Project Part 2

Romtin Toranji, Andrew Nguyen, Matthew Coleman

(a)

In [360]:

```
import numpy as np
import pandas as pd

file=np.load("part2.npz")
beta_old=file['beta_old']
N=file['N']
Svc_0_PMF=file['Svc_0_PMF']
Lc=file['Lc']
Ic_0=file['Ic_0']
gamma=file['gamma']
L_observed=file['L_observed']
nb_nodes = Svc_0_PMF.shape[0]
```

In [250]:

```
def SIR(t, Z, N, betas, gamma):
    (S_vec, I_vec, R) = (Z[:16].reshape(4, 4), Z[16:20], Z[-1])
    I = I_vec.sum()
    dSdt = -betas*S_vec*I/N
    dIdt = -dSdt.sum(axis=0)-gamma*I_vec
    dRdt = np.array([gamma*I])
    return np.concatenate((dSdt.flatten(), dIdt, dRdt))
```

In [251]:

```
from scipy.integrate import solve_ivp
from scipy.optimize import fmin
from sklearn.metrics import mean_squared_error

Svc_0_PMF /= Svc_0_PMF.sum(axis=(1, 2), keepdims=True)
Svc_0 = (N-Ic_0.sum(axis=1))[:, np.newaxis, np.newaxis]*Svc_0_PMF
Svc_0 = Svc_0.round().astype(int)
R0 = np.zeros((nb_nodes)).astype(int)

t = np.linspace(0, 199, 200)

S = np.zeros((nb_nodes, 4, 4, len(t)))
I = np.zeros((nb_nodes, 4, len(t)))
R = np.zeros((nb_nodes, len(t)))
L = np.zeros((nb_nodes, len(t)))

for N_node, node_index in zip(N, range(nb_nodes)):

    initial_conditions = np.concatenate(
        (Svc_0[node_index].flatten(), Ic_0[node_index], [R0[node_index]]))

    soln = solve_ivp(
        SIR, (t[0], t[9]), initial_conditions, args=(N_node, beta_old, gamma), t_eval=

    S[node_index, :, :, :10] = soln.y[:16].reshape(4, 4, -1)
    I[node_index, :, :10] = soln.y[16:20]
    R[node_index, :10] = soln.y[20]
    L[node_index, :10] = Lc@soln.y[16:20]

alpha_predicted = np.zeros((nb_nodes))

def loss_func(alpha):

    ###### WRITE YOUR FUNCTION HERE

    soln_obj=solve_ivp(SIR, (t[9], t[29]), np.concatenate((S[node_index, :, :, 9:10].f
                                                           I[node_index, :, 9:10].flat
                       , args=(N_node, beta_old*alpha, gamma), t_eval=t[9:29])
    '''
    check the value below I'm not sure how to calculate it
    '''

    Lpred = Lc@soln_obj.y[16:20]

    mse = mean_squared_error(L_observed[node_index][9:29].flatten(), (Lpred).flatten()



    return mse
```

```
for N_node, node_index in zip(N, range(nb_nodes)):
    alpha_predicted[node_index] = fmin(loss_func, 0.5, disp=False)
```

In [252]:

```
#Get the values for days 9-29 with alpha_predicted

for N_node, node_index in zip(N, range(nb_nodes)):

    initial_conditions = np.concatenate((S[node_index, :, :, 9].flatten(),
                                         I[node_index, :, 9].flatten(), [R[node_index,

    soln=solve_ivp(SIR, (t[9], t[29]), initial_conditions, args=(N_node, beta_old*alph

    #soln = solve_ivp(
    #    SIR, (t[0], t[29]), initial_conditions, args=(N_node, beta_old, gamma), t_eva

    S[node_index, :, :, 10:30] = soln.y[:16].reshape(4, 4, -1)
    I[node_index, :, 10:30] = soln.y[16:20]
    R[node_index, 10:30] = soln.y[20]
    L[node_index, 10:30] = Lc@soln.y[16:20]
```

In [253]:

```
#Predicted SIR values for days 29-200 using alpha_predicted

for N_node, node_index in zip(N, range(nb_nodes)):

    S_pred = S[:]
    I_pred = I[:]
    R_pred = R[:]
    L_pred = L[:]

    initial_conditions = np.concatenate(
        (S[node_index,:,:,29].flatten(), I[node_index,:,29], [R[node_index,29]]))

    soln = solve_ivp(
        SIR, (t[29], t[199]), initial_conditions, args=(N_node, alpha_predicted[node_i

    S_pred[node_index, :, :, 30:200] = soln.y[:16].reshape(4, 4, -1)
    I_pred[node_index, :, 30:200] = soln.y[16:20]
    R_pred[node_index, 30:200] = soln.y[20]
    L_pred[node_index, 30:200] = Lc@soln.y[16:20]
```

(b)

In [254]:

```
def g(money):
    alpha = 1/np.log2(2*10**(-3)*money+2)
    return alpha

def g_inv(alpha):
    money = (2**(1/alpha)-2)/(2*10**(-3))
    return money

import matplotlib.pyplot as plt

def scatter_plot(alpha_afterNPI):
    plt.figure()
    colors=(beta_old[np.newaxis,:,:]*Svc_0_PMF).mean(axis=(1,2))
    plt.scatter(alpha_predicted, alpha_afterNPI, s=N/500, c=colors, cmap='jet', alpha=
    plt.xlabel(r"$\alpha_{own}$")
    plt.ylabel(r"$\alpha_{after\ NPI}$")

S_after = np.zeros((nb_nodes, 4, 4, len(t)))
I_after = np.zeros((nb_nodes, 4, len(t)))
R_after = np.zeros((nb_nodes, len(t)))
L_after = np.zeros((nb_nodes, len(t)))

def total_recovered_difference(alpha_afterNPI):

    S_after = np.zeros((nb_nodes, 4, 4, len(t)))
    I_after = np.zeros((nb_nodes, 4, len(t)))
    R_after = np.zeros((nb_nodes, len(t)))
    L_after = np.zeros((nb_nodes, len(t)))

    for N_node, node_index in zip(N, range(nb_nodes)):

        initial_conditions = np.concatenate(
            (S[node_index,:,:,29].flatten(), I[node_index,:,29], [R[node_index,29]]))

        soln_after = solve_ivp(
            SIR, (t[29], t[199]), initial_conditions, args=(N_node, alpha_afterNPI[nod

        S_after[node_index, :, :, 30:200] = soln_after.y[:16].reshape(4, 4, -1)
        I_after[node_index, :, 30:200] = soln_after.y[16:20]
        R_after[node_index, 30:200] = soln_after.y[20]
        L_after[node_index, 30:200] = Lc@soln_after.y[16:20]

    difference = np.sum(R_pred[:,-1] - R_after[:,-1])



    return difference
```

In [239]:

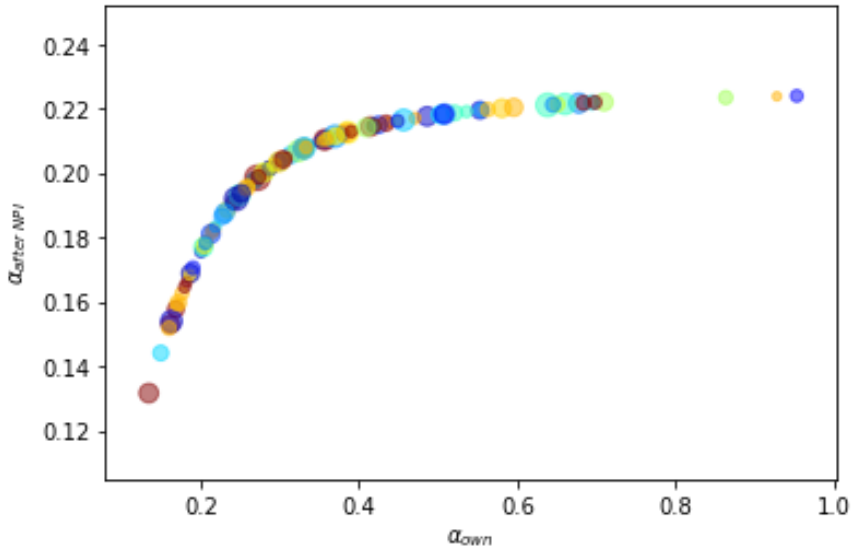
```
# Here is getting the Solution with alpha_afterNPI

#S_after = np.zeros((nb_nodes, 4, 4, len(t)))
#I_after = np.zeros((nb_nodes, 4, len(t)))
#R_after = np.zeros((nb_nodes, len(t)))
#L_after = np.zeros((nb_nodes, len(t)))


#alpha_afterNPI = g(g_inv(alpha_predicted) + 10000)

#for N_node, node_index in zip(N, range(nb_nodes)):

#    initial_conditions = np.concatenate(
#        (S[node_index,:,:,29].flatten(), I[node_index,:,29], [R[node_index,29]]))

#    soln_after = solve_ivp(
#        SIR, (t[29], t[199]), initial_conditions, args=(N_node, alpha_afterNPI[node_i

#    S_after[node_index, :, :, 30:200] = soln_after.y[:16].reshape(4, 4, -1)
#    I_after[node_index, :, 30:200] = soln_after.y[16:20]
#    R_after[node_index, 30:200] = soln_after.y[20]
#    L_after[node_index, 30:200] = Lc@soln_after.y[16:20]

#difference = np.sum(R_pred[:,-1] - R_after[:,-1])
```

Policy #1

i.

