In [409]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [410]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tru
e).
```

In [411]:
```
p_data = pd.read_csv("/content/drive/My Drive/cm121/p.tsv", sep='\t', header=None)
q_data = pd.read_csv("/content/drive/My Drive/cm121/q.tsv", sep='\t', header=None)
```

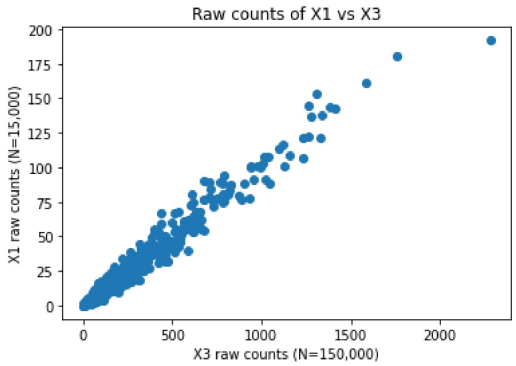
In [412]:
```
np.random.seed(0)
```

In [413]:
```
p = np.array(p_data[0])
q = np.array(q_data[0])
print(p.shape, q.shape)
```

```
(500,) (500,)
```

In [414]:
```
## part a
N = [15000, 30000, 150000]
N_samples = 3
X = []
for n in N:
  X.append(np.random.multinomial(n, p))

X = np.array(X).T
print("Size of X (d x 3):", X.shape) # d x 3
print("X:", X) # 500 rows, 3 columns
print("Sum of each column (should be N_j):", np.sum(X, axis=0)) # columns sum to Nj

X1 = X[:, 0]
X2 = X[:, 1]
X3 = X[:, 2]

plt.scatter(X3, X1)
plt.title("Raw counts of X1 vs X3")
plt.xlabel("X3 raw counts (N=150,000)")
plt.ylabel("X1 raw counts (N=15,000)")
plt.show()
```

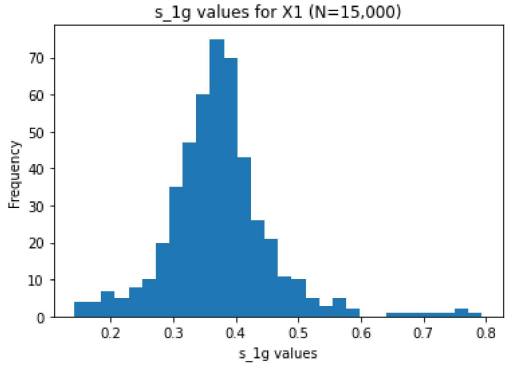
```
Size of X (d x 3): (500, 3)
X: [[ 56 143 638]
 [  6  11  60]
 [ 18  34 178]
 ...
 [ 37  83 356]
 [  6  18  74]
 [ 27  50 332]]
Sum of each column (should be N_j): [ 15000  30000 150000]
```

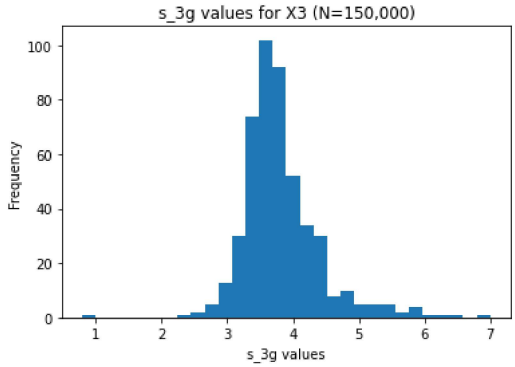


In [415]:
```
## part b
def DEseq_norm(X):
  # s_jg is scalar, s is matrix of same size as X
  # just have to divide each entry of X by prod(X_kg) ** (1/N) to get s
  # if X_jg is 0, then remove row from s_jg computation
  s = []
  N_sample = len(X[0])
  for g in range(len(X)):
    if 0 in X[g]:
      continue
    denom = np.prod(X[g], dtype=np.float64) ** (1/N_sample)
    s.append(X[g] / denom)
  s = np.array(s)
  sample_norm = np.median(s, axis=0)
  return s, sample_norm

