Q3

1. Create your own histogram function that sorts N samples into M bins and time how long it takes to run for M=1000 and different values of N.

My histogram function will sort each sample into a bin by taking each sample’s value compared to the lower part of the range and dividing that value by the size of the range divided by the number of bins. i.e. the bin number of s ample will be (value-range\_min)/(size\_of\_range/M).

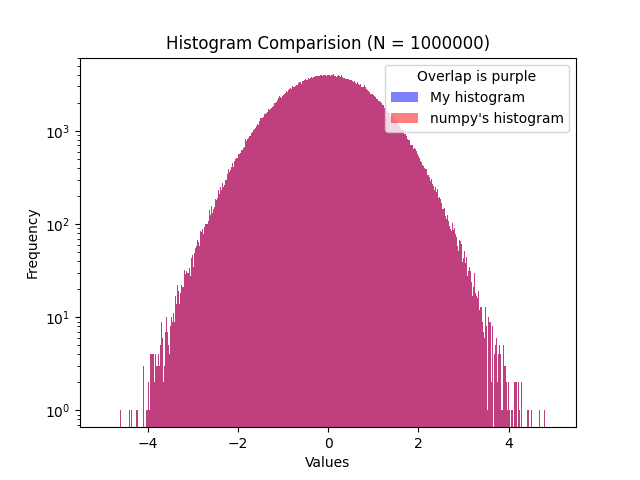
Timing results:

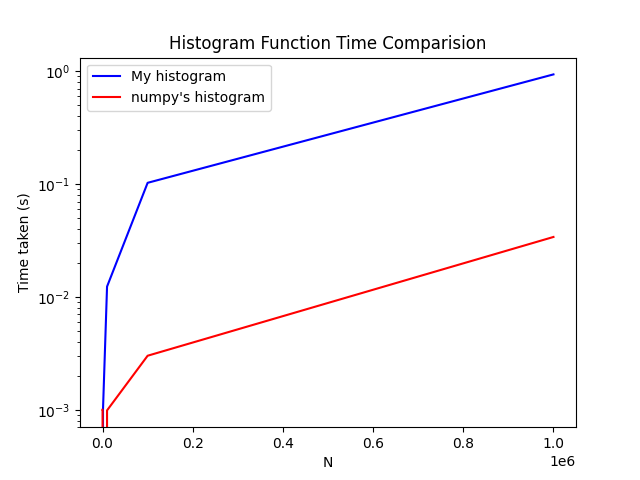
|  |  |
| --- | --- |
| N | Time (s) |
| 10 | 0.0 |
| 100 | 0.000965118408203125 |
| 1000 | 0.0019991397857666016 |
| 10000 | 0.025830745697021484 |
| 100000 | 0.21095919609069824 |
| 1000000 | 2.0834832191467285 |

Note: It rounds down to 0.0 if the time is small enough.

1. How does your histogram function’s time compare to numpy’s built in histogram function?

I made sure to check that histogram functions returned the same values (with the exception of numpy’s hist returning the final edge of the bin, while mine stops one short so that the two arrays are equal in length).



Then comparing the functions’ time complexity:

Note: numpy’s function seems to be slower at N=10 than N=100 due to the rounding, unsure why.

My method is slower, likely due to np's methods directly using C code while I use numpy's methods to access arrays. My method seems to have slightly worse time complexity: the proportional difference between them grows with n (from ~10x faster at 10k to ~50x faster at 1 million). This is strange since my method is O(N).'''