In [260]:

```
alpha_equal = g(g_inv(alpha_predicted) + 10000)
scatter_plot(alpha_equal)
```



We can see when we have an equal distribution of non-pharmaceutical interventions, it appears that NPI has a bigger effect on the larger α values than the smaller ones. It seems the relationship between α_{own} and $\alpha_{after\ NPI}$ seems to be logarithmic. It also seems the equal distribution of funds makes is so some very small populations have large decreases in the α value, while the bigger populations do not.

ii.

In [259]:

```
total_recovered = total_recovered_difference(alpha_equal)
print(f'The Difference in the Number of people who succumbed to the virus after 200 da
```

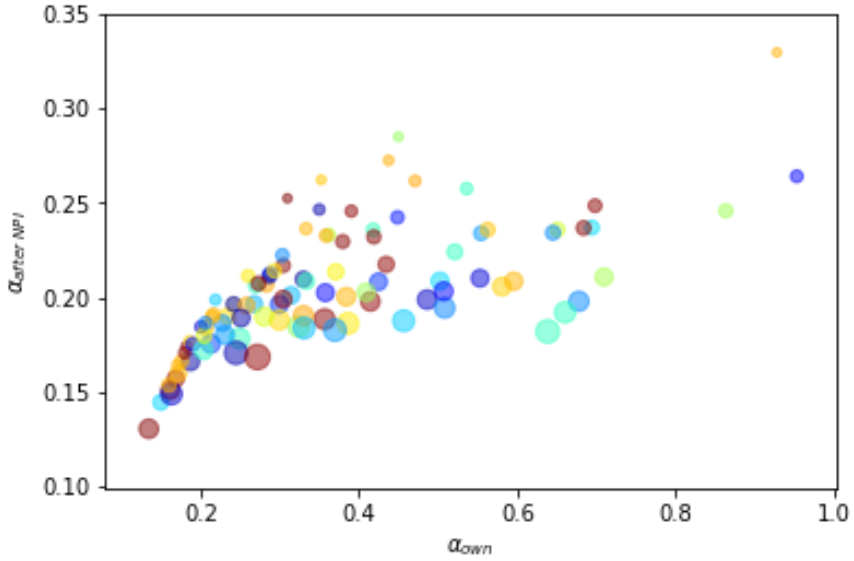
```
The Difference in the Number of people who succumbed to the virus after
200 days when using equal NPI funding is approximately 320440
```

Policy #2

i.

In [243]:

```
population_prop = N/np.sum(N)
money_allocation1 = population_prop*1000000
alpha_prop = g(g_inv(alpha_predicted) + money_allocation1)
scatter_plot(alpha_prop)
```



As we can see from this plot, there is still a semi-logarithmic distribution of the points. As we can also see, many of the bigger population nodes seem to have had a greater decrease in the α value than the smaller nodes. this would make sense, as we would try and put more money to stop it from spreading faster in these areas. Because there is more weight put on these bigger populations, out scatterplot shows to be more flat then if we provide equal funding.

ii.

In [244]:

```
total_recovered = total_recovered_difference(alpha_prop)
print(f'The Difference in the Number of people who succumbed to the virus after 200 da
```

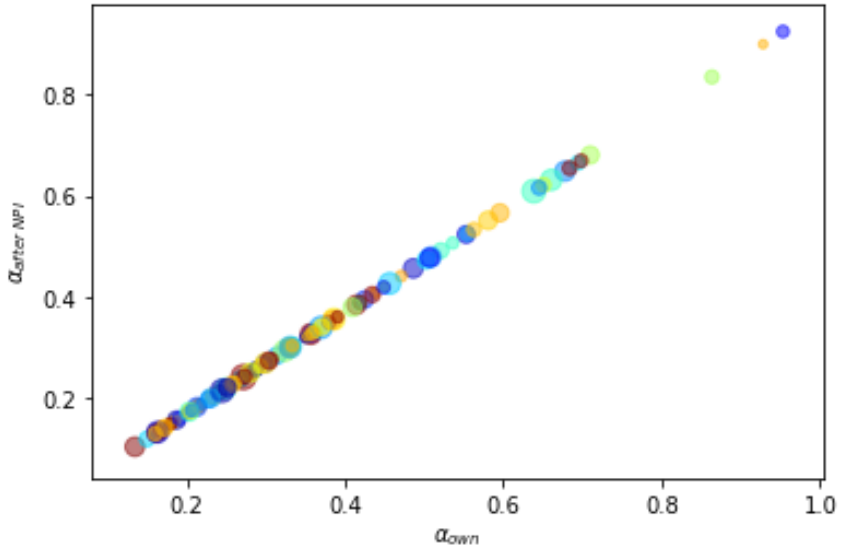
```
The Difference in the Number of people who succumbed to the virus after
200 days when using equal NPI funding is approximately 335994
```

Policy #3

i.

In [255]:

```
def delta_a(alpha):

    alpha_after = alpha_predicted-alpha
    money_total = g_inv(alpha_after)-g_inv(alpha_predicted)
    dif = abs(1000000 - sum(money_total))



    return dif


change_a = fmin(delta_a, 0.1, disp=False)
scatter_plot((alpha_predicted-change_a))
```



In [346]:

```
money_allocation3 = g_inv(alpha_predicted-change_a)-g_inv(alpha_predicted)
```

In [256]:

```
npi_pol_3 = g_inv(alpha_predicted-change_a)-g_inv(alpha_predicted)
```

In [257]:

```
print((alpha_predicted-change_a))
```

```
[0.20411617 0.35659954 0.19799282 0.68035061 0.64847805 0.17743923
 0.3281625  0.32717067 0.35006191 0.1576791  0.25343105 0.30025722
 0.28480312 0.18682644 0.10496761 0.2121957  0.17091152 0.52355434
 0.55136501 0.15280849 0.24007656 0.27497036 0.45676357 0.2802621
 0.30414821 0.1391122  0.34019863 0.18468105 0.44150478 0.63109981
 0.26998378 0.27380459 0.35453121 0.14671718 0.5662698  0.60907985
 0.36092617 0.14145595 0.25859462 0.83416835 0.38505336 0.40815517
 0.3953561  0.15662929 0.21953518 0.40500576 0.6649789  0.29441616
 0.21523525 0.18918544 0.1201913  0.18343593 0.30047695 0.23817226
 0.15020839 0.62151882 0.16085292 0.49167873 0.24229366 0.52482692
 0.33266689 0.25721523 0.32042324 0.38846828 0.23051235 0.47890102
 0.17389919 0.38935634 0.42707302 0.13218242 0.13363125 0.30133754
 0.26364834 0.42055751 0.3798196  0.26989223 0.5333871  0.61574489
 0.34131252 0.17479335 0.30372707 0.47298977 0.66895229 0.32903835
 0.25064937 0.27482235 0.24361049 0.5066415  0.92386562 0.4776498
 0.14215175 0.32293789 0.22857448 0.20102016 0.4193124  0.17672207
 0.13056118 0.89881679 0.22179042 0.65441372]
```

ii.

In [258]:

```
total_recovered = total_recovered_difference(alpha_predicted-change_a)
print(f'The Difference in the Number of people who succumbed to the virus after 200 da
```

```
The Difference in the Number of people who succumbed to the virus after
200 days when using equal change in alpha is approximately 62743
```

Policy #4

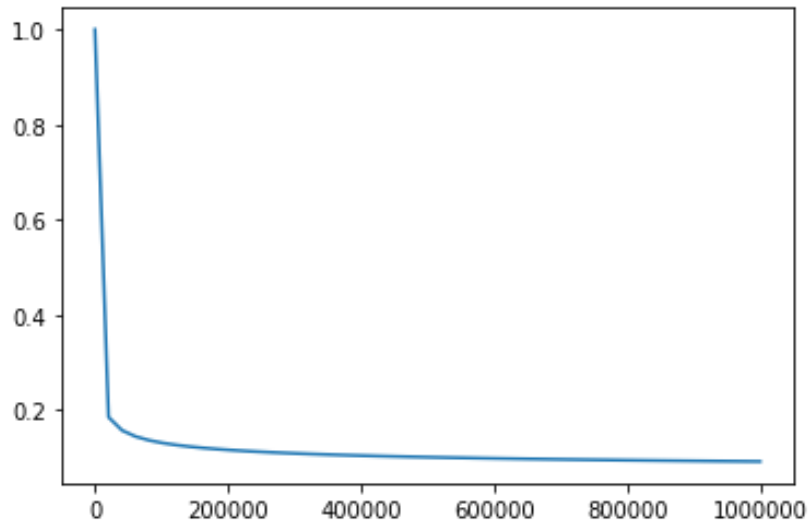
i.

In [196]:

```
x = np.linspace(0,1000000)
plt.plot(x, g(x))
```

Out[196]:

```
[<matplotlib.lines.Line2D at 0x25d9aaa4c48>]
```



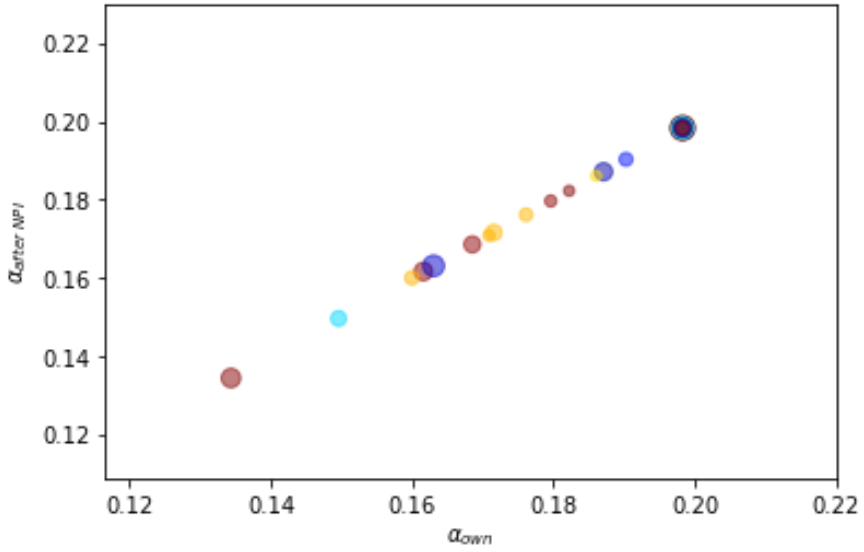
As you see above the maximum change per dollar is when alpha is at its greatest. Therefore in order to maximize the change per dollar, it would be best to find the greatest alpha value for every dollar and give that node the dollar.

In [220]:

```
#maximize the minimization of alpha per dollar
#in order to find the maximum change in alpha we need to find the maximum alpha values
#
import operator
alpha_news = alpha_predicted
money_allocation = alpha_predicted*0
for i in range (1000000):
    index = np.argmax(alpha_news)
    value = alpha_news[index]
    alpha_news[index] = g(g_inv(value)+ 1)
    money_allocation[index] = money_allocation[index]+ 1
```

In [221]:

```
scatter_plot(alpha_news)
```

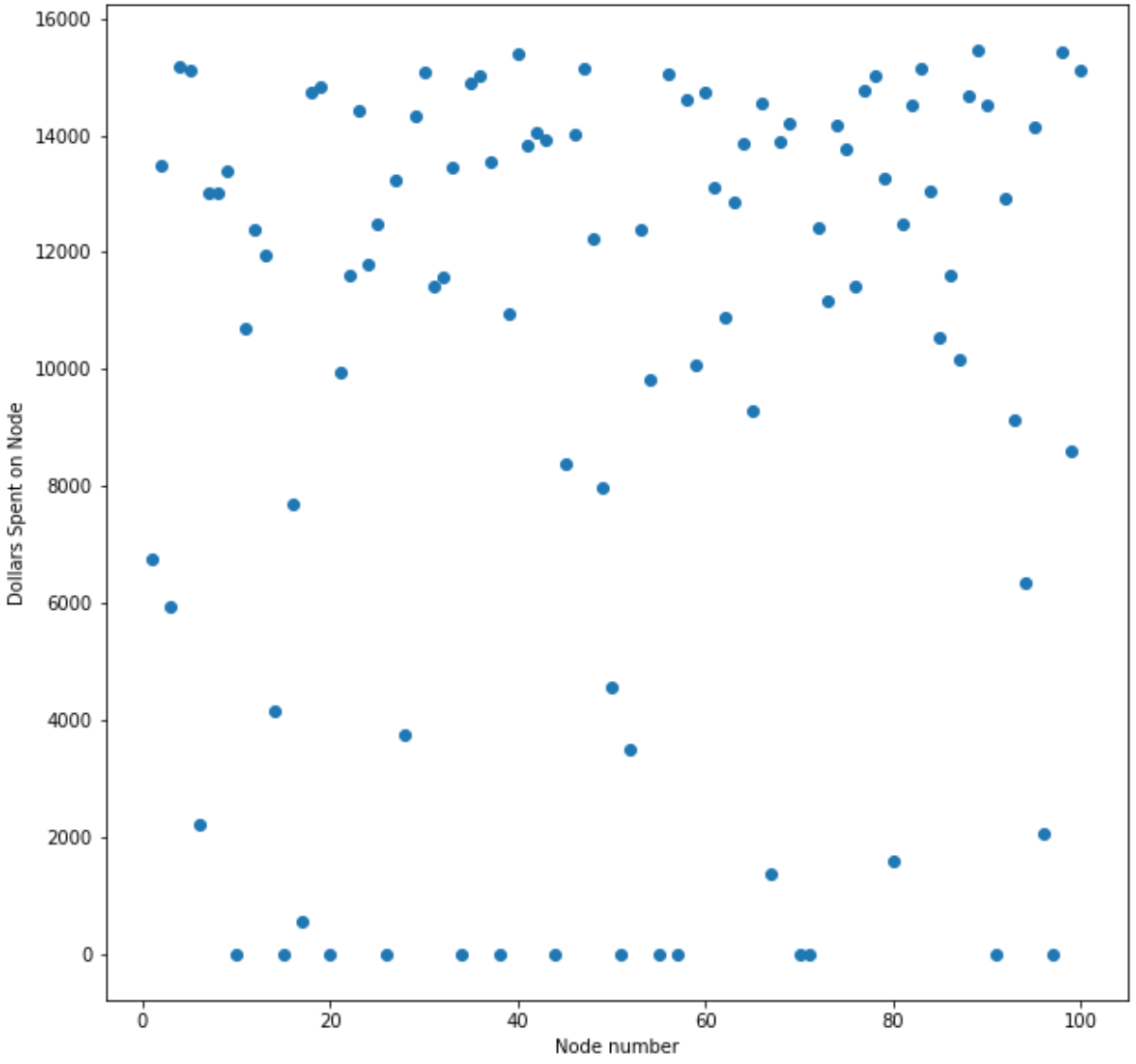


In [222]:

```
a_list  =list(range(1, 101))
plt.figure(figsize=(10,10))
plt.xlabel("Node number")
plt.ylabel("Dollars Spent on Node")
plt.scatter(a_list,money_allocation)
```

Out[222]:

```
<matplotlib.collections.PathCollection at 0x25daad859c8>
```



ii.

In [223]:

```
total_recovered = total_recovered_difference(alpha_news)
print(f'The Difference in the Number of people who succumbed to the virus after 200 da
```

```
The Difference in the Number of people who succumbed to the virus after
200 days when maximizing change in alpha per dollar is approximately 33
0597
```

(c)

Fairness Metric

Since nothing is known about the lifestyles of the nodes, their characteristics, cultures, etc. we can not make any individual assumptions about them. Therefore, we must assume that the given parameters happen to be randomly assigned. Under this assumption, we will be looking at percentage of the node who have been sick. By assigning all nodes as equals, we will not be overlooking nodes in the minority. In order to do so, we need to calculate the percentage of each node that gets sick and find the variance over all nodes.

In [232]:

```
def recovered_difference_per(alpha_afterNPI):

    S_after = np.zeros((nb_nodes, 4, 4, len(t)))
    I_after = np.zeros((nb_nodes, 4, len(t)))
    R_after = np.zeros((nb_nodes, len(t)))
    L_after = np.zeros((nb_nodes, len(t)))
    difference = np.zeros((nb_nodes, len(t)))

    for N_node, node_index in zip(N, range(nb_nodes)):

        initial_conditions = np.concatenate(
            (S[node_index,:,:,29].flatten(), I[node_index,:,29], [R[node_index,29]]))

        soln_after = solve_ivp(
            SIR, (t[29], t[199]), initial_conditions, args=(N_node, alpha_afterNPI[nod
            t_eval=t[30:200])

        S_after[node_index, :, :, 30:200] = soln_after.y[:16].reshape(4, 4, -1)
        I_after[node_index, :, 30:200] = soln_after.y[16:20]
        R_after[node_index, 30:200] = soln_after.y[20]
        L_after[node_index, 30:200] = Lc@soln_after.y[16:20]

    difference = (R_pred[:,-1] - R_after[:,-1])/N



    return difference
```

Policy 1

In [269]:

```
import statistics
print("The variance in infected percentage of population accross nodes using equal spe
print(statistics.variance(recovered_difference_per(alpha_equal)))
```

```
The variance in infected percentage of population accross nodes using e
qual spending is:
0.011964932620521923
```

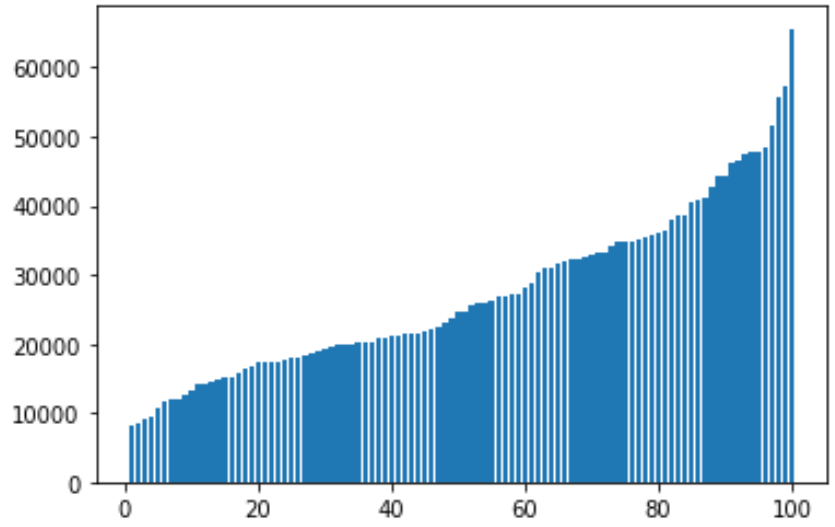
The problem with this method is that we are ignoring groups with large amounts of people. In reality, cities like New York and Los Angeles will get very little funding in comparison to the vast amounts of people that live in them. However, small towns and villages will get a lot of funding. Potentially, these towns can be safe havens from disease. However, we can in essence overfund the small towns. If a town where everyone knows each other gets a ton of money, the town may not know what to do with the money. Possibly leading to inflation in the town and other economical consequences.

In [359]:

```
N_sort = N.sort()
plt.bar(list(range(1,101)),N)
```

Out[359]:

```
<BarContainer object of 100 artists>
```



From the graph above, we can see that the biggest population is over 6x the population of the smallest. With funds being the same, large groups are being disadvantaged.

Policy 2

In [270]:

```
print("The variance in infected percentage of population accross nodes using spending
print(statistics.variance(recovered_difference_per(alpha_prop)))
```

```
The variance in infected percentage of population accross nodes using s
pending proportional to population is:
0.013234255648944995
```

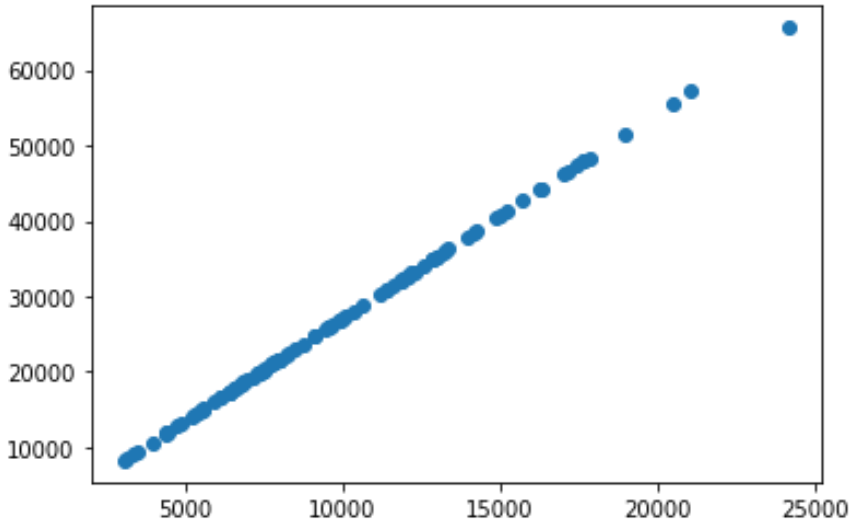
The policy is good in that it provides every node with capital equitable to their number of constituents. The math is fairly simple and can lead to transparency among civilians. However, the relationship in funding needed to population is not necessarily linear. Take for example a grocery trip to Costco or your local grocery store. If you are looking to buy an individual apple, it would be cheaper to buy at the local grocery store instead of Costco. This is a result of Costco requiring you to buy apples in bulk. However when making a large purchase of apples, Costco will be cheaper due to its bulk prices. This relationship can be extrapolated to humans. It is cheaper per person to stop the spread of a large group than a small one. Therefore, it would not be best to have a linear relationship for funding.

In [370]:

```
population_prop = N/np.sum(N)
money_allocation1 = population_prop*1000000
money_allocation1.sort()
N.sort()
plt.scatter(money_allocation1,N)
```

Out[370]:

```
<matplotlib.collections.PathCollection at 0x25da3a17cc8>
```



This graph shows a linear relationship between spending and population. While this may be good in theory, the Costco example will tell us otherwise.

Policy 3

In [335]:

```
print("The variance in infected percentage of population accross node using equal chan
print(statistics.variance(recovered_difference_per(alpha_predicted-change_a)))
```

```
The variance in infected percentage of population accross node using eq
ual change in alphais:
0.0002661461203192 84
```

This policy is good in that it is able change alpha the same accross nodes. Therefore, diversity of nodes is not penalized. However, some nodes will receive much more funding than others. While the difference is not as big as with policy 4, people will still be upset. Also, the relationship is not transparent to citizens

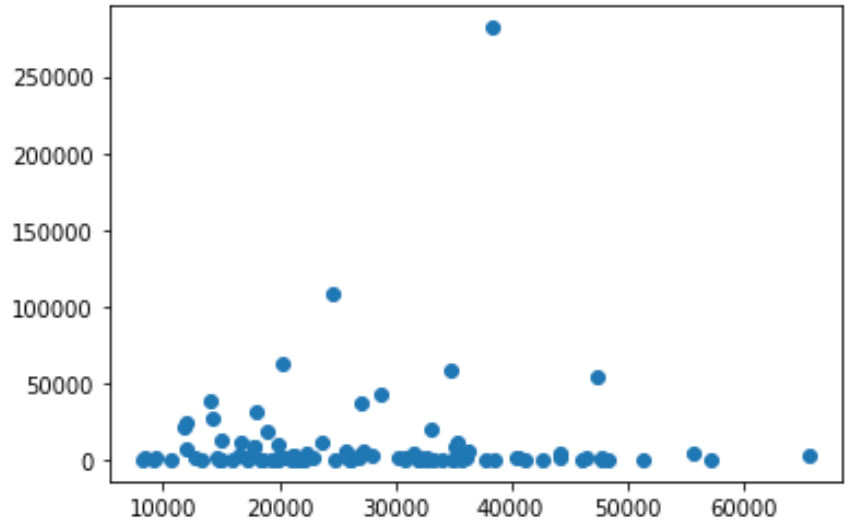
in that it is difficult to understand. In addition, this policy incentivizes malpractice. Nodes will notice that having higher alpha values will recieve more funding. While in a utopian society everything would be perfect, in the real world people will manipulate the alphas to their advantage.

In [350]:

```
plt.scatter(N,money_allocation3)
```

Out[350]:

```
<matplotlib.collections.PathCollection at 0x25da178f788>
```



From the graph above, we can see that one node received significantly more money than the others. Also, that node was in the middle of the road in terms of population. Therefore, the spending did not affect the greatest number of people.

Policy 4

In [336]:

```
print("The variance in infected percentage of population accross node using maximizing
print(statistics.variance(recovered_difference_per(alpha_news)))
```

```
The variance in infected percentage of population accross node using ma
ximizing the total decrease in alphais:
0.013712974928626427
```

The best part of this policy is that it maximizes the negative change in alpha per dollar. Overall, alphas accross the board will decrease the most. In additon, since there is no way to know how long it will take a pandemic to run its course, it is best to minimize alpha as much as we can. If the disease ravages for a relatively short period, we can maximize the amount of non-infected. However if the disease lasts for a

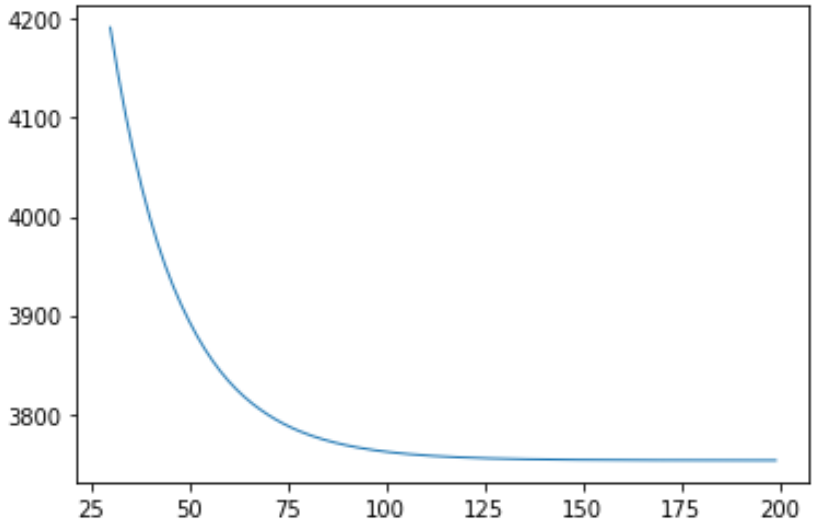
while, this policy may be utterly useless. Most of the alphas would not have changed enough to converge to a non-zero number of uninfected people. Also, as with policy 3, this practice incentivizes manipulating alphas in order to receive funding.

In [334]:

```
def total_recovered_difference_graph(alpha_afterNPI):

    S_after = np.zeros((nb_nodes, 4, 4, len(t)))
    I_after = np.zeros((nb_nodes, 4, len(t)))
    R_after = np.zeros((nb_nodes, len(t)))
    L_after = np.zeros((nb_nodes, len(t)))

    for N_node, node_index in zip(N, range(nb_nodes)):

        initial_conditions = np.concatenate(
            (S[node_index,:,:,29].flatten(), I[node_index,:,29], [R[node_index,29]]))

        soln_after = solve_ivp(
            SIR, (t[29], t[199]), initial_conditions, args=(N_node, alpha_afterNPI[nod
            t_eval=t[30:200])

        S_after[node_index, :, :, 30:200] = soln_after.y[:16].reshape(4, 4, -1)
        I_after[node_index, :, 30:200] = soln_after.y[16:20]
        R_after[node_index, 30:200] = soln_after.y[20]
        L_after[node_index, 30:200] = Lc@soln_after.y[16:20]
        if(node_index == 88):
            plt.plot(t[30:200], np.sum(soln_after.y[0:4], axis = 0), linewidth=1)


    difference = np.sum(R_pred[:,-1] - R_after[:,-1])



    return difference
total_recovered_difference_graph(alpha_news)
```

Out[334]:

```
330596.549264804
```



The graph above shows the plot for the node with the most money spent on it. As shown by the graph above, a solution must be found within the first 100 or so days for this method to work. The faster, the more lives saved. Otherwise, the number of susceptible converge to zero, making this effort a waste of money.

The fairest policy

In my opinion the fairest policy would be to do a combination of policy 1 and 2. Therefore, minority groups are not neglected and most of the money is not going to the majority.

In [ ]: