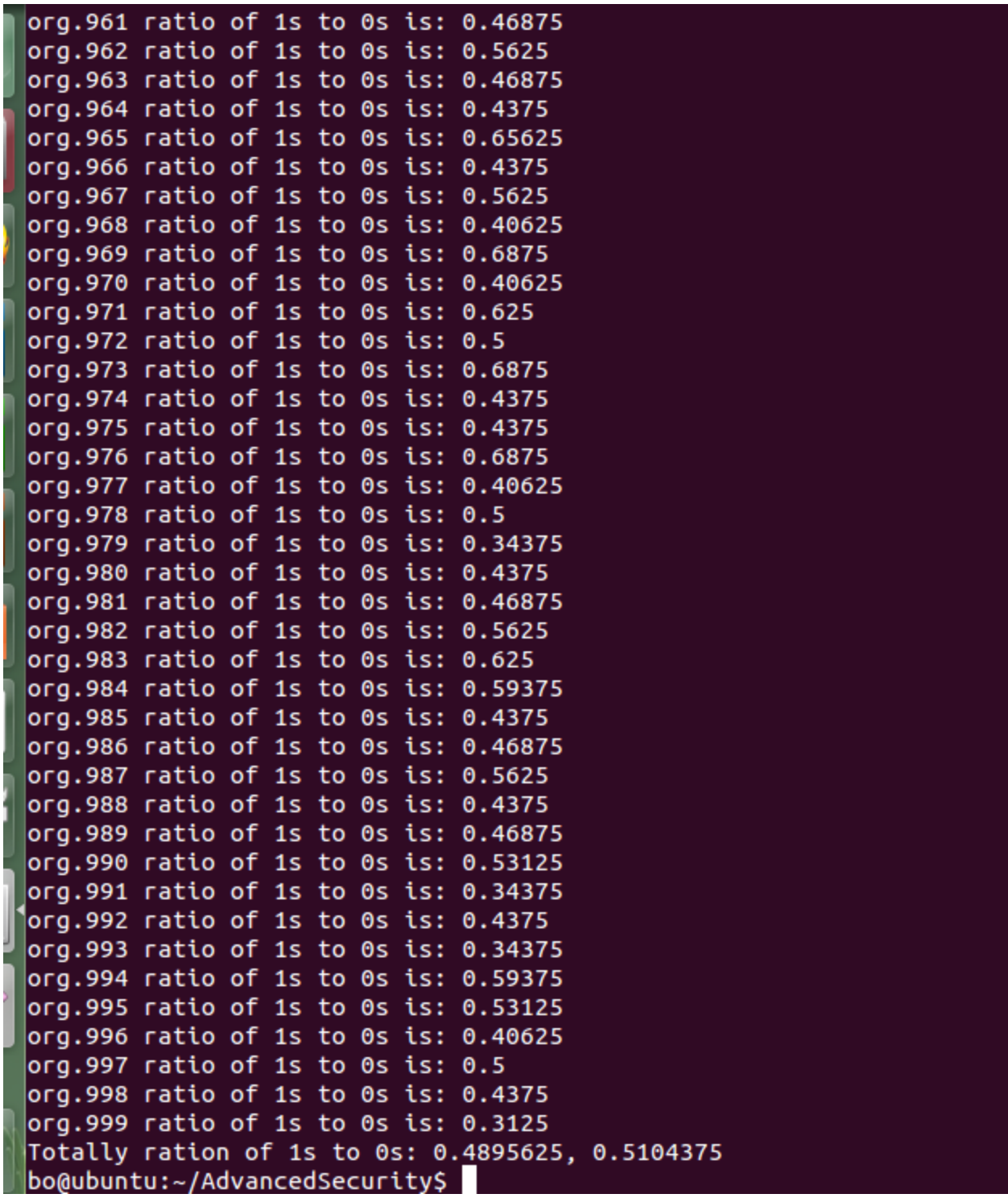
Analysis

1.

```
bo@ubuntu:~/AdvancedSecurity$ python analysis1.py
Collision in hashes.orig
06606003 ocours 2
06e0e0e0 ocours 2
eb6b605a ocours 2
dd6d606d ocours 2
060ae0f5 ocours 2
collision between .alt and .orig
ddeaec8a ocours 2
de0e2033 ocours 2
0620201e ocours 3
0620601e ocours 2
06761867 ocours 2
06761863 ocours 2
0660201e ocours 3
060ae0f5 ocours 2
06761874 ocours 2
d9e06041 ocours 2
06606008 ocours 2
06606003 ocours 2
0660601e ocours 2
eb6b6081 ocours 2
06b36067 ocours 2
eb6b605a ocours 2
060a200b ocours 2
d9e02041 ocours 2
d2b4b84a ocours 2
06e0e0e0 ocours 2
dd6d606d ocours 2
bo@ubuntu:~/AdvancedSecurity$
```

- From the result above, the number of collisions in .org file are 5. There are 10 messages map to 5 digests.
- Digest same results together in .mod, the number of collisions are 21, there are 44 messages map to 21 digests.
- Because I hard coded the “temp” as “1,2,3, /”, and pad the “new” whenever it needs with “x”. Even though I do use transportation, XOR and encryption (By adding temp value to new), some of the data can run to each other. If the array size bigger, will better.

2.

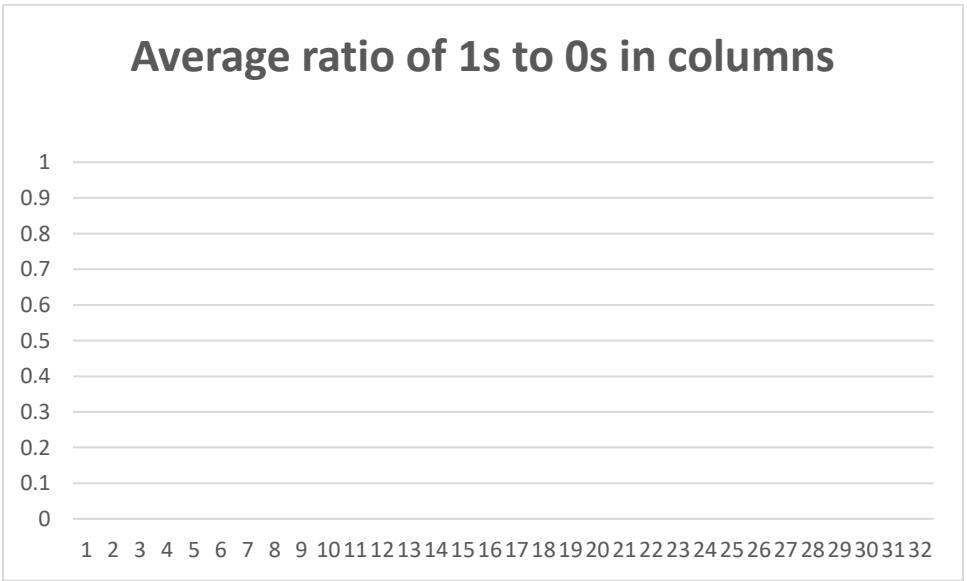


- The result is 0.4895625 for 1s.
- Which means there are more 0s and less 1s. Probably because the temp array I used for hard coded which is “1, 2, 3, /” and the padding “x”.
- The reason why my program 1s and 0s are kind of balancing, I use the digest from starts from the beginning of the message. That means every part of message can affect the final digest. Since message are random, so the 1s and 0s are around 50%.

3.

```
buntu: ~/AdvancedSecurity
bo@ubuntu:~/AdvancedSecurity$ python analysis3.py
Column: 0 of 1s ratio is 0.604
Column: 1 of 1s ratio is 0.643
Column: 2 of 1s ratio is 0.296
Column: 3 of 1s ratio is 0.532
Column: 4 of 1s ratio is 0.449
Column: 5 of 1s ratio is 0.593
Column: 6 of 1s ratio is 0.623
Column: 7 of 1s ratio is 0.424
Column: 8 of 1s ratio is 0.576
Column: 9 of 1s ratio is 0.403
Column: 10 of 1s ratio is 0.731
Column: 11 of 1s ratio is 0.402
Column: 12 of 1s ratio is 0.397
Column: 13 of 1s ratio is 0.419
Column: 14 of 1s ratio is 0.389
Column: 15 of 1s ratio is 0.46
Column: 16 of 1s ratio is 0.611
Column: 17 of 1s ratio is 0.37
Column: 18 of 1s ratio is 0.716
Column: 19 of 1s ratio is 0.406
Column: 20 of 1s ratio is 0.378
Column: 21 of 1s ratio is 0.435
Column: 22 of 1s ratio is 0.375
Column: 23 of 1s ratio is 0.424
Column: 24 of 1s ratio is 0.499
Column: 25 of 1s ratio is 0.481
Column: 26 of 1s ratio is 0.514
Column: 27 of 1s ratio is 0.501
Column: 28 of 1s ratio is 0.515
Column: 29 of 1s ratio is 0.491
Column: 30 of 1s ratio is 0.508
Column: 31 of 1s ratio is 0.501
bo@ubuntu:~/AdvancedSecurity$
```

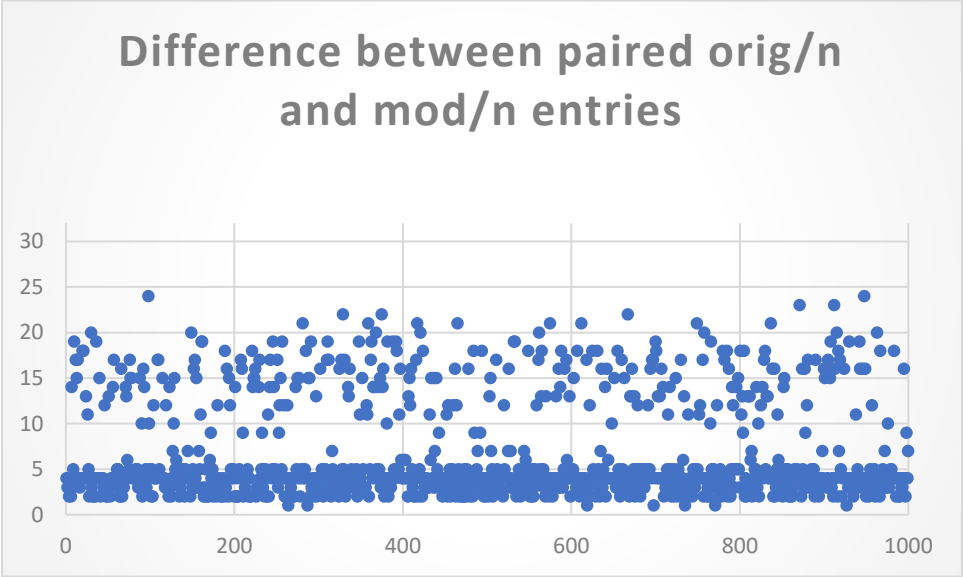
The plot is:



- This can make me tell there are more 0s than 1s.
- Which means there are more 0s and less 1s. Probably because the temp array I used for hard coded which is “1, 2, 3, /” and the padding “x”.

4.

```
bo@ubuntu:~/AdvancedSecurity$ emacs analysis4.py
bo@ubuntu:~/AdvancedSecurity$ python analysis4.py
The average differences is :6.0
bo@ubuntu:~/AdvancedSecurity$
```



- My result is 6 on average. So only 6 bits different compare with orig/ and mod/. Slightly different for mostly files.
- From the plot, there are some points around 2 differences and 24 differences. The reason probably because the only different in those two file is " ~ ", some permutation and substitution exaggerate this different but some are mitigate. This is my guess may not right. If I change the hard-coded array and pad character, this change can change.

5.

```
bo@ubuntu:~/AdvancedSecurity$ python analysis5.py
hash.org
Unique string in hash.org are: 995
There are 5 collision in hash.org
Shannon entropy in hash.org is: 9.95578428466
hash.alt:
Unique string in hash.alt are: 1977
There are 23 collision in hash.alt
Shannon entropy in hash.alt is: 10.9420293972
bo@ubuntu:~/AdvancedSecurity$
```

- My hash is 9.95578428466 (orig/) and 10.9420293972 (orig/ and mod/), so which is close to 9.966 and 10.966.
- From my hash function, the entropy changes significantly, for orig/ part, only 0.01 less than ideal, but for orig/ and mod/ part, the change is 0.02. which means less random.

Reflection

1. Yes, my hash function kind of collision resistant, because 1000 input only 5 collisions and 2000 input slightly different only 44 collisions.
2. Sort of, there are some collisions in my hash function, so H(x) = H(y) but x and y are different.

3. I will choose random character generator for array temp. Which can give me better randomness of hash value.
4. I think this project is interesting, and probably if I use python for hash function can make my life easier.
5. For the hash function part, if there is a good hash and not too hard, I probably can follow this logic and make better hash.