1. In order to perform this operation in a two layer perceptron network, with signum function as activation function, we need to find out the appropriate weights for a 2-input AND gate, 3-input AND gate and 2-input OR gate. We do this by forming the inequality equations using the truth table.

AND GATE

X_1	X_2	$X_1 X_2$
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

when $X_1 = X_2 = -1$ $y = Sgn(W_0 - W_1 - W_2) = -1$

$\Rightarrow W_0 - W_1 - W_2 < 0 \Rightarrow W_0 < W_1 + W_2$ — (1)

Similarly, when $X_1 = -1, X_2 = 1$, we get $W_0 + W_2 < W_1$ — (2)

$X_1 = 1, X_2 = -1$, we get $W_0 + W_1 < W_2$ — (3)

$X_1 = X_2 = 1$, we get $W_0 + W_1 + W_2 > 0$ — (4)

Following values satisfy the above inequalities; $W_0 = -5$, $W_1 = 4$ and $W_2 = 4$

To make a 3-input AND gate, we extend this further and equate W_3 to 4 ie $W_3 = 4$

OR GATE

X_1	X_2	$X_1 + X_2$
-1	-1	-1
-1	1	1
1	-1	1
1	1	1

when $X_1 = X_2 = -1$ $y = Sgn(W_0 - W_1 - W_2) = -1$

$\Rightarrow W_0 - W_1 - W_2 < 0 \Rightarrow W_0 < W_1 + W_2$ — (1)

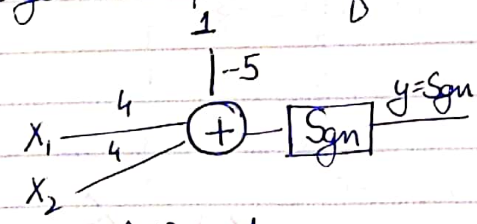
Similarly, when $X_1 = -1, X_2 = 1$, we get $W_0 + W_2 > W_1$ — (2)

$X_1 = 1, X_2 = -1$, we get $W_0 + W_1 > W_2$ — (3)

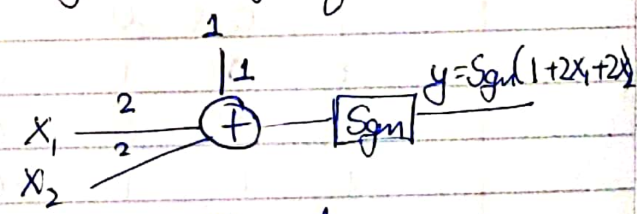
$X_1 = X_2 = 1$, we get $W_0 + W_1 + W_2 > 0$ — (4)

Following values satisfy the above inequalities; $W_0 = 1$, $W_1 = 2$, $W_2 = 2$

To get the compliment of an input we will just negate the weight for that input.

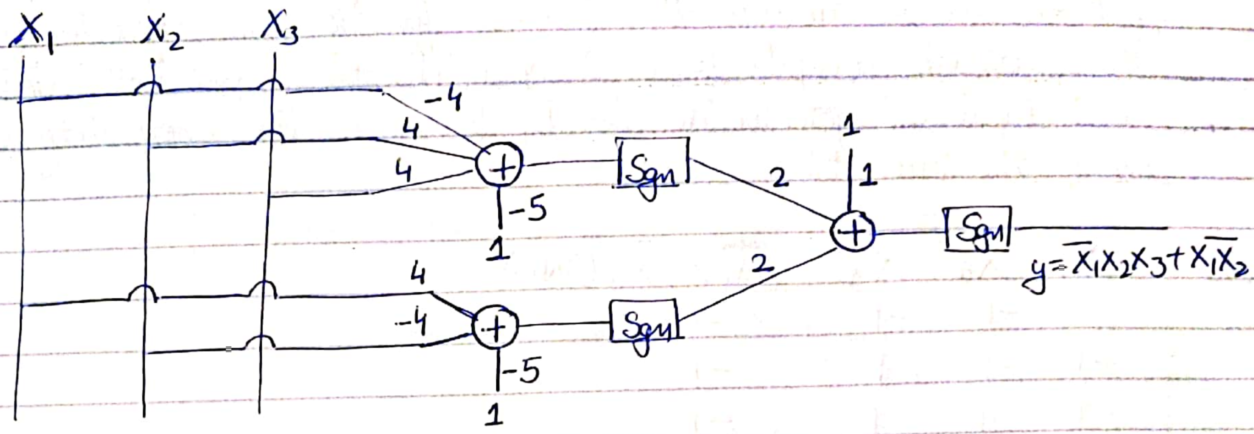


AND gate



OR gate

The 2-layered perceptron network to give an output of $\bar{X}_1 X_2 X_3 + X_1 \bar{X}_2$ is:



