0:
        y.append(-1)
    else:
        y.append(j-1)

    spin = 0
    for t in range(2): #Go through and calculate the energy at eacb of the
        for z in range(2):
            spin += lattice[x[t]][y[z]]
    return spin


def z_t(lattice, T):
    """Calculates the partition functiob based on the mean approx theorem."
    size = len(lattice)
    z_t = 0
    for row in range(size):
        for col in range(size):
            s = s_bar(lattice[row][col], lattice, row, col, T)
            z_i = 2*np.cosh((4*s)/T) # since episilon = +- 1 here
            z_t += z_i
    return z_t

def u_bar(lattice, T):
    """calculate the avg energy as d(ln(z))/dbeta"""
    size = len(lattice)
    u_bar = 0
    for row in range(size):
        for col in range(size):
            s = s_bar(lattice[row][col], lattice, row, col, T)
```

```
            u_i = 8 * s * np.tanh(4*s/T)
            u_bar += u_i

    return u_bar


def c_v(lattice, T):
    """Calculate cv as derivative of u / t"""
    size = len(lattice)
    c_bar = 0
    for row in range(size):
        for col in range(size):
            s = s_bar(lattice[row][col], lattice, row, col, T)
            c_i = (-1*32) * (s**2) * (1/(np.cosh(4*s/T)))**2 * ((1.38*10**-
            c_bar += c_i
    return c_bar


def s_t(lattice):
    s = 0
    size = len(lattice)
    for row in range(size):
        for col in range(size):
            s+= lattice[row][col]
    return s


# find most occuring state

def finder(li):
    data_dic = {}
    for i in li:
        if i in data_dic:
            data_dic[i] += 1
        else:
            data_dic[i] = 1
    high = 0
    val = -100
    for item in data_dic:
        if data_dic[item] > val:
            high = item
            val = data_dic[item]
    return high


def ising_2(size, iterations, T):

    """returns list of [z_avg, u_avg, cv_avg]"""
    lat = lattice(size)

    u_tot = 0
    z_tot = 0 #my avg of each
    cv_tot = 0
    s_list = []

    for times in range(iterations ):
        row = r.randint(0, size-1)
```

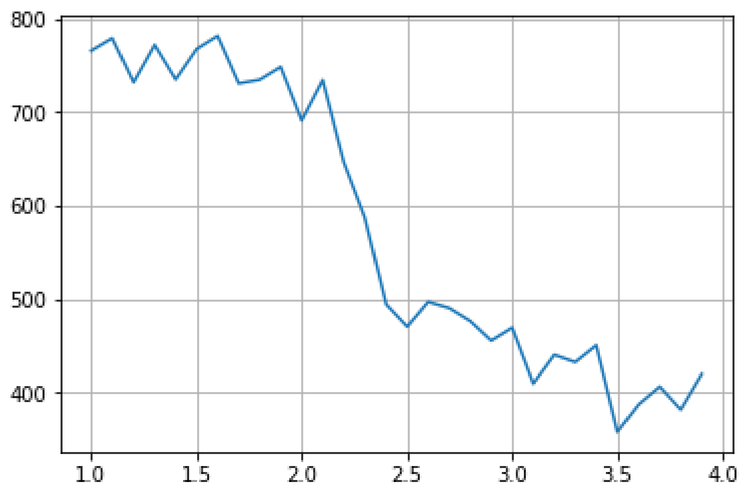
```
        col = r.randint(0, size-1)
        new = flip(lat[row][col])
        change(lat[row][col], new, row, col, lat, T)

        u_tot += u_bar(lat, T)
        z_tot += z_t(lat, T)
        cv_tot += c_v(lat, T)

        s = s_t(lat)
        s_list.append(s)

    '''
    ups_x = []
    ups_y = []
    downs_x = []
    downs_y = []
    for row in range(size):
        for col in range(size):
            if lat[row][col] == 1:
                ups_y.append(row)
                ups_x.append(col)
            if lat[row][col] == -1:
                downs_x.append(col)
                downs_y.append(row)
    plt.scatter(ups_x, ups_y)
    plt.scatter(downs_x, downs_y)
    plt.grid()
    plt.show()
    '''
    return [u_tot/ iterations, z_tot / iterations, cv_tot/ iterations, s_li
```

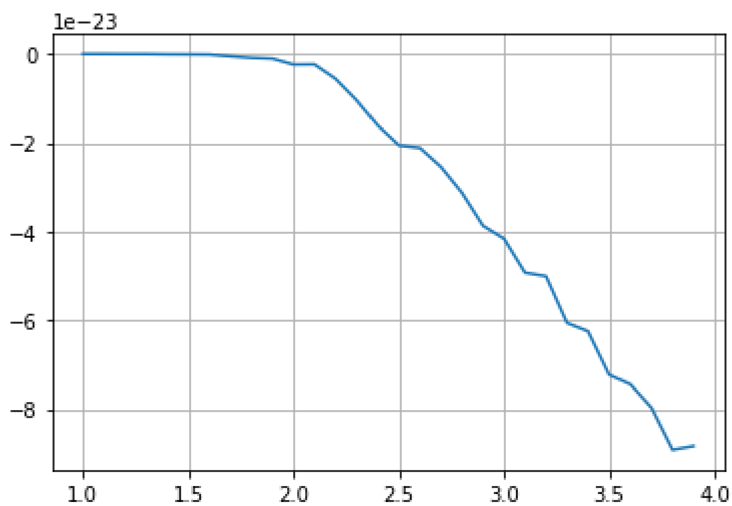
In [107]:
```
ising_2(20, 1000, 2)
```

```
 24,
 24,
 26,
 26,
 26,
 26,
 26,
 28,
 28,
 30,
 30,
 30,
 32,
 32,
 32,
 30,
 30,
 32,
 34,
 34
```

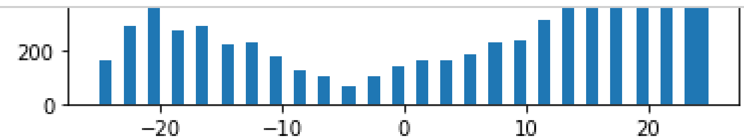
In [108]:
```
def temp(size, iterations):
    T = np.arange(1,4, 0.1)
    u_list = []
    c_list = []

    for t in T:
        i = ising_2(size, iterations, t)
        u_list.append(i[0])
        c_list.append(i[2])

    plt.plot(T, u_list)

    plt.grid()
    plt.show()


    plt.plot(T,c_list)
    plt.grid()
    plt.show()
```

In [109]:
```
temp(5, 1000)
```



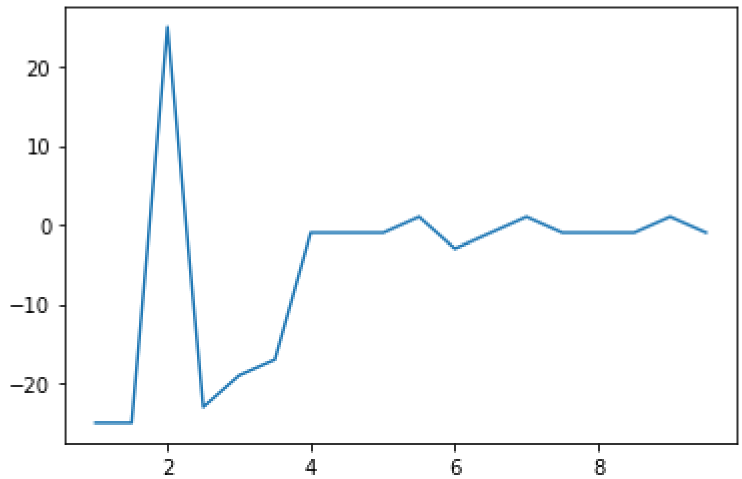


In [110]:
```
# 8.28

temps = [10, 5, 4, 3, 2.5, 2, 1]


for t in temps:

    i = ising_2(5, 10000, t)
    plt.hist(i[3], 50)
    plt.show()
```



In [111]:
```
ts = np.arange(1, 10, .5)
s_l = []
for t in ts:
    x = ising_2(5, 10000, t)[3]
    s_l.append(finder(x))

plt.plot(ts, s_l)
```

Out[111]: [<matplotlib.lines.Line2D at 0x1186d1bd0>]



In [104]:

9

In [ ]: