CSL7630: Algorithms for Big Data

Assignment 2: Sketching and Streaming

Saurabh Shashikant Burewar (B18CSE050)

Requirements

The algorithms are implemented in Python 3.8 and requires the following -

- Numpy
- tqdm

Dataset Generation

1. Building data

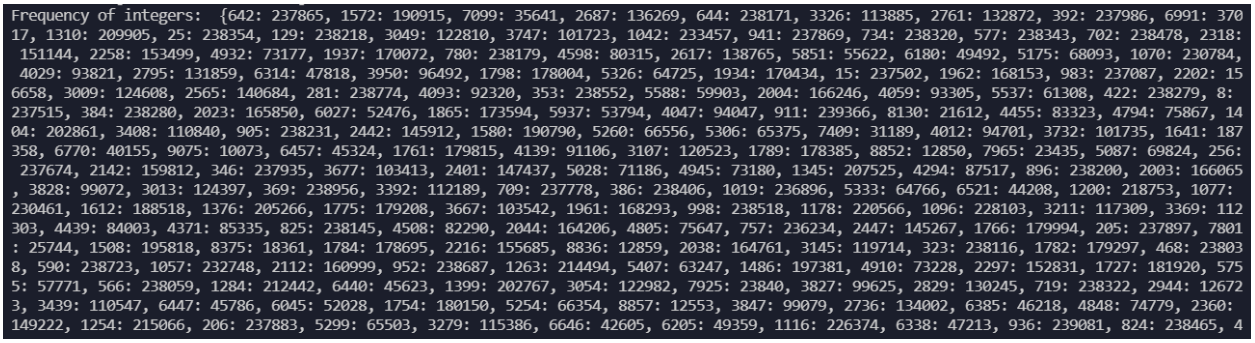
```
python data.py
```

The dataset is generated using the code snippet provided in the assignment. Running the code created a dataset of random integers of the size 4.95 GB.

2. Building statistics

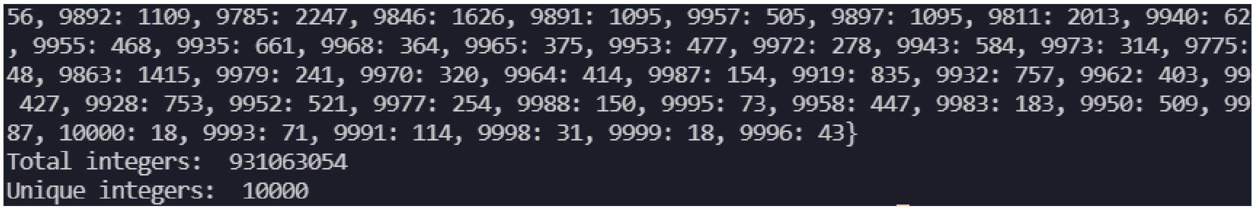
```
python stats.py
```

This script runs through the data and records all the required statistics which include number of integers, unique integers and frequency of each integer. The results are as follows -



```
Frequency of integers:  {642: 237865, 1572: 190915, 7099: 35641, 2687: 136269, 644: 238171, 3326: 113885, 2761: 132872, 392: 237986, 6991: 370
17, 1310: 209905, 25: 238354, 129: 238218, 3049: 122810, 3747: 101723, 1042: 233457, 941: 237869, 734: 238320, 577: 238343, 702: 238478, 2318:
 151144, 2258: 153499, 4932: 73177, 1937: 170072, 780: 238179, 4598: 80315, 2617: 138765, 5851: 55622, 6180: 49492, 5175: 68093, 1070: 230784,
 4029: 93821, 2795: 131859, 6314: 47818, 3950: 96492, 1798: 178004, 5326: 64725, 1934: 170434, 15: 237502, 1962: 168153, 983: 237087, 2202: 15
6658, 3009: 124608, 2565: 140684, 281: 238774, 4093: 92320, 353: 238552, 5588: 59903, 2004: 166246, 4059: 93305, 5537: 61308, 422: 238279, 8:
237515, 384: 238280, 2023: 165850, 6027: 52476, 1865: 173594, 5937: 53794, 4047: 94047, 911: 239366, 8130: 21612, 4455: 83323, 4794: 75867, 14
04: 202861, 3408: 110840, 905: 238231, 2442: 145912, 1580: 190790, 5260: 66556, 5306: 65375, 7409: 31189, 4012: 94701, 3732: 101735, 1641: 187
358, 6770: 40155, 9075: 10073, 6457: 45324, 1761: 179815, 4139: 91106, 3107: 120523, 1789: 178385, 8852: 12850, 7965: 23435, 5087: 69824, 256:
 237674, 2142: 159812, 346: 237935, 3677: 103413, 2401: 147437, 5028: 71186, 4945: 73180, 1345: 207525, 4294: 87517, 896: 238200, 2003: 166065
, 3828: 99072, 3013: 124397, 369: 238956, 3392: 112189, 709: 237778, 386: 238406, 1019: 236896, 5333: 64766, 6521: 44208, 1200: 218753, 1077:
230461, 1612: 188518, 1376: 205266, 1775: 179208, 3667: 103542, 1961: 168293, 998: 238518, 1178: 220566, 1096: 228103, 3211: 117309, 3369: 112
303, 4439: 84003, 4371: 85335, 825: 238145, 4508: 82290, 2044: 164206, 4805: 75647, 757: 236234, 2447: 145267, 1766: 179994, 205: 237897, 7801
: 25744, 1508: 195818, 8375: 18361, 1784: 178695, 2216: 155685, 8836: 12859, 2038: 164761, 3145: 119714, 323: 238116, 1782: 179297, 468: 23803
8, 590: 238723, 1057: 232748, 2112: 160999, 952: 238687, 1263: 214494, 5407: 63247, 1486: 197381, 4910: 73228, 2297: 152831, 1727: 181920, 575
5: 57771, 566: 238059, 1284: 212442, 6440: 45623, 1399: 202767, 3054: 122982, 7925: 23840, 3827: 99625, 2829: 130245, 719: 238322, 2944: 12672
3, 3439: 110547, 6447: 45786, 6045: 52028, 1754: 180150, 5254: 66354, 8857: 12553, 3847: 99079, 2736: 134002, 6385: 46218, 4848: 74779, 2360:
149222, 1254: 215066, 206: 237883, 5299: 65503, 3279: 115386, 6646: 42605, 6205: 49359, 1116: 226374, 6338: 47213, 936: 239081, 824: 238465, 4
```

….



```
56, 9892: 1109, 9785: 2247, 9846: 1626, 9891: 1095, 9957: 505, 9897: 1095, 9811: 2013, 9940: 62
, 9955: 468, 9935: 661, 9968: 364, 9965: 375, 9953: 477, 9972: 278, 9943: 584, 9973: 314, 9775:
48, 9863: 1415, 9979: 241, 9970: 320, 9964: 414, 9987: 154, 9919: 835, 9932: 757, 9962: 403, 99
 427, 9928: 753, 9952: 521, 9977: 254, 9988: 150, 9995: 73, 9958: 447, 9983: 183, 9950: 509, 99
87, 10000: 18, 9993: 71, 9991: 114, 9998: 31, 9999: 18, 9996: 43}
Total integers:  931063054
Unique integers:  10000
```

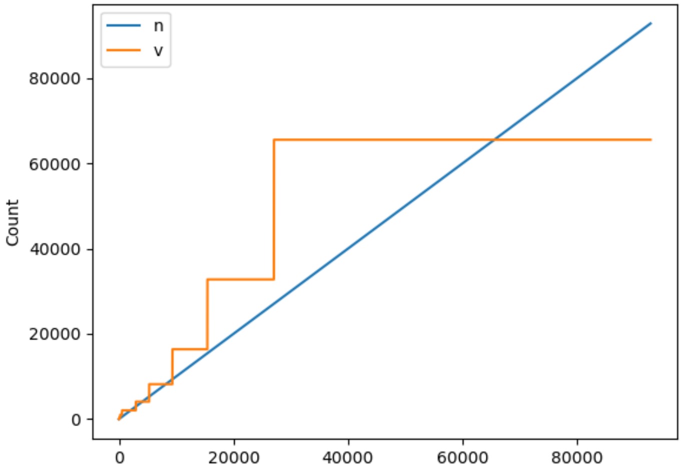
Sketching

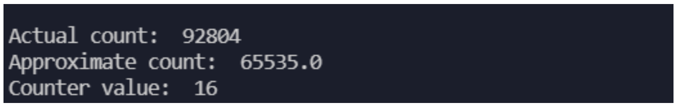
1. Approximate Count

```
python morris.py
```

We create the morris model with the increment and estimate methods. We also add the method to return the counter value. We use this morris model to create the morris+, which basically averages the results of n Morris models.

Running it for a short dataset of around 90,000 entries and plotting the counter states, we get the following result -

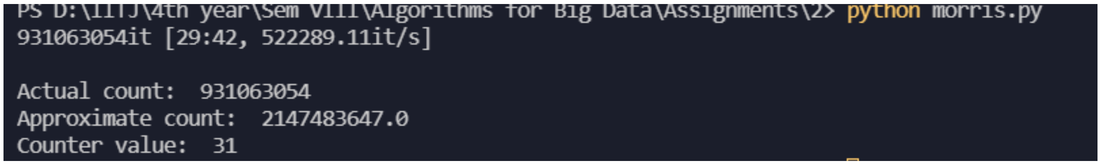




```
Actual count:  92804
Approximate count:  65535.0
Counter value:  16
```

Running it for full dataset for different n, we get -

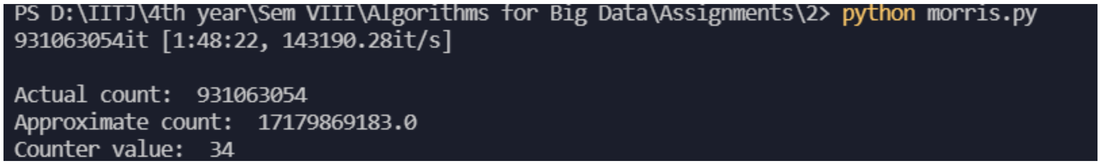
n = 2



```
PS D:\IITJ\4th year\Sem VIII\Algorithms for Big Data\Assignments\2> python morris.py
931063054it [29:42, 522289.11it/s]

Actual count:  931063054
Approximate count:  2147483647.0
Counter value:  31
```

n = 10



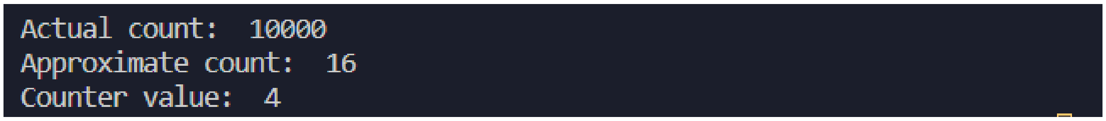
```
PS D:\IITJ\4th year\Sem VIII\Algorithms for Big Data\Assignments\2> python morris.py
931063054it [1:48:22, 143190.28it/s]

Actual count:  931063054
Approximate count:  17179869183.0
Counter value:  34
```

2. Approximate Distinct Count

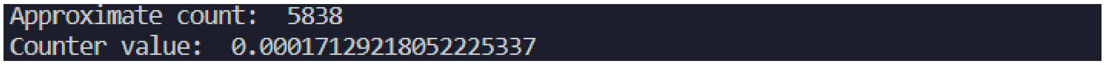
```
python distinct.py
```

The algorithm discussed for non-idealised count is implemented. The results are as follows -



```
Actual count:  10000
Approximate count:  16
Counter value:  4
```

Implementing a FM type algorithm on the other hand gives the following result -



```
Approximate count:  5838
Counter value:  0.00017129218052225337
```

3. Frequency Query

```
python count.py
python countmin.py
```

We get 15 random entries from the dataset which we will count the frequency of.

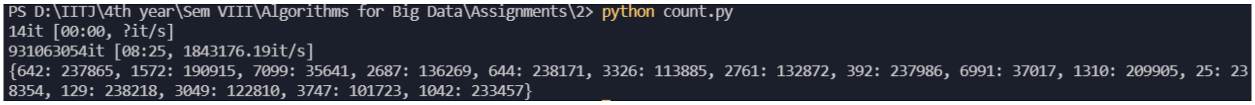
The count sketch is implemented with update and estimate methods. We also have a function to create 2-pairwise hash functions. We give the values of t and k and run the program. For each entry, it updates the sketch and at last it gives the estimate.
Here, we have taken t = 20 and k = 5, so the corresponding values of ε and δ will be,

$$\varepsilon = 2/5 = 0.4$$

$$\delta = 1/2^{20} = 0.0000009536$$

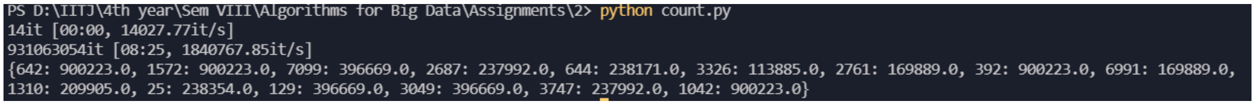
The count min sketch is also implemented in a similar way. Here too, we take the value of t and k and for each entry, it updates the sketch and at last it gives the estimate. The values of ε and δ are the same here.

The actual results using linear counting are as follows -



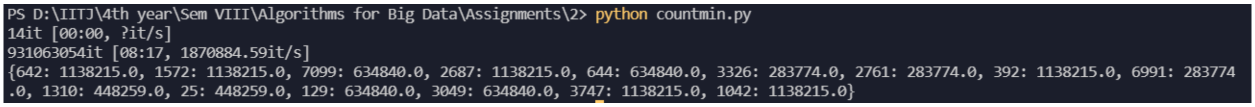
```
PS D:\IITJ\4th year\Sem VIII\Algorithms for Big Data\Assignments\2> python count.py
14it [00:00, ?it/s]
931063054it [08:25, 1843176.19it/s]
{642: 237865, 1572: 190915, 7099: 35641, 2687: 136269, 644: 238171, 3326: 113885, 2761: 132872, 392: 237986, 6991: 37017, 1310: 209905, 25: 23
8354, 129: 238218, 3049: 122810, 3747: 101723, 1042: 233457}
```

The results using count sketch are as follows -



```
PS D:\IITJ\4th year\Sem VIII\Algorithms for Big Data\Assignments\2> python count.py
14it [00:00, 14027.77it/s]
931063054it [08:25, 1840767.85it/s]
{642: 900223.0, 1572: 900223.0, 7099: 396669.0, 2687: 237992.0, 644: 238171.0, 3326: 113885.0, 2761: 169889.0, 392: 900223.0, 6991: 169889.0,
1310: 209905.0, 25: 238354.0, 129: 396669.0, 3049: 396669.0, 3747: 237992.0, 1042: 900223.0}
```

The results using count min sketch are as follows -



```
PS D:\IITJ\4th year\Sem VIII\Algorithms for Big Data\Assignments\2> python countmin.py
14it [00:00, ?it/s]
931063054it [08:17, 1870884.59it/s]
{642: 1138215.0, 1572: 1138215.0, 7099: 634840.0, 2687: 1138215.0, 644: 634840.0, 3326: 283774.0, 2761: 283774.0, 392: 1138215.0, 6991: 283774
.0, 1310: 448259.0, 25: 448259.0, 129: 634840.0, 3049: 634840.0, 3747: 1138215.0, 1042: 1138215.0}
```

Streaming

All the above scripts work on the data as a stream.

References

- https://courses.engr.illinois.edu/cs498abd/fa2020/slides/04-lec.pdf
- https://arpitbhayani.me/blogs/morris-counter
- Sketching Algorithms - Jelani Nelson
- https://www.analyticsvidhya.com/blog/2021/06/beginners-guide-to-flajolet-martin-algorithm/
- https://stackoverflow.com/questions/16284317/obtaining-a-k-wise-independent-hash-function
- https://courses.cs.washington.edu/courses/cse522/14sp/lectures/lect05.pdf