2. Let the inputs for second layer of neurons be, Y_1, Y_2 & Y_3.

$\therefore$ $z = u(w_0 + w_1 Y_1 + w_2 Y_2 + w_3 Y_3)$

For $z = 1$

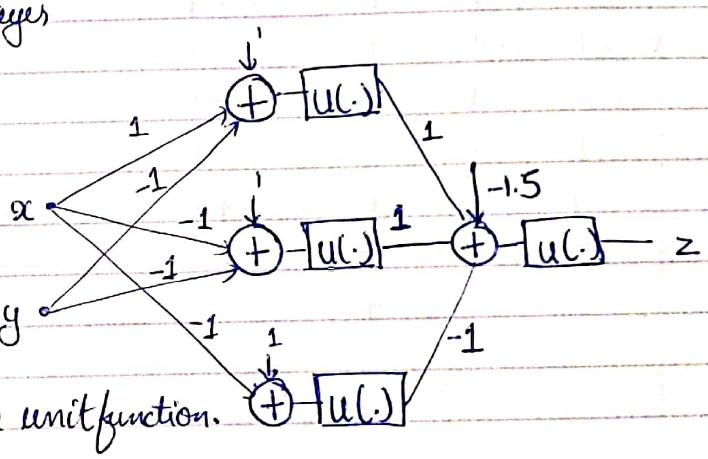
$1 = u(-1.5 + Y_1 + Y_2 - Y_3)$

$\Rightarrow -1.5 + Y_1 + Y_2 - Y_3 \geq 0$

This is possible if and only if

$Y_1 = 1$, $Y_2 = 1$ & $Y_3 = 0$

$\because$ Y_1, Y_2 & Y_3 are an output of a unit function.



For the neurons in first layer.

$Y_1 = 1 = u(1 + x - y)$

$\Rightarrow 1 + x \geq y$ — ①

$Y_2 = 1 = u(1 - x - y)$

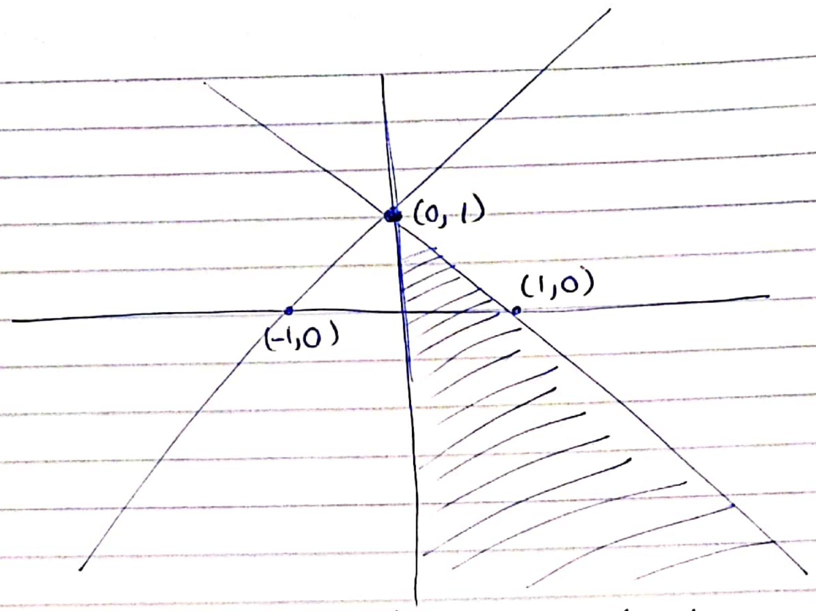
$\Rightarrow 1 \geq x + y$ — ②

$Y_3 = -1 = u(-x)$

$\Rightarrow -x < 0$ or $x > 0$ — ③

When we plot ①, ② & ③ on the coordinate axis, we get

NOT gate.



The shaded region is where we will get a value of $z=1$ as per the given network. The points on the y-axis will not give an output of 1.

In [79]:
```
import numpy as np
import matplotlib.pylab as plt
```

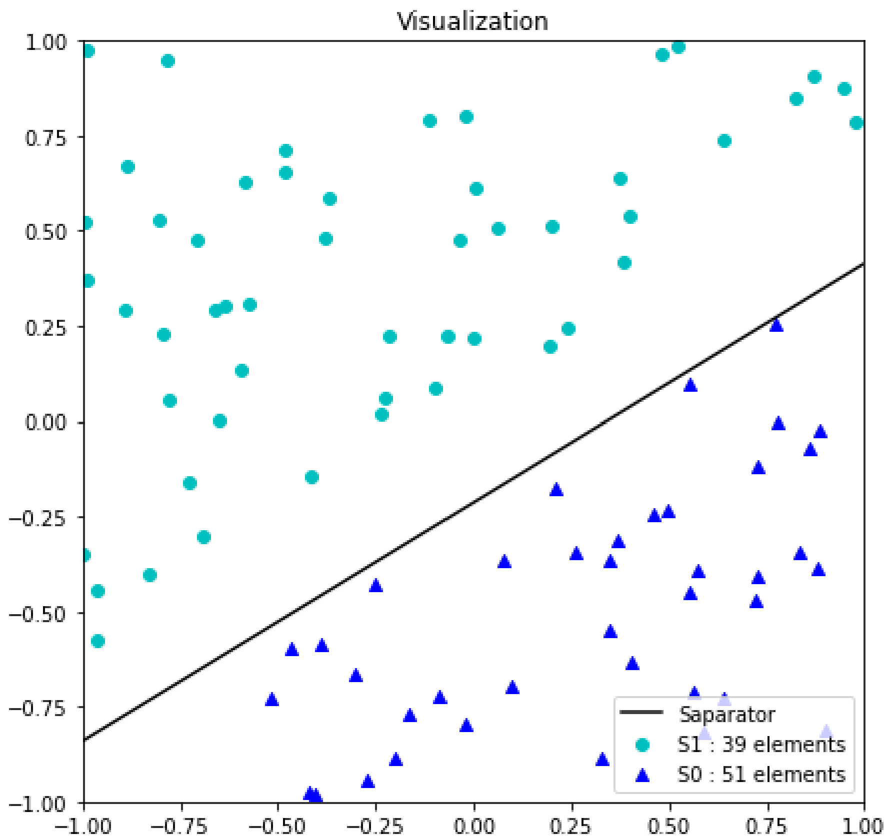
In [80]:
```
w0 = np.random.uniform(-1/4, 1/4)
w1 = np.random.uniform(-1, 1)
w2 = np.random.uniform(-1, 1)
original_omega = [w0, w1, w2]
print('Weights are: ', original_omega)
```

```
Weights are:  [-0.1384237500991744, 0.4070161866245532, -0.648870514784301]
```

Above are the Weights randomly and uniformly generated.

In [81]:
```
S = 2 * np.random.rand(100,2) - 1
S0 = []
S1 = []
for i in S:
    if (1*w0)+(i[0]*w1)+(i[1]*w2) >= 0:
            S1.append([i[0]] + [i[1]] + [0])
    elif (i[0]*w1)+(i[1]*w2) < 0:
            S0.append([i[0]] + [i[1]] + [1])
dataset = S0 + S1
```

In [82]:
```
x1 = -(w0-w2)/w1
x2 = -(w0+w2)/w1
X = np.array([x1, x2])
Y = np.array([-1.0, +1.0])

S1_x = []
S1_y = []
S0_x = []
S0_y = []

for i in S0:
    S0_x.append(i[0])
    S0_y.append(i[1])
for i in S1:
    S1_x.append(i[0])
    S1_y.append(i[1])

fig, ax = plt.subplots(figsize=(7,7))
blue = plt.scatter(S0_x, S0_y, c ='c', label='S1 : {} elements'.format(len(S1_
x)))
red = plt.scatter(S1_x, S1_y, c='b', marker = "^", label='S0 : {} elements'.fo
rmat(len(S0_x)))
line = ax.plot(X, Y, c = 'black', label='Saparator')
plt.title('Visualization')
plt.legend(loc="lower right")
plt.ylim([-1,1])
plt.xlim([-1,1])
plt.show()
```



This is the Graph showing the data points and saparator as per the weights.

In [83]:
```
def activation_fn(x):
    if x >= 0:
        y = 1
    else:
        y = 0
    return y
```

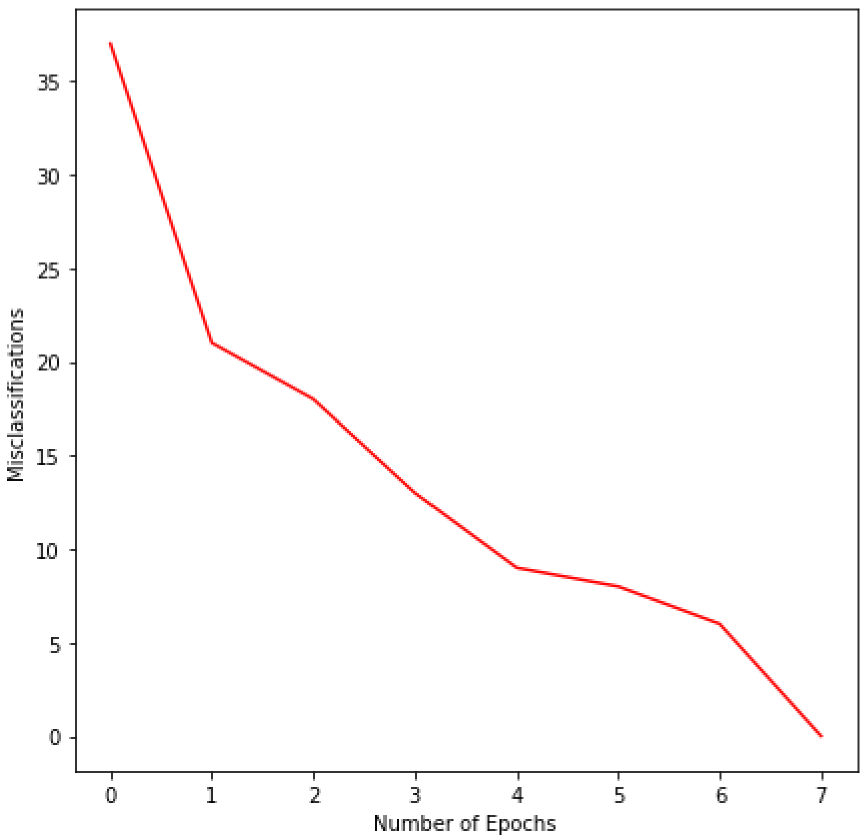
In [84]:
```
w0_1 = np.random.uniform(-1, 1)
w1_1 = np.random.uniform(-1, 1)
w2_1 = np.random.uniform(-1, 1)

omega = []
omega = [w0_1, w1_1, w2_1]

def misclassified(dataset, omega):
    misclassifications = 0
    for each in dataset:
        y = (omega[0]+(each[0]*omega[1])+(each[1]*omega[2]))
        y = activation_fn(y)
        if y != each[2]:
            misclassifications += 1
    return misclassifications
a = misclassified(dataset, omega)
print ('Misclassifications: ', a)
```

```
Misclassifications:  37
```

In [85]:
```
def perceptron_training(omega):
    epoch = 0
    omegas = []
    missed = []
    while (misclassified(dataset,omega)!=0):
        missed.append(misclassified(dataset,omega))
        #print ('Number of missclassifications: ', missed[epoch])
        epoch = epoch + 1
        #print ('Epoch Number: ', epoch)
        for each in range(len(dataset)):
            y = omega[0] + (dataset[each][0]*omega[1]) + (dataset[each][1]*ome
ga[2])
            y = activation_fn(y)
            updated_input =[1]+dataset[each][0:2]
            desired_output = dataset[each][2]
            difference = desired_output-y
            if difference != 0:
                updated_input[0]= updated_input[0]*learning_rate*difference
                updated_input[1]= updated_input[1]*learning_rate*difference
                updated_input[2]= updated_input[2]*learning_rate*difference
                omega[0] = omega[0]+updated_input[0]
                omega[1] = omega[1]+updated_input[1]
                omega[2] = omega[2]+updated_input[2]
        #print ('Updated weights: ', omega)
        omegas.append(omega)
    final_misclassification = misclassified(dataset,omega)
    #print ('Number of missclassifications: ', final_misclassification)
    print ('Optimal weights: ', omegas[-1])
    return omegas, missed
```

In [86]:
```
omega = [w0_1, w1_1, w2_1]
learning_rate = 1
print ('Initial weights: ' , omega)
omegas=[]
omegas, missed = perceptron_training(omega)
n_epochs = range(len(omegas)+1)
fig, ax = plt.subplots(figsize=(7,7))
ax.plot(n_epochs, missed+[0], c = 'red')
plt.ylabel('Misclassifications')
plt.xlabel('Number of Epochs')
plt.show()
```

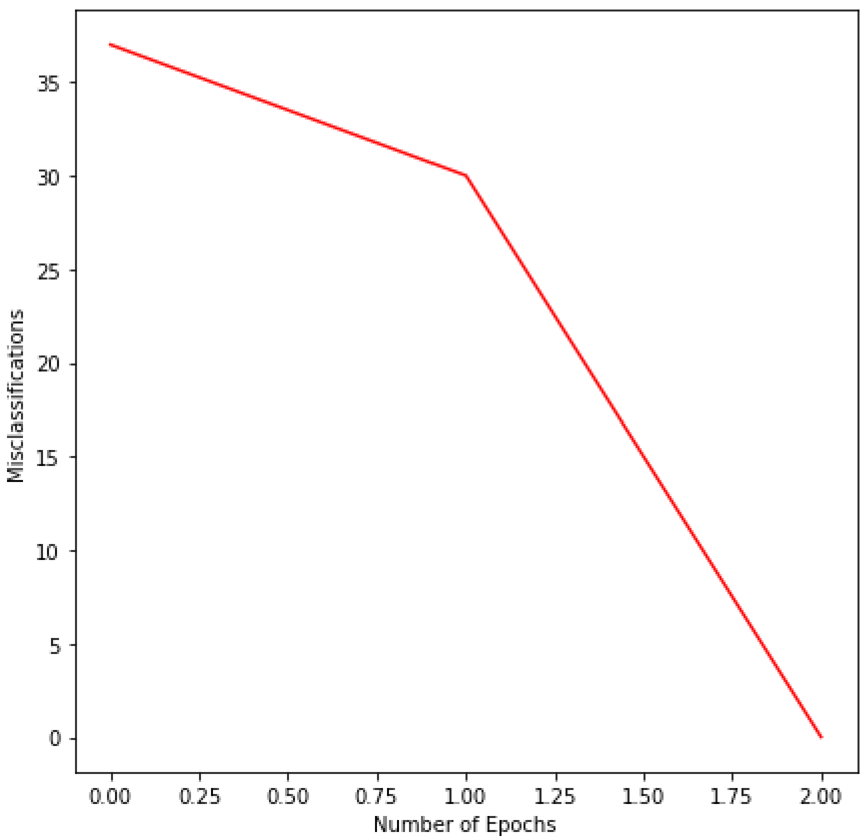
```
Initial weights:  [0.5287608918539548, 0.519143087090677, 0.5442670517294279]
Optimal weights:  [0.5287608918539548, -2.2146632503464887, 3.148461171571202
8]
```



