CS450 Assignment 1

Katya Griffiths-Julien, Jerry Lau

**Task A**

This was completed using the Faljolet-Martin algorithm. The function trailing\_zeros is used to count the number of trailing zeros in the binary representation of the provided element in the csv.

We then used the deque to keep track of the maximum return value of the trailing\_zeros and also kept a counter to know when the return value is called for 10^i by the outside program.

Lastly, before returning the value, the counter is used so that improper values cannot be sent to the main. For example, in the trial data set, using the algorithm when i=1, only 10 values have parsed. However, the generated response would be 16 as 144 has 4 trailing zeros. Meaning that there would be 2^4=16 distinct values for 10 elements. We used a counter value saved in the deque so that the number of items from the algorithm cannot be greater than the number of parsed elements.

**Task B**

For this task, the instructions were not clear as to what type of implementation was required to complete it. We were given that the dataset was of Pareto type 1.

In the slides about streaming algorithms, counting the 2nd moment presented the solution for Alon-Matias-Szegedy Algorithm. From external sources, it is shown that the median was estimated using a random choice of the given dataset. Attempting to implement this solution in the constraint of a 100 item deque was not feasible in the sense that the algorithm requires the entire dataset to be saved then to do a random choice on the dataset to calculate the median and then update the value accordingly with further values. Thus, as it was not covered in the slides nor were we asked to use a specific method, a different method was found to figure out how to complete the task. Therefore, a different method was used to figure out how to complete the task.

It was decided that using an implementation similar to counting sort would be an efficient method in this dataset of Pareto type 1. In the counting sort, the initial step is to use the index of the array to determine the count of each value in the dataset. Then when looking for the median value, it can be determined by finding the halfway count of the number of items that were streamed.

In our given scenario, the deque is used to hold the number of counts of each item. Since the size of the deque is only limited to 100, all value above 100 will be held in the last bin. This only applies to Pareto type 1 dataset where the distribution is skewed right. Pareto type 1 dataset are generally used to describe the allocation of wealth and in many cases, a small percentage hold large values thus creating a skewed distribution. Given these constraints, a deque of size 100 is more than enough to categorize our values and determine the median.

When the function is asked to return the value for the median at 10^i, we calculate the cumulative sum of all the values in the deque as the value of i is not known. Since the deque is semi-sorted we can find the index which holds half of the cumulative sum calculated.

**Task C**

As explained in task B, the usage of the counting sort was modified to determine the mode. In this case, the mode is simply the maximum value of the streamed dataset into the deque with the index determining what the value is. Since the size of the deque is limited to 100, all values above are incremented in the last bin. Once again since the dataset is of Pareto type 1, it does not cause any issues as the skewness of the dataset will not change the functioning of the estimation.

When the function is asked to return the value for the mode at 10^i, all we have to do is find the maximum value in the deque with the index being the value of the stream that is the mode.

**Test parameters**

We used Google Colab to run the program and connect on the python 3 google compute engine. There was 0.9/12.7GB ram and 26.4/107.7GB disk connected. Some changes to the code were made such as specifically declaring the 'global' for memory1b and also having to change the way the dataset is streamed into the program.

**Test output**

trial.csv

TESTING YOUR CODE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Beginning stream input

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Result at stream element # 10:

1A: Distinct values: 10

1B: Median: 5.00

1C: Most frequent value: 1

[current memory sizes: A: 2776, B: 3360, C: 2872]

Result at stream element # 100:

1A: Distinct values: 16

1B: Median: 4.00

1C: Most frequent value: 4

[current memory sizes: A: 2776, B: 3408, C: 3016]

Result at stream element # 1000:

1A: Distinct values: 128

1B: Median: 4.00

1C: Most frequent value: 4

[current memory sizes: A: 2776, B: 3524, C: 3364]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Stream Terminated

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(peak memory usage was: 163740 )

test.csv

TESTING YOUR CODE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Beginning stream input

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Result at stream element # 10:

1A: Distinct values: 8

1B: Median: 3.00

1C: Most frequent value: 3

[current memory sizes: A: 2776, B: 3344, C: 2824]

Result at stream element # 100:

1A: Distinct values: 16

1B: Median: 3.00

1C: Most frequent value: 3

[current memory sizes: A: 2776, B: 3392, C: 2968]

Result at stream element # 1000:

1A: Distinct values: 32

1B: Median: 3.00

1C: Most frequent value: 3

[current memory sizes: A: 2776, B: 3540, C: 3412]

Result at stream element # 10000:

1A: Distinct values: 128

1B: Median: 3.00

1C: Most frequent value: 3

[current memory sizes: A: 2776, B: 3712, C: 3928]

Result at stream element # 100000:

1A: Distinct values: 512

1B: Median: 3.00

1C: Most frequent value: 3

[current memory sizes: A: 2776, B: 3716, C: 3940]

Result at stream element # 1000000:

1A: Distinct values: 8192

1B: Median: 3.00

1C: Most frequent value: 3

[current memory sizes: A: 2776, B: 3716, C: 3940]

Result at stream element # 10000000:

1A: Distinct values: 32768

1B: Median: 3.00

1C: Most frequent value: 3

[current memory sizes: A: 2776, B: 3716, C: 3940]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Stream Terminated

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(peak memory usage was: 163740 )

**Conclusion**

From the output, we can conclude that using the counting sort method was very successful in estimating the median and mode for the provided dataset. When we ran the estimator with the test dataset, the median and mode are the actual values in the provided dataset. However, this method is very suspectable to overfitting the data. Thus, the counting sort method can only be used for a Pareto type 1 dataset without too much overfitting. As for the increase in the exponential powers of the data stream, the tests shows that after 10000 the memory size does not increase. This is due to the limitation that was placed as a constraint since increasing the size will show further increases to the memory size used when going past 10^5 items.