s, s_j = DEseq_norm(X)
print("S:", s, s.shape) # filtered rows with 0, so smaller than X
print("Sample specific norm factor:", s_j)
```

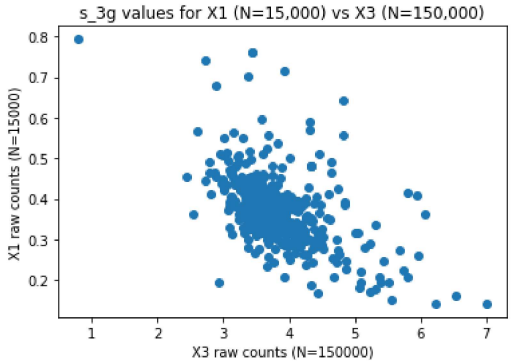
```
S: [[0.32514203 0.8302734  3.7042967 ]
 [0.3792447  0.69528195 3.79244701]
 [0.37689065 0.71190456 3.72702976]
 ...
 [0.35916316 0.80569034 3.45573204]
 [0.30010007 0.9003002  3.70123416]
 [0.35280913 0.65335024 4.33824558]] (479, 3)
Sample specific norm factor: [0.37040112 0.73533254 3.71168547]
```

In [416]:
```
## part c
s_1g = s[:, 0]
s_3g = s[:, 2]

plt.hist(s_1g, bins=30)
plt.title("s_1g values for X1 (N=15,000)")
plt.xlabel("s_1g values")
plt.ylabel("Frequency")
plt.show()

plt.hist(s_3g, bins=30)
plt.title("s_3g values for X3 (N=150,000)")
plt.xlabel("s_3g values")
plt.ylabel("Frequency")
plt.show()

print("Sample specific norm factor s_j:", s_j)
print("s_1:", s_j[0])
print("s_3:", s_j[2])

plt.scatter(s_3g, s_1g)
plt.title("s_3g values for X1 (N=15,000) vs X3 (N=150,000)")
plt.xlabel("X3 raw counts (N=150000)")
plt.ylabel("X1 raw counts (N=15000)")
plt.show()
```





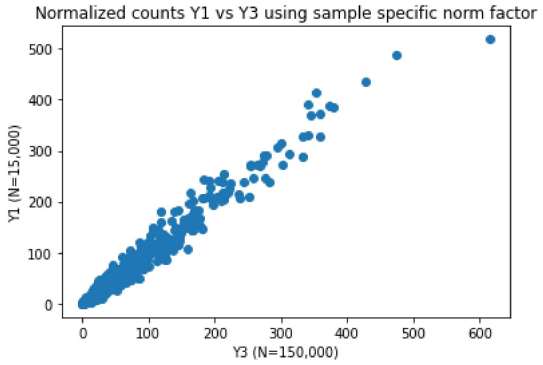

```
Sample specific norm factor s_j: [0.37040112 0.73533254 3.71168547]
s_1: 0.3704011230042894
s_3: 3.71168546524298
```



The sample specific norm factor for X1 is 0.3704, while for X3 it is 3.712, and the histograms show this. The median point (s_1g, s_3g) resides right in the bulk of the points on the scatterplot.

The histogram for s_1g shows that the count of gene g in X1, when normalized with the counts of gene g in the other samples, is only about 0.3704, with some genes being more frequent compared to other samples (we see some s_1g up to 0.8). Similarly, the count of gene g in X3, when normalized with the counts of gene g in the other samples, is higher at 3.712. We see some genes less frequently compared to the other samples (some s_3g are at less than 1), while some genes are more frequent compared to the other samples (with some normalized counts as high as 6 or 7 comapred to the median amount of about 3.7).

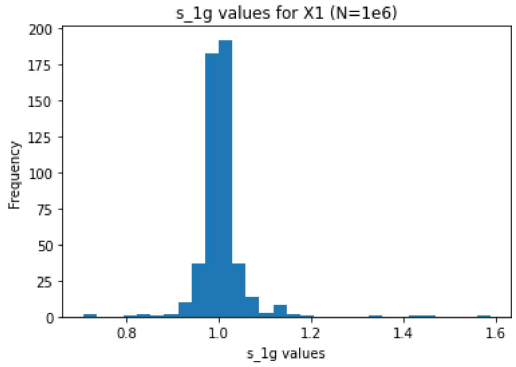
In [417]:
```
## part d
Y = X / s_j
plt.scatter(Y[:, 2], Y[:, 0]) # Y1 vs Y3
plt.title("Normalized counts Y1 vs Y3 using sample specific norm factor")
plt.xlabel("Y3 (N=150,000)")
plt.ylabel("Y1 (N=15,000)")
plt.show()
```

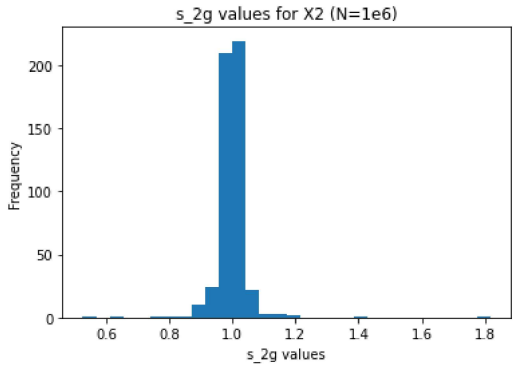


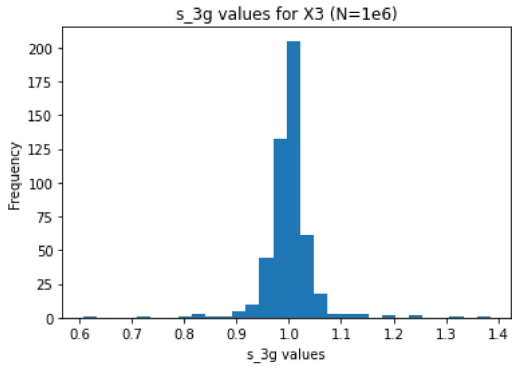
Comparing to the scatterplot X1 vs X3 in part a, the points themselves look to be the exact same, just scaled differently to the same normalized count, despite N1 = 15000 and N3 = 150000. It is now a much easier comparison to make now, given that the scales are the same for both samples. After normalization, all the points fall in the [0, 500] range (with one outlier), comapred to the plot in part a where the points are on different scales, from 15000 to 150000.

In [418]:
```
## part e
N_j = [1e6] * 3
X = []
for n in N_j:
  X.append(np.random.multinomial(n, p))

X = np.array(X).T
print("Size of X (d x 3):", X.shape) # d x 3
print("X:", X) # 500 rows, 3 columns
print("Sum of each column (should be N_j):", np.sum(X, axis=0)) # columns sum to Nj

X1 = X[:, 0]
X2 = X[:, 1]
X3 = X[:, 2]

s, s_j = DEseq_norm(X)
print("S:", s, s.shape)
print("Sample specific norm factor:", s_j)

s_1g = s[:, 0]
s_2g = s[:, 1]
s_3g = s[:, 2]

plt.hist(s_1g, bins=30)
plt.title("s_1g values for X1 (N=1e6)")
plt.xlabel("s_1g values")
plt.ylabel("Frequency")
plt.show()

plt.hist(s_2g, bins=30)
plt.title("s_2g values for X2 (N=1e6)")
plt.xlabel("s_2g values")
plt.ylabel("Frequency")
plt.show()

plt.hist(s_3g, bins=30)
plt.title("s_3g values for X3 (N=1e6)")
plt.xlabel("s_3g values")
plt.ylabel("Frequency")
plt.show()
```

```
Size of X (d x 3): (500, 3)
X: [[4155 4285 4222]
 [ 400  413  382]
 [1146 1127 1093]
 ...
 [2510 2578 2556]
 [ 531  584  526]
 [2023 1964 2019]]
Sum of each column (should be N_j): [1000000 1000000 1000000]
S: [[0.98451953 1.01532278 1.00039505]
 [1.00469797 1.03735065 0.95948656]
 [1.02158628 1.00464899 0.97434014]
 ...
 [0.98514743 1.01183668 1.00320192]
 [0.97184458 1.06884602 0.9626935 ]
 [1.01058145 0.98110824 1.00858326]] (499, 3)
Sample specific norm factor: [1.00025319 0.99971652 1.00080228]
```





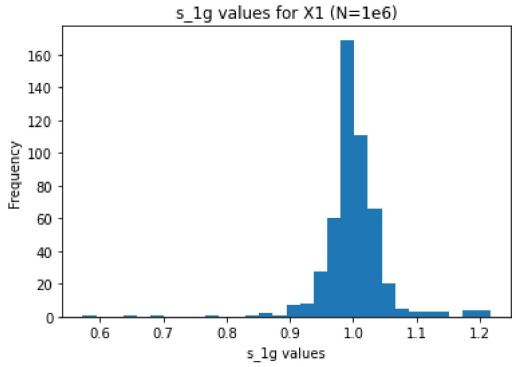


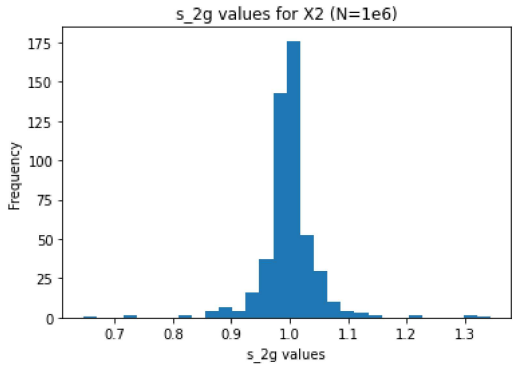
Since the sample sizes are the same, the sample specific norm factors (s_j) are also very similar at around 1 (not exactly similar since the samples are random). This is expected, because when we pull from the same abundance distribution, the count of gene g would be similar among samples.

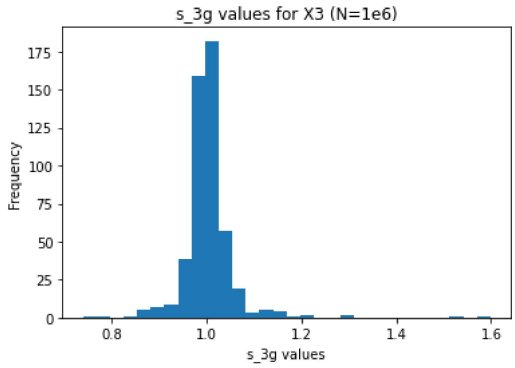
Compared to the s_j with N=15000, 3000, 150000 from part c where the s_j were different because of the different sample sizes, the norm factors are the same for each sample; this result is intuitive.

In [419]:
```
## part f
N_j = [1e6] * 3
X_q = []
for n in N_j:
  X_q.append(np.random.multinomial(n, q))

X_q = np.array(X_q).T
print("Size of X (d x 3):", X_q.shape) # d x 3
print("X:", X_q) # 500 rows, 3 columns
print("Sum of each column (should be N_j):", np.sum(X_q, axis=0)) # columns sum to Nj

X_q1 = X_q[:, 0]
X_q2 = X_q[:, 1]
X_q3 = X_q[:, 2]

s, s_j = DEseq_norm(X_q)
print("S:", s, s.shape)
print("Sample specific norm factor:", s_j)

s_1g = s[:, 0]
s_2g = s[:, 1]
s_3g = s[:, 2]

plt.hist(s_1g, bins=30)
plt.title("s_1g values for X1 (N=1e6)")
plt.xlabel("s_1g values")
plt.ylabel("Frequency")
plt.show()

plt.hist(s_2g, bins=30)
plt.title("s_2g values for X2 (N=1e6)")
plt.xlabel("s_2g values")
plt.ylabel("Frequency")
plt.show()

plt.hist(s_3g, bins=30)
plt.title("s_3g values for X3 (N=1e6)")
plt.xlabel("s_3g values")
plt.ylabel("Frequency")
plt.show()
```