This is the Graph showing the relation between misclassifications and number of epochs for learning rate of 1.

In [87]:
```
omega = [w0_1, w1_1, w2_1]
learning_rate = 10
print ('Initial weights: ' , omega)
omegas=[]
omegas, missed = perceptron_training(omega)
n_epochs = range(len(omegas)+1)
fig, ax = plt.subplots(figsize=(7,7))
ax.plot(n_epochs, missed+[0], c = 'red')
plt.ylabel('Misclassifications')
plt.xlabel('Number of Epochs')
plt.show()
```

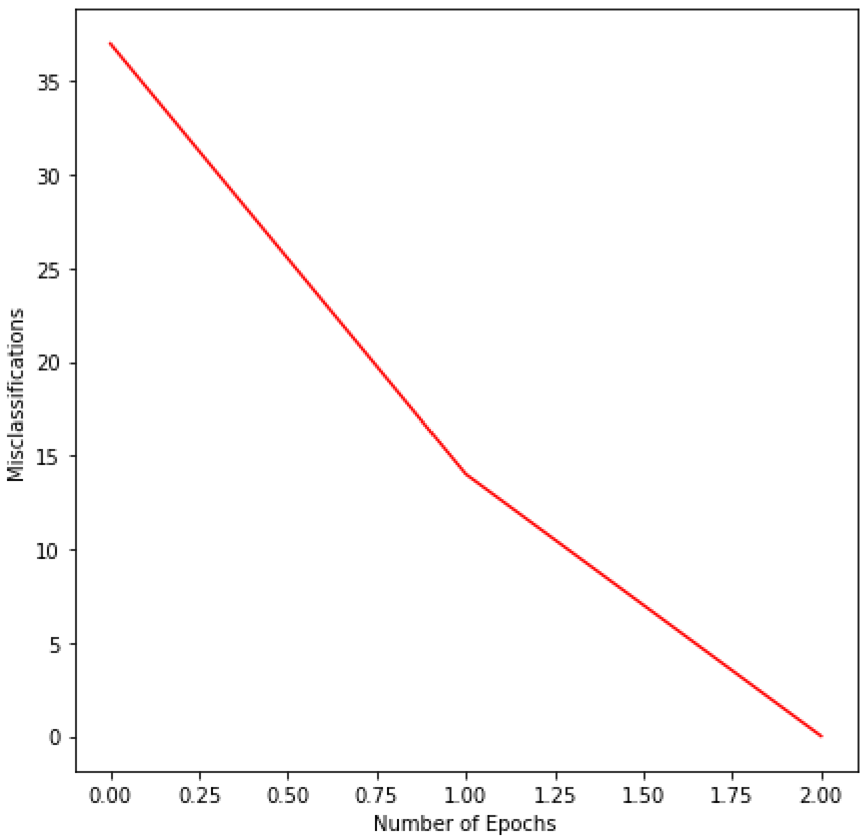
```
Initial weights:  [0.5287608918539548, 0.519143087090677, 0.5442670517294279]
Optimal weights:  [0.5287608918539561, -10.591139959050054, 16.32510293064790
5]
```



This is the Graph showing the relation between misclassifications and number of epochs for learning rate of 10.

In [88]:
```
omega = [w0_1, w1_1, w2_1]
learning_rate = 0.1
print ('Initial weights: ' , omega)
omegas=[]
omegas, missed = perceptron_training(omega)
n_epochs = range(len(omegas)+1)
fig, ax = plt.subplots(figsize=(7,7))
ax.plot(n_epochs, missed+[0], c = 'red')
plt.ylabel('Misclassifications')
plt.xlabel('Number of Epochs')
plt.show()
```

```
Initial weights:  [0.5287608918539548, 0.519143087090677, 0.5442670517294279]
Optimal weights:  [0.028760891853954834, -0.3147491688538412, 0.5413782668195
63]
```



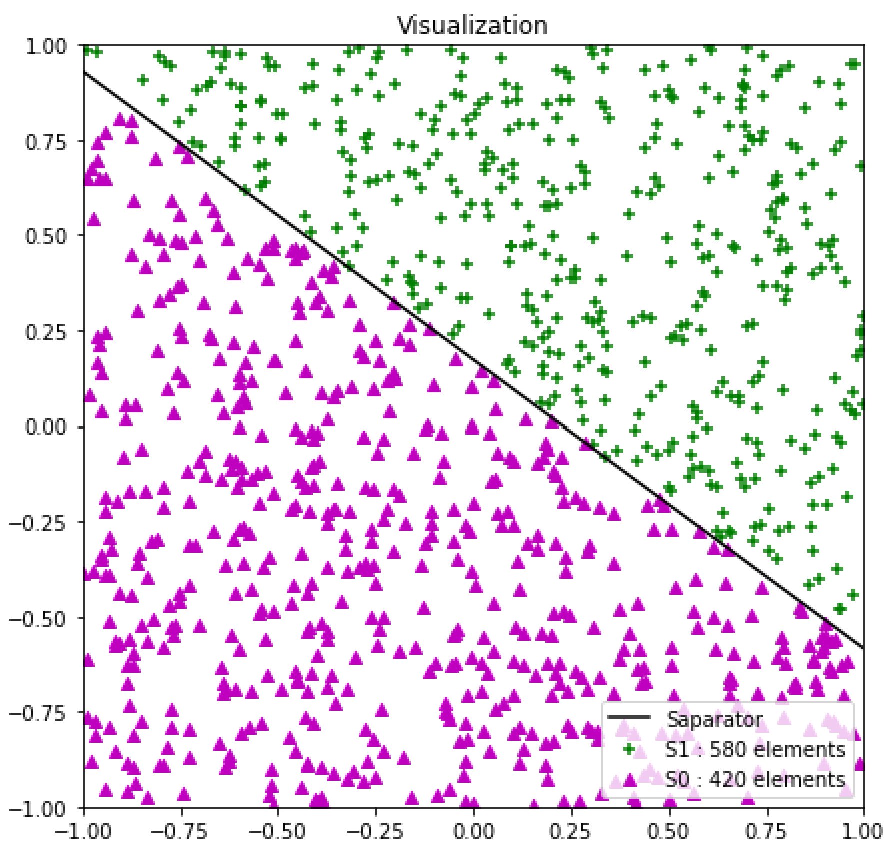
This is the Graph showing the relation between misclassifications and number of epochs for learning rate of 0.1.

In [89]:
```
w0 = np.random.uniform(-1/4, 1/4)
w1 = np.random.uniform(-1, 1)
w2 = np.random.uniform(-1, 1)
original_omega = [w0, w1, w2]
print('The original weights: ', original_omega)
```

```
The original weights:  [0.09996263703480268, -0.4385393324167328, -0.57920968
15746655]
```

In [90]:
```
S = 2 * np.random.rand(1000,2) - 1
S0 = []
S1 = []
for i in S:
    if (1*w0)+(i[0]*w1)+(i[1]*w2) >= 0:
            S1.append([i[0]] + [i[1]] + [0])
    elif (i[0]*w1)+(i[1]*w2) < 0:
            S0.append([i[0]] + [i[1]] + [1])
dataset = S0 + S1
```

In [91]:
```
x1 = -(w0-w2)/w1
x2 = -(w0+w2)/w1
X = np.array([x1, x2])
Y = np.array([-1.0, +1.0])

S1_x = []
S1_y = []
S0_x = []
S0_y = []

for i in S0:
    S0_x.append(i[0])
    S0_y.append(i[1])
for i in S1:
    S1_x.append(i[0])
    S1_y.append(i[1])

fig, ax = plt.subplots(figsize=(7,7))
blue = plt.scatter(S0_x, S0_y, c ='g',marker="+", label='S1 : {} elements'.for
mat(len(S1_x)))
red = plt.scatter(S1_x, S1_y, c='m', marker = "^", label='S0 : {} elements'.fo
rmat(len(S0_x)))
line = ax.plot(X, Y, c = 'black', label='Saparator')
plt.title('Visualization')
plt.legend(loc="lower right")
plt.ylim([-1,1])
plt.xlim([-1,1])
plt.show()
```



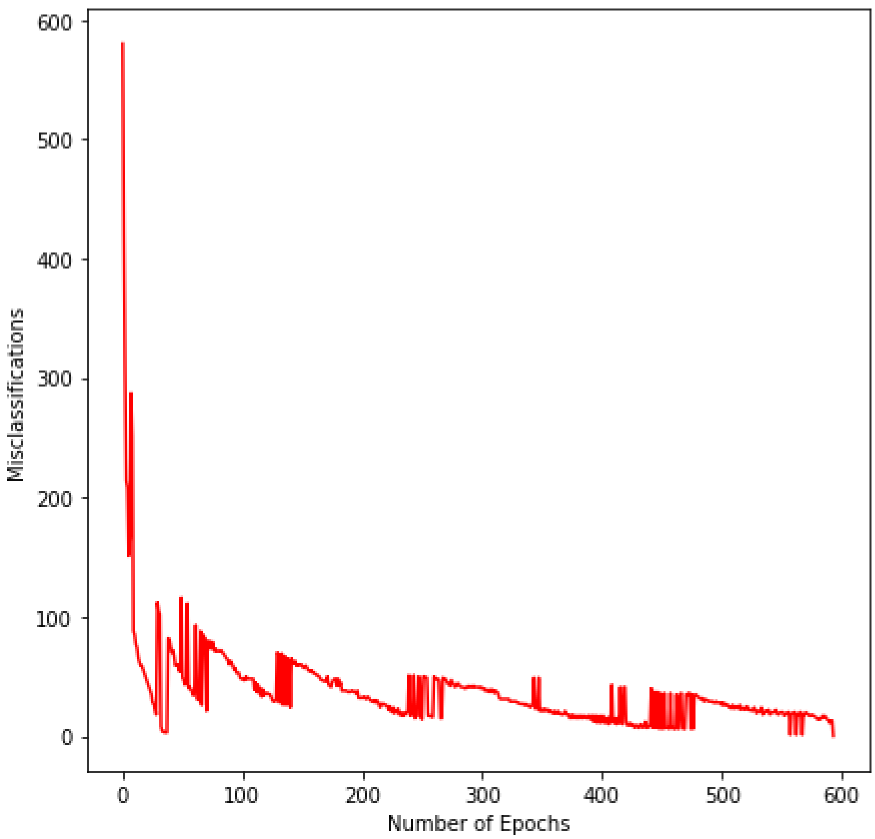
This is the Graph showing the dataset and saparator.

In [92]:
```
w0_1 = np.random.uniform(-1, 1)
w1_1 = np.random.uniform(-1, 1)
w2_1 = np.random.uniform(-1, 1)

omega = []
omega = [w0_1, w1_1, w2_1]

def misclassified(dataset, omega):
    misclassifications = 0
    for each in dataset:
        y = (omega[0]+(each[0]*omega[1])+(each[1]*omega[2]))
        y = activation_fn(y)
        if y != each[2]:
            misclassifications += 1
    return misclassifications
a = misclassified(dataset, omega)
print ('Misclassifications: ', a)
```

```
Misclassifications:  580
```

In [93]:
```
omega = [w0_1, w1_1, w2_1]
learning_rate = 1
print ('Initial weights: ' , omega)
omegas=[]
omegas, missed = perceptron_training(omega)
n_epochs = range(len(omegas)+1)
fig, ax = plt.subplots(figsize=(7,7))
ax.plot(n_epochs, missed+[0], c = 'red')
plt.ylabel('Misclassifications')
plt.xlabel('Number of Epochs')
plt.show()
```

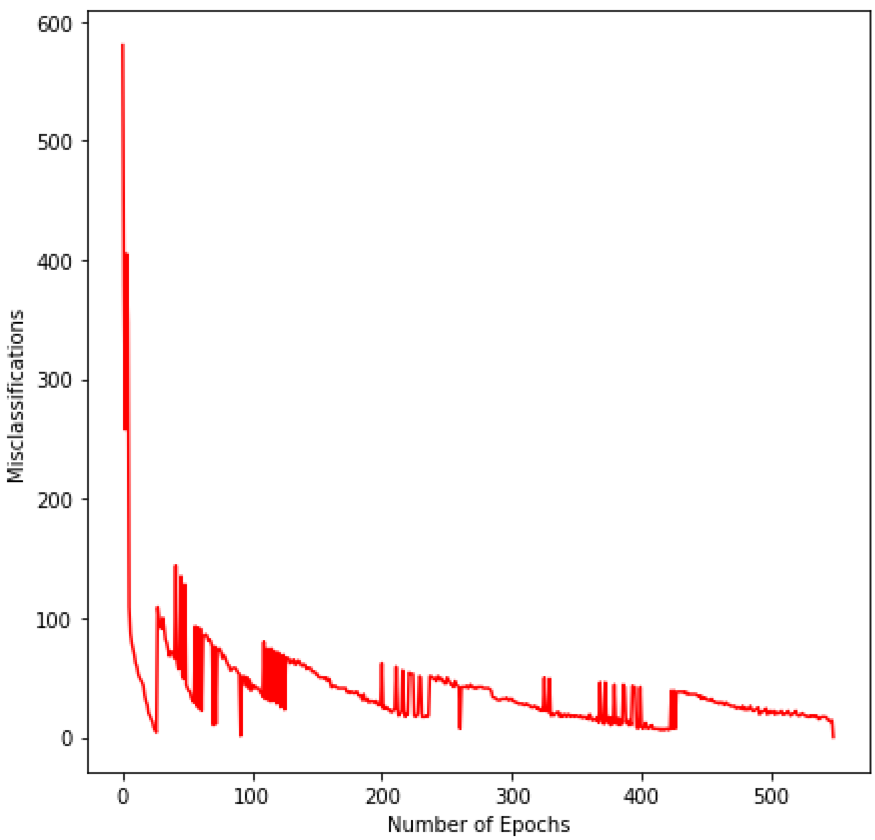
```
Initial weights:  [0.8045976450716181, -0.16702616467902964, 0.10978624846401
885]
Optimal weights:  [-5.195402354928381, 22.59918415799668, 30.042272790184988]
```



This is the Graph showing the relation between misclassifications and number of epochs for learning rate of 1.

In [94]:
```
omega = [w0_1, w1_1, w2_1]
learning_rate = 10
print ('Initial weights: ' , omega)
omegas=[]
omegas, missed = perceptron_training(omega)
n_epochs = range(len(omegas)+1)
fig, ax = plt.subplots(figsize=(7,7))
ax.plot(n_epochs, missed+[0], c = 'red')
plt.ylabel('Misclassifications')
plt.xlabel('Number of Epochs')
plt.show()
```

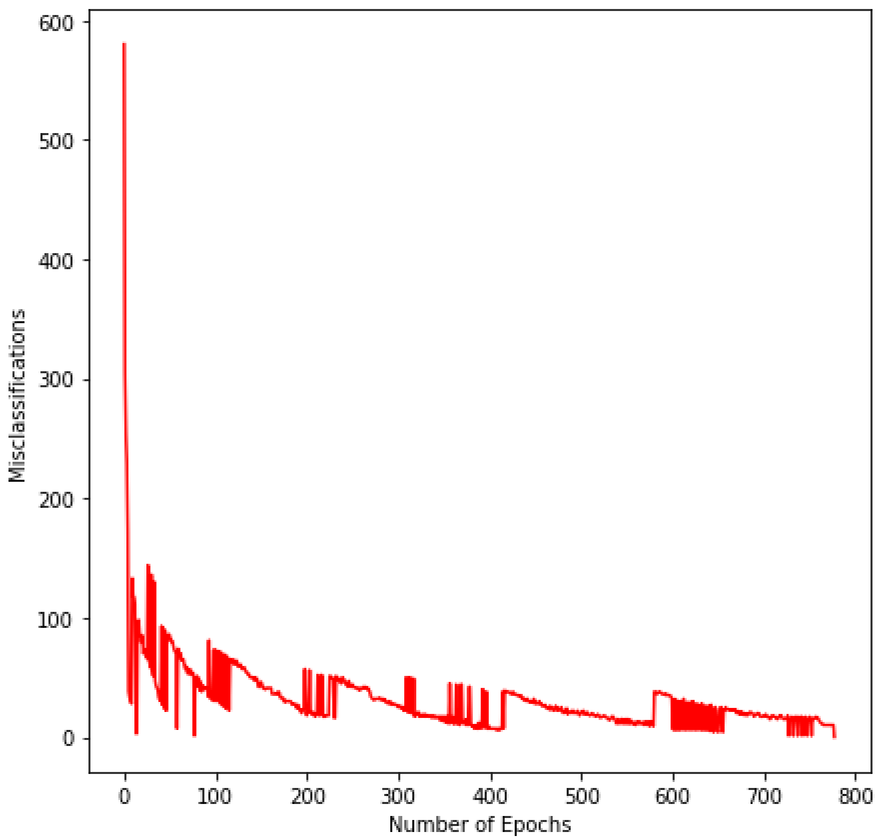
```
Initial weights:  [0.8045976450716181, -0.16702616467902964, 0.10978624846401
885]
Optimal weights:  [-49.19540235492838, 214.61596172077478, 286.1124863004799
3]
```



This is the Graph showing the relation between misclassifications and number of epochs for learning rate of 10.

In [95]:
```
omega = [w0_1, w1_1, w2_1]
learning_rate = 0.1
print ('Initial weights: ' , omega)
omegas=[]
omegas, missed = perceptron_training(omega)
n_epochs = range(len(omegas)+1)
fig, ax = plt.subplots(figsize=(7,7))
ax.plot(n_epochs, missed+[0], c = 'red')
plt.ylabel('Misclassifications')
plt.xlabel('Number of Epochs')
plt.show()
```

```
Initial weights:  [0.8045976450716181, -0.16702616467902964, 0.10978624846401
885]
Optimal weights:  [-0.5954023549283818, 2.5874350101988552, 3.438019189040667
7]
```



This is the Graph showing the relation between misclassifications and number of epochs for learning rate of 0.1.

Learning rate helps us find the covergence optimally. If the learning rate is increased to a high value then the algorithm might cross the optimum value and if the value of leaning rate is too small then it will take a lot of time for the algorithm to converge. Therefore, even though it is certain that the algorithm will finally converage, we must select an efficient value for the leaning rate.

The perceptron training algorithm must always converge for all positive learning rates. If the input classes are linearly separable, then the PTA will converge for any $\eta > 0$. This property would give us the exact same results of final weights everytime.

We can conclude that the number of epochs increases with the increase in the size of dataset.