```
Size of X (d x 3): (500, 3)
X: [[41230 41128 41114]
 [ 3826  3702  3852]
 [11393 11430 11427]
 ...
 [ 1546  1559  1529]
 [  330   314   312]
 [ 1208  1273  1276]]
Sum of each column (should be N_j): [1000000 1000000 1000000]
S: [[1.00176637 0.99928808 0.99894792]
 [1.00876285 0.97606902 1.01561801]
 [0.99792809 1.00116897 1.00090619]
 ...
 [1.00089486 1.00931118 0.9898889 ]
 [1.03589216 0.98566708 0.97938895]
 [0.96490583 1.01682543 1.01922172]] (499, 3)
Sample specific norm factor: [0.99887273 0.99916662 0.99988166]
```







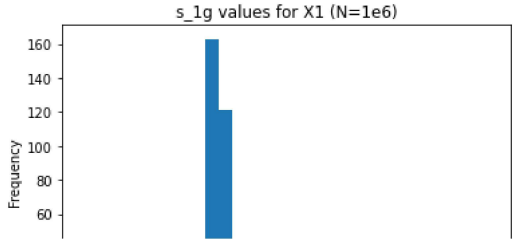
Between samples, the the sample specific norm factors (s_j) are very similar at around 1 (not exactly similar since the samples are random), since the sample sizes are the same.

Comapred to the results from part e, these results are similar since although we use a different distribution q, the sample sizes are the same, and so we would expect, when we pull from the same abundance distribution, that the count of gene g would be similar among samples. It does not matter which abundance distribution we sample from, because for both part e and f, we only compare samples from the same distribution.

Compared to the s_j with N=15000, 3000, 150000 from part c where the s_j were different because of the different sample sizes, the norm factors are the same for each sample; again, this result is intuitive.

In [420]:
```
## part g
X_g = np.concatenate((X, X_q), axis=1)

s, s_j = DEseq_norm(X_g)
print("S:", s, s.shape)
print("Sample specific norm factor:", s_j)

for i in range(len(X_g[0])):
  plt.hist(s[:, i], bins=30)
  plt.title("s_" + str(i+1) + "g values for X" + str(i+1) + " (N=1e6)")
  plt.xlabel("s_" + str(i+1) + "g values")
  plt.ylabel("Frequency")
  plt.show()
```

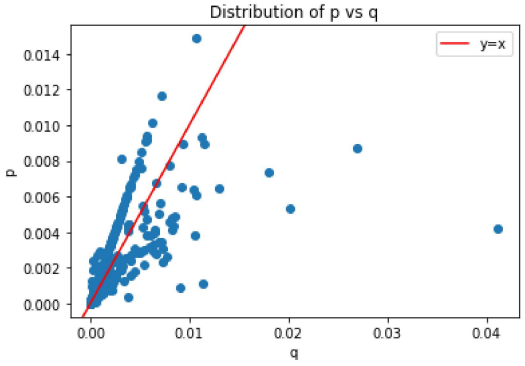
```
S: [[0.3152639  0.32512775 0.32034757 3.12835871 3.12061938 3.11955712]
 [0.32551411 0.33609331 0.31086597 3.11354243 3.01263306 3.13470085]
 [0.32022882 0.31491961 0.30541893 3.18356623 3.1939052  3.19306691]
 ...
 [1.26525179 1.29952952 1.28843967 0.77931445 0.78586754 0.77074501]
 [1.2727584  1.39979455 1.26077386 0.7909798  0.75262926 0.74783544]
 [1.27788743 1.24061835 1.27536072 0.76306872 0.80412788 0.80602292]] (499, 6)
Sample specific norm factor: [1.25639055 1.2542608  1.25670336 0.8011548  0.79957458 0.8014164 ]
```



Here, we see that the sample specific norm factor for the samples from p (X1, X2, X3) are around 1.255 (s_j), while for the samples from q (X4, X5, X6), they are around 0.8. Since the sample size is still 1e6 for all 6 samples, the results are similar to part e and f in that all s_j from the same true relative abundance of d genes (either p or q) are the same.

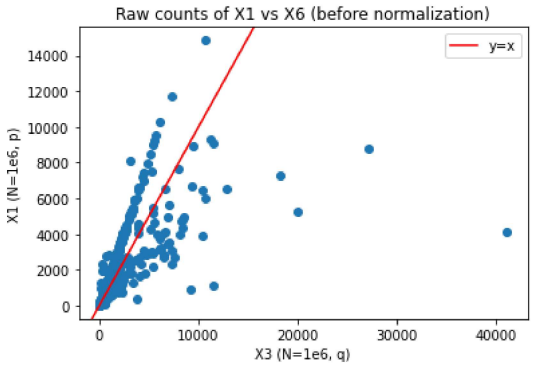
The difference is that due to the different distributions of p and q, the s_j differs between samples from p and q despite the sample sizes being the same. We can understand the difference by plotting p vs q.

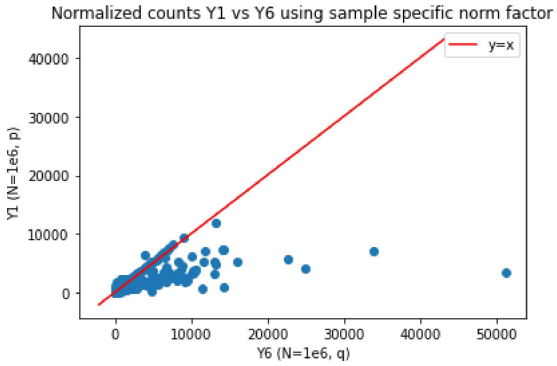
In [421]:
```
plt.scatter(q, p)
plt.title("Distribution of p vs q")
plt.xlabel("q")
plt.ylabel("p")
xpoints = ypoints = plt.xlim()
plt.plot(xpoints, ypoints, color='red', scalex=False, scaley=False, label='y=x')
plt.legend()
plt.show()
```



As we can see from the scatter plot, for the most part, the same genes that are lowly expressed in q are more highly expressed in p (these are the points above the y=x line). This explains why the norm factors for the samples from p are higher than the ones from q - the median genes are more highly expressed in p, and so to normalize their counts, we have to divide them by a higher factor to get to the same scale.

In [422]:
```
Y = X_g / s_j

plt.scatter(X_g[:,5], X_g[:,0])
plt.title("Raw counts of X1 vs X6 (before normalization)")
plt.xlabel("X3 (N=1e6, q)")
plt.ylabel("X1 (N=1e6, p)")
xpoints = ypoints = plt.xlim()
plt.plot(xpoints, ypoints, color='red', scalex=False, scaley=False, label='y=x')
plt.legend()
plt.show()

plt.scatter(Y[:,5], Y[:,0])
plt.title("Normalized counts Y1 vs Y6 using sample specific norm factor")
plt.xlabel("Y6 (N=1e6, q)")
plt.ylabel("Y1 (N=1e6, p)")
plt.plot(xpoints, ypoints, color='red', scalex=False, scaley=False, label='y=x')
plt.legend()
plt.show()
```





As we can see, before normalization, the scatter plot basically matches the distribution of p vs q scatterplot - this is because X1 is from p and X6 is from q, and the sample sizes are the same.

After normalization, because we divide the gene counts from X1 by about 1.25 and the counts from X2 by about 0.8, the normalized gene counts for X1 (from p) decrease, while the ones for X6 (from q) increase, and so the graph is like an inverse of the original graph before normalization; additionally the larger points are scaled down more due to normalization, and so the points are more bunched up (less points are farther away).

In [422]: