Analysis of Time Complexity of Sorting Algorithms

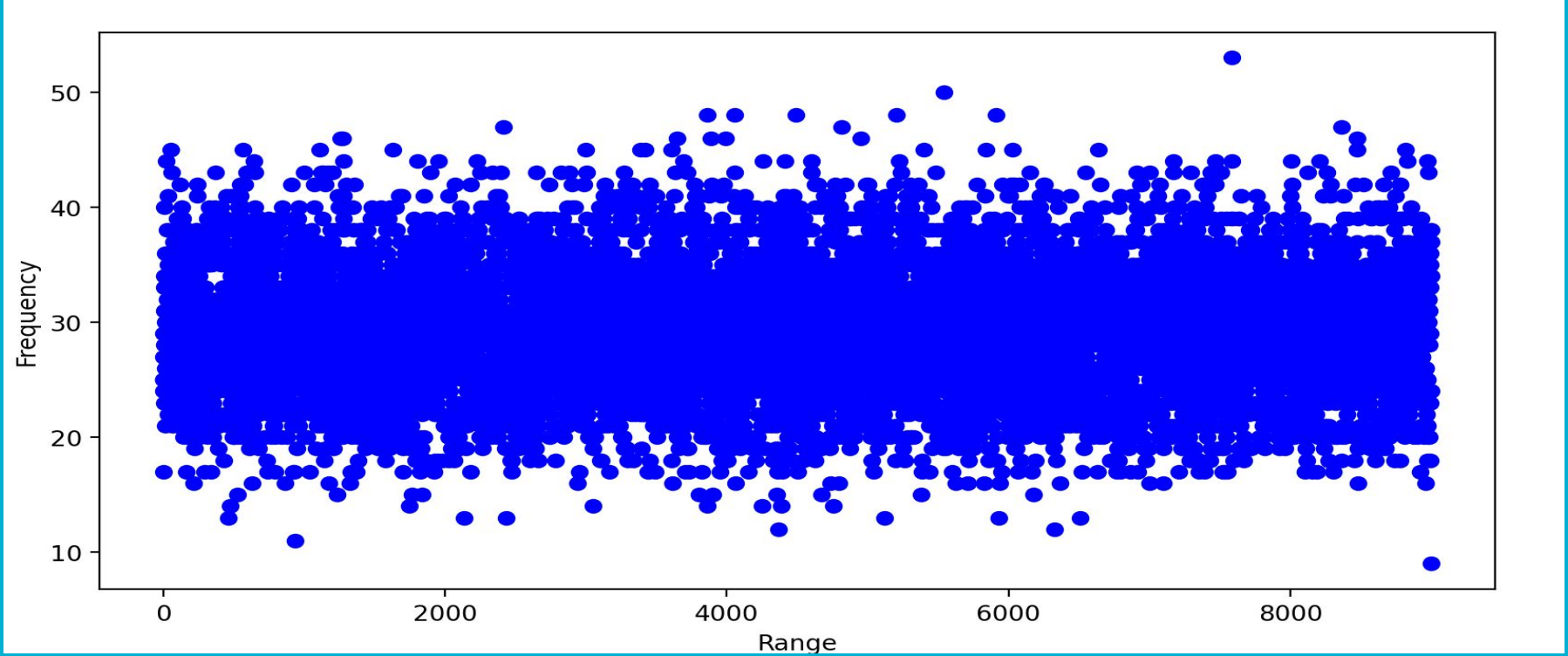
—

By Arghyadeep Saha (2020CSB001) CST, 4th semester

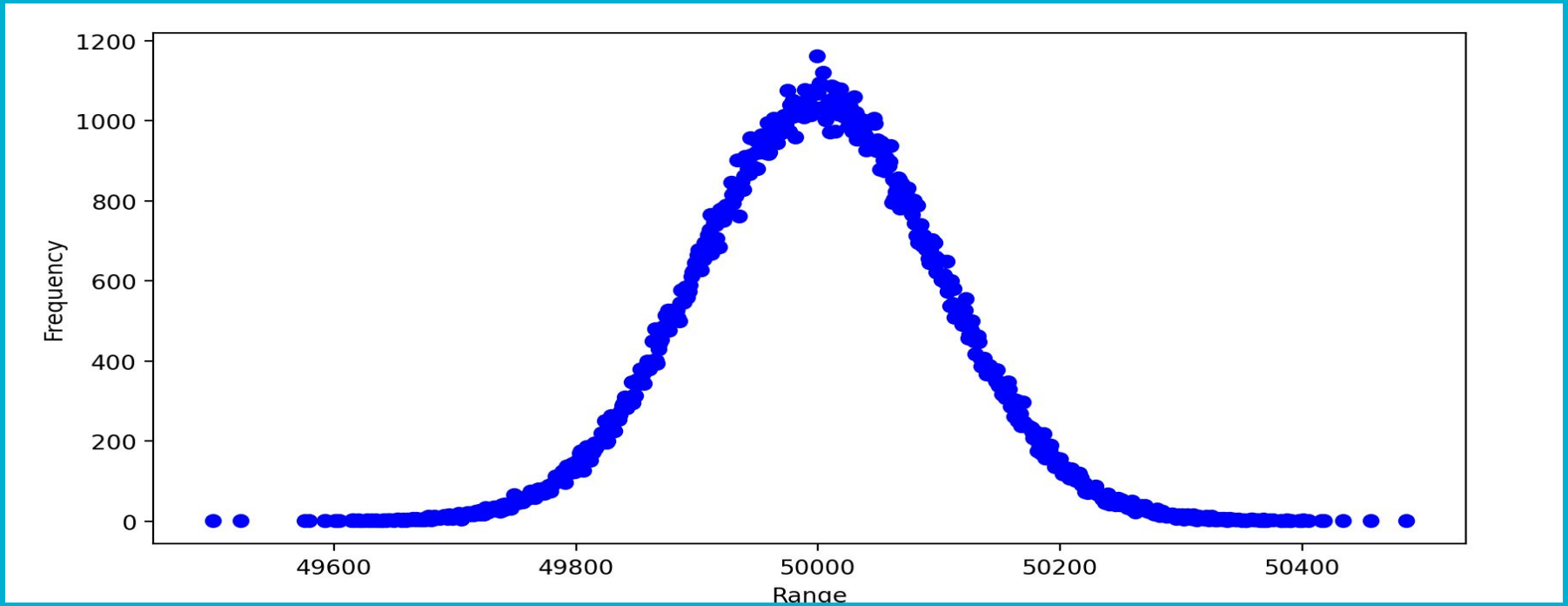
Test Dataset Generation

- Normal and uniform dataset of various length from 2 to 2^18 lengths was generated using numpy in python.
- Two types of datasets were generated to differentiate between various behaviours based on the distributions of the numbers in the dataset
- The main purpose of this dataset is to study the relationship between various theoretical and experimental time and operation complexity
- In the following section we will see the differences between theoretical and experimental values and make important conclusions
- Matplotlib and numpy packages were used for the experiment
- Here is the github link of the code of the entire experiment - https://github.com/arghya17/Algorithm_lab

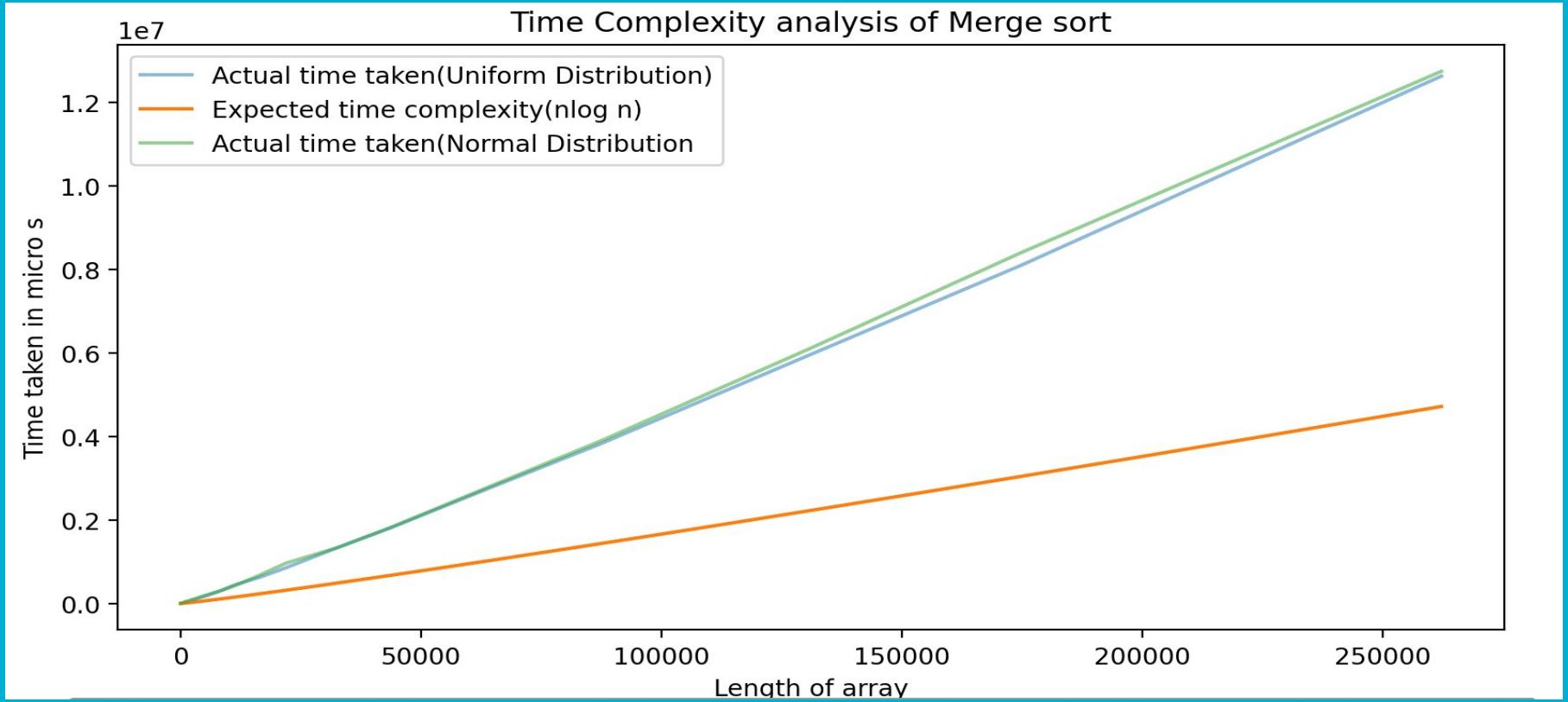
Uniform Distribution

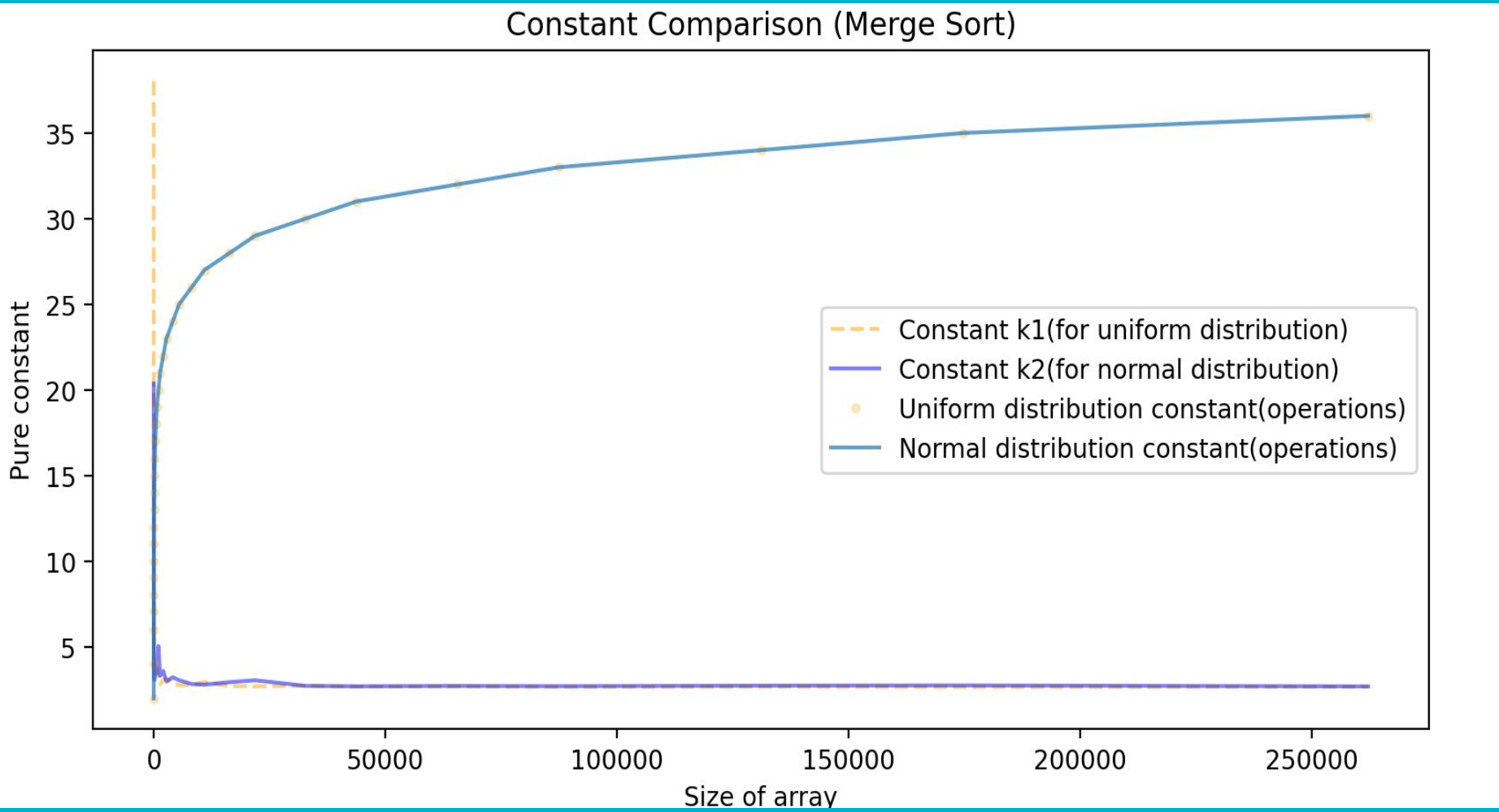


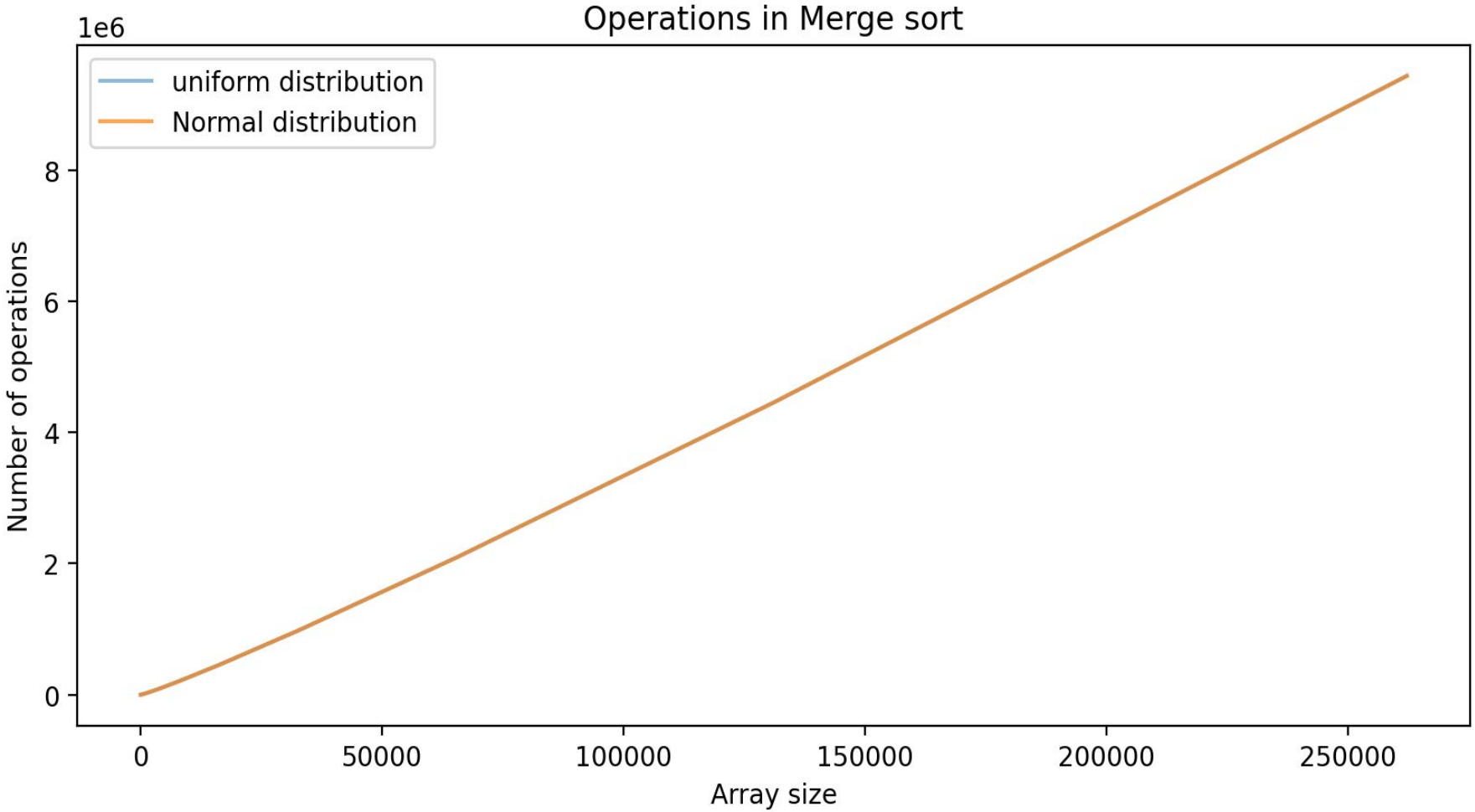
Normal distribution



Merge Sort Analysis




Constant Comparison (Merge Sort)
Pure constant
35
30
25
20
15
10
5
0
50000
100000
150000
200000
250000
Size of array
Constant k1(for uniform distribution)
Constant k2(for normal distribution)
Uniform distribution constant(operations)
Normal distribution constant(operations)


Operations in Merge sort
1e6
uniform distribution
Normal distribution
Number of operations
0
2
4
6
8
0
50000
100000
150000
200000
250000
Array size

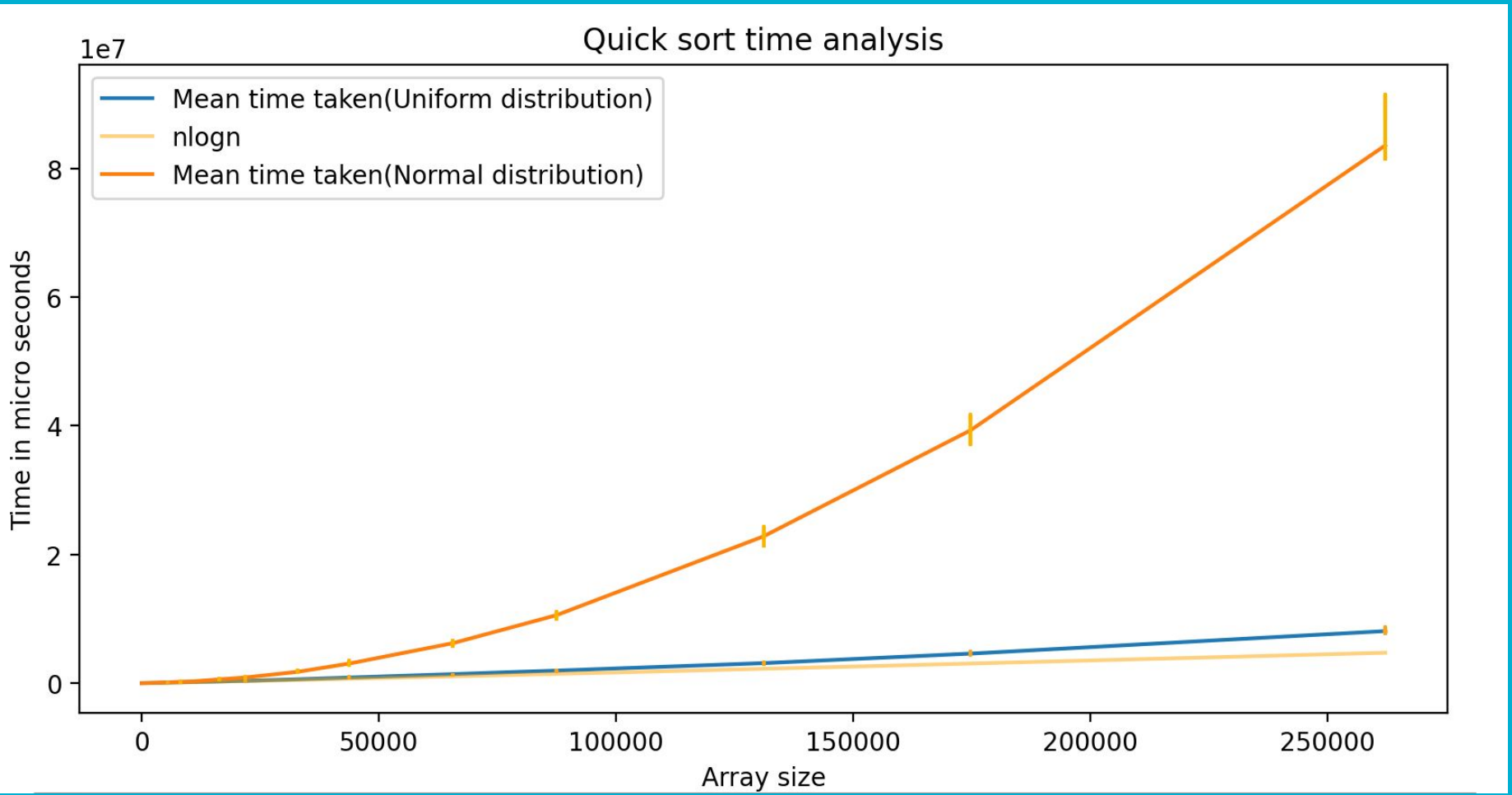
Observations

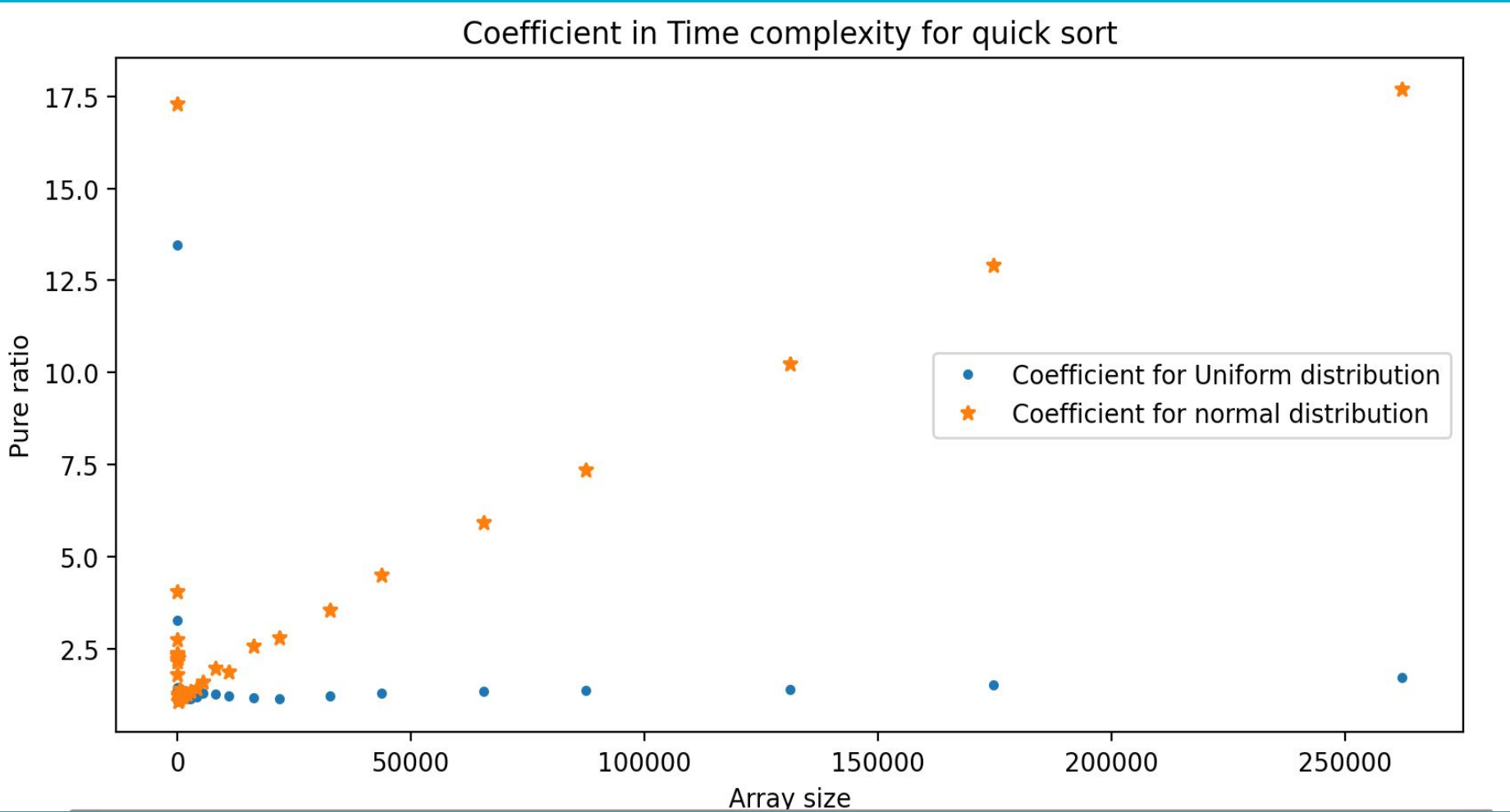
- From the above graphs it is clear that 0(nlog n) is a fairly good estimation of the time complexity of merge sort since the ratio of the time complexity with nlog n is almost constant
- Note that the time complexity of the merge sort does not vary with the distribution of data in the array
- Even the operations performed for normal and uniform datasets remain almost constant
- Thus we can conclude that merge sort gives a very good n logn sorting algorithm which can be used without thinking about the data distribution in the arrays
- Drawback - O(n) auxiliary space complexity of the merge sort

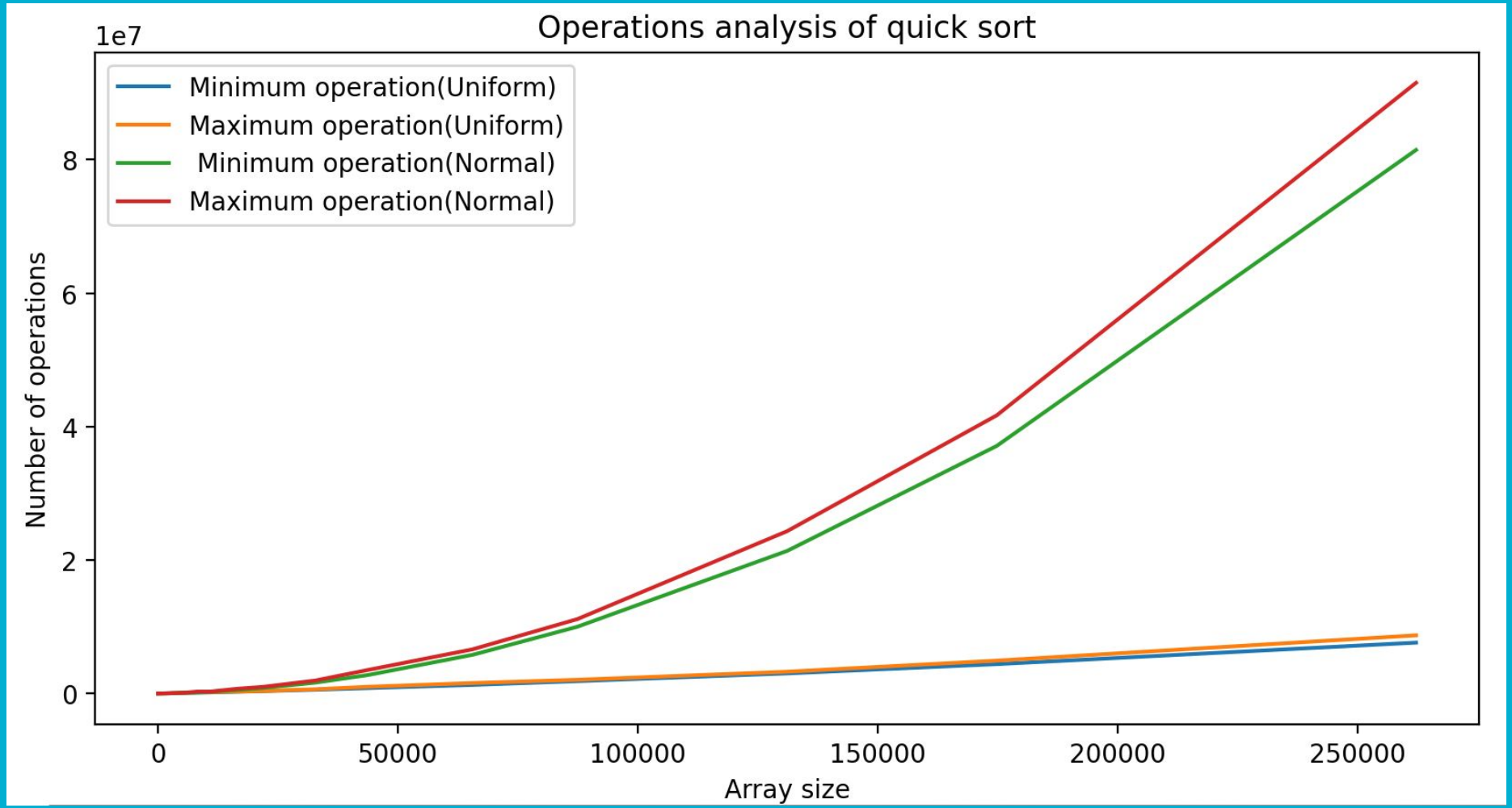
Quick Sort Analysis

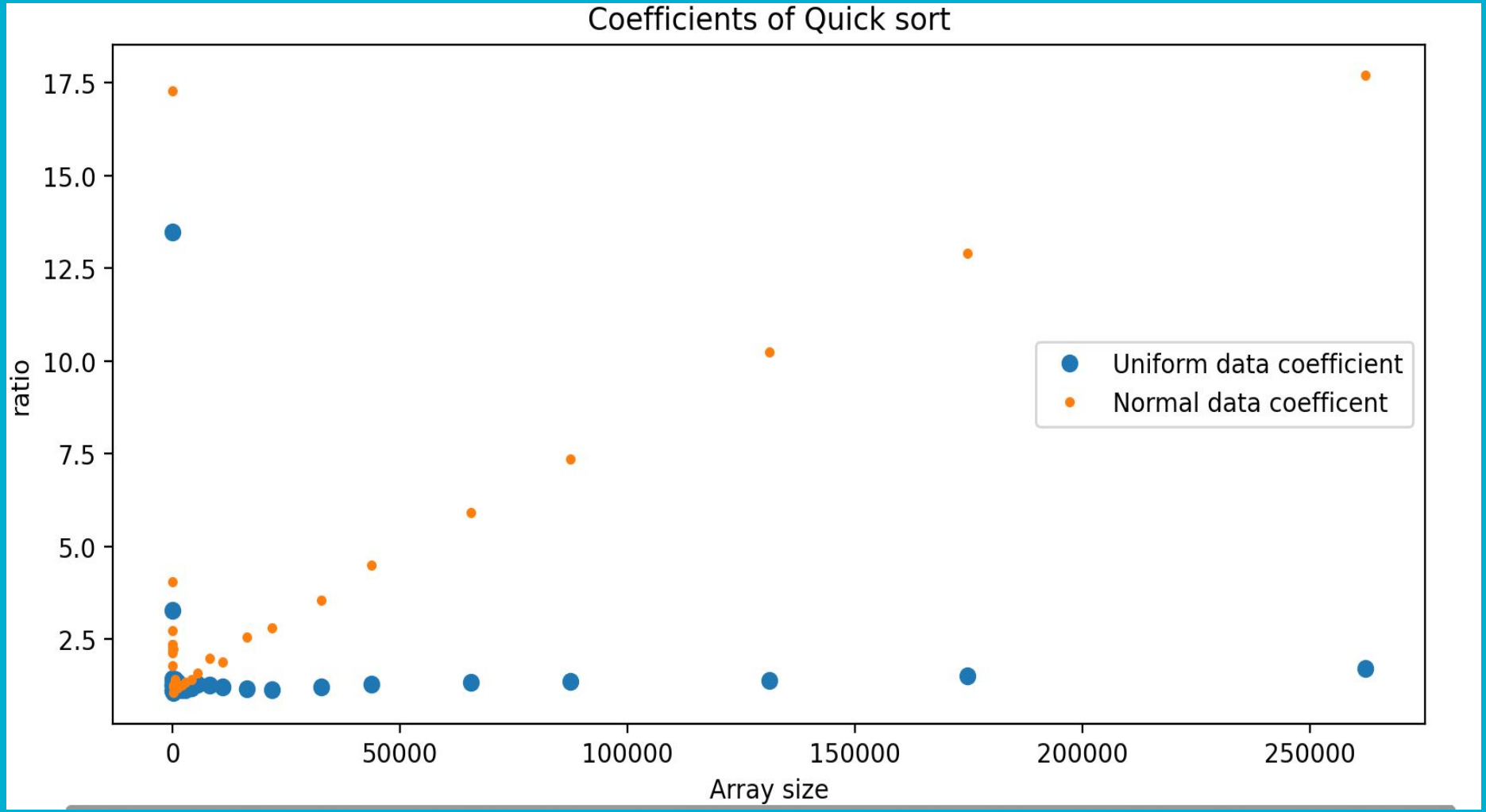
Procedure used-

- 10 permutations of the each dataset is used to check the variation of the time complexity based on the different arrangement of the data in the array
- In case of measuring the number of operations performed only the number of swaps were taken into account
- The test data was generated by quick_test_data.py
- For the implementation of the quick sort algorithm look into the quick sort file in the github repository


Quick sort time analysis
1e7
Mean time taken(Uniform distribution)
nlogn
Mean time taken(Normal distribution)
Time in micro seconds
0
2
4
6
8
0
50000
100000
150000
200000
250000
Array size


Coefficient in Time complexity for quick sort
Pure ratio
Array size
Coefficient for Uniform distribution
Coefficient for normal distribution
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0
50000
100000
150000
200000
250000


Operations analysis of quick sort
1e7
Minimum operation(Uniform)
Maximum operation(Uniform)
Minimum operation(Normal)
Maximum operation(Normal)
Number of operations
0
2
4
6
8
0
50000
100000
150000
200000
250000
Array size

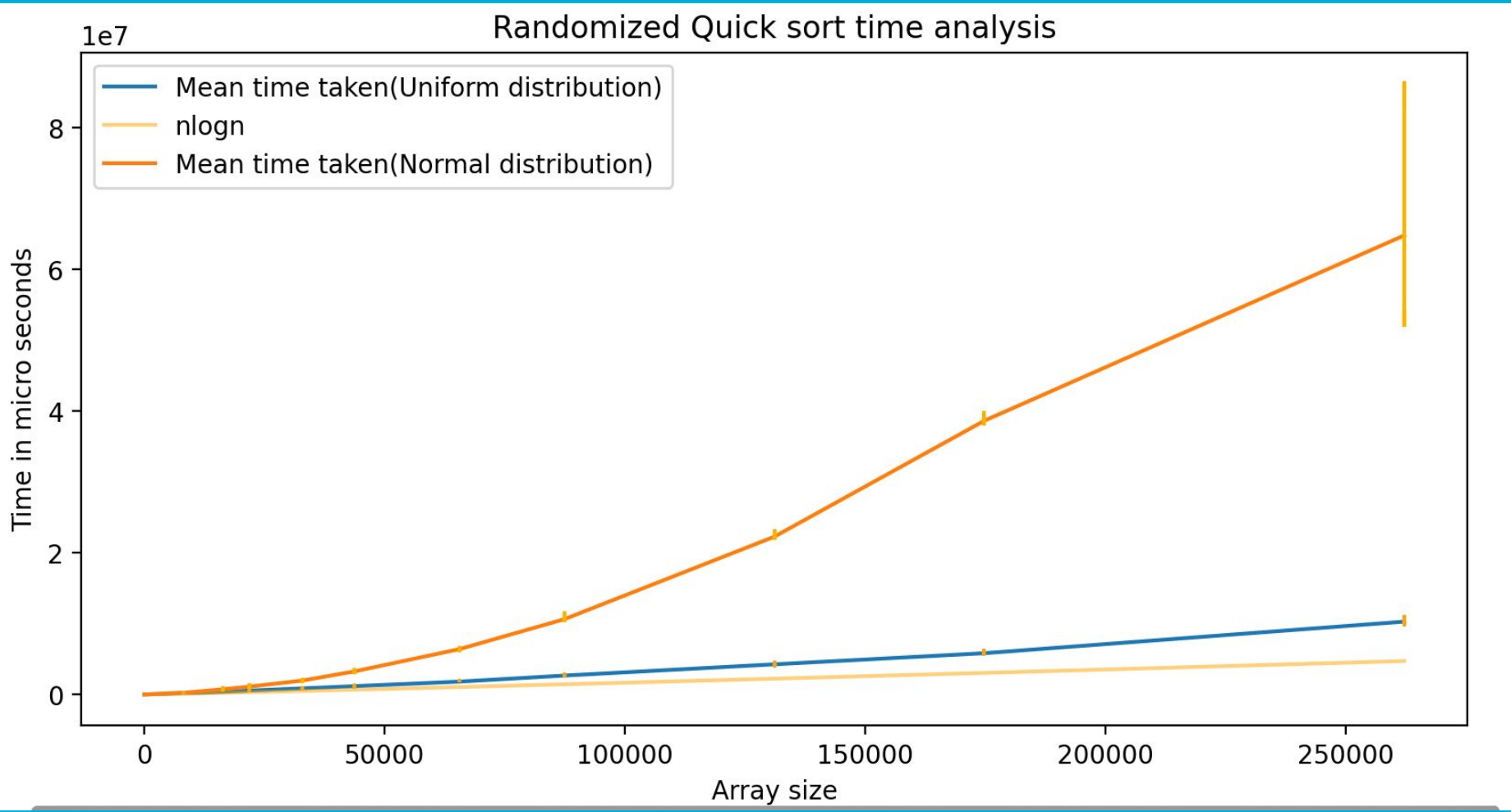

Coefficients of Quick sort
ratio
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0
50000
100000
150000
200000
250000
Array size
Uniform data coefficient
Normal data coefficent

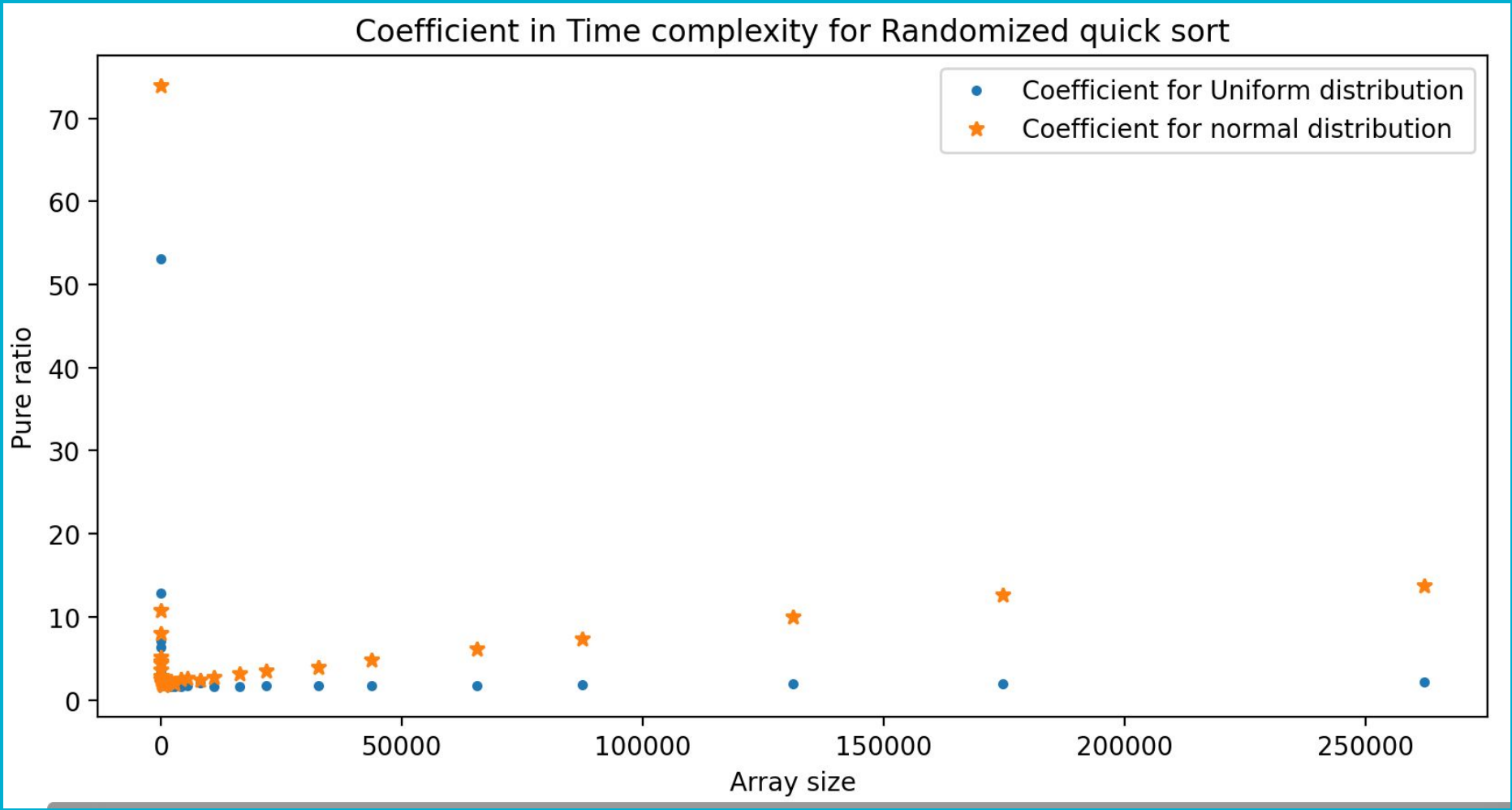
Observations

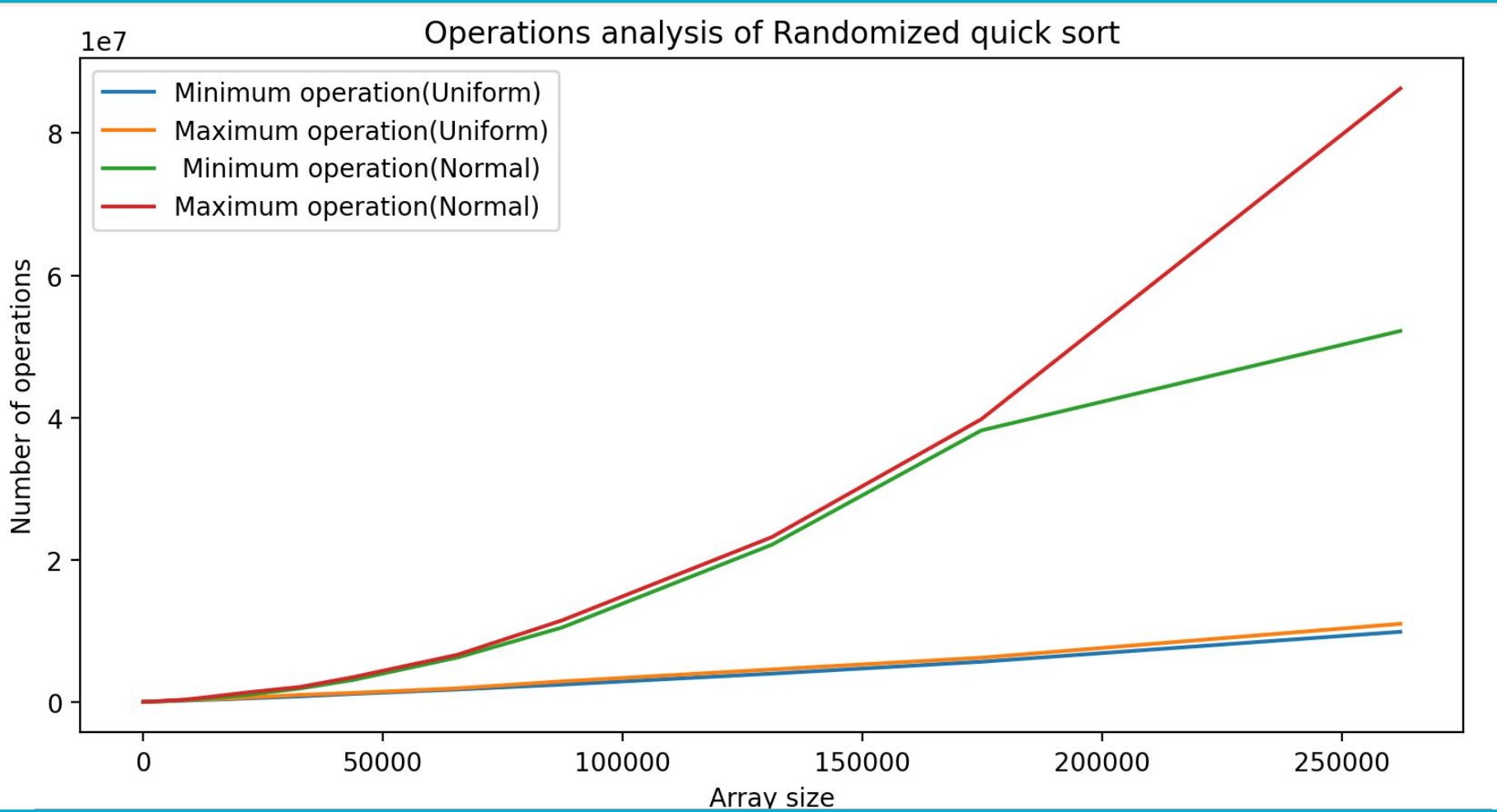
- The ratio for quicksort did not come a constant because there is no deterministic complexity of quick sort and it varies between 0(nlog n) and 0(n^2) based on the choice of the pivotal element. The complexity of quick sort can be considered as a probability distribution
- The time complexity also varied based on the distribution of the data points in the data set. The normal dataset gave a worse complexity simply because though there is a better chance or probability of getting a good choice of pivot during the first split but the chances of getting median splits for the smaller splits deep into the recursive calls of the quick sort becomes very difficult for normal distribution but for uniform distribution the chance of getting a fifty fifty split remains same throughout reducing the complexity
- Same thing occurs with the number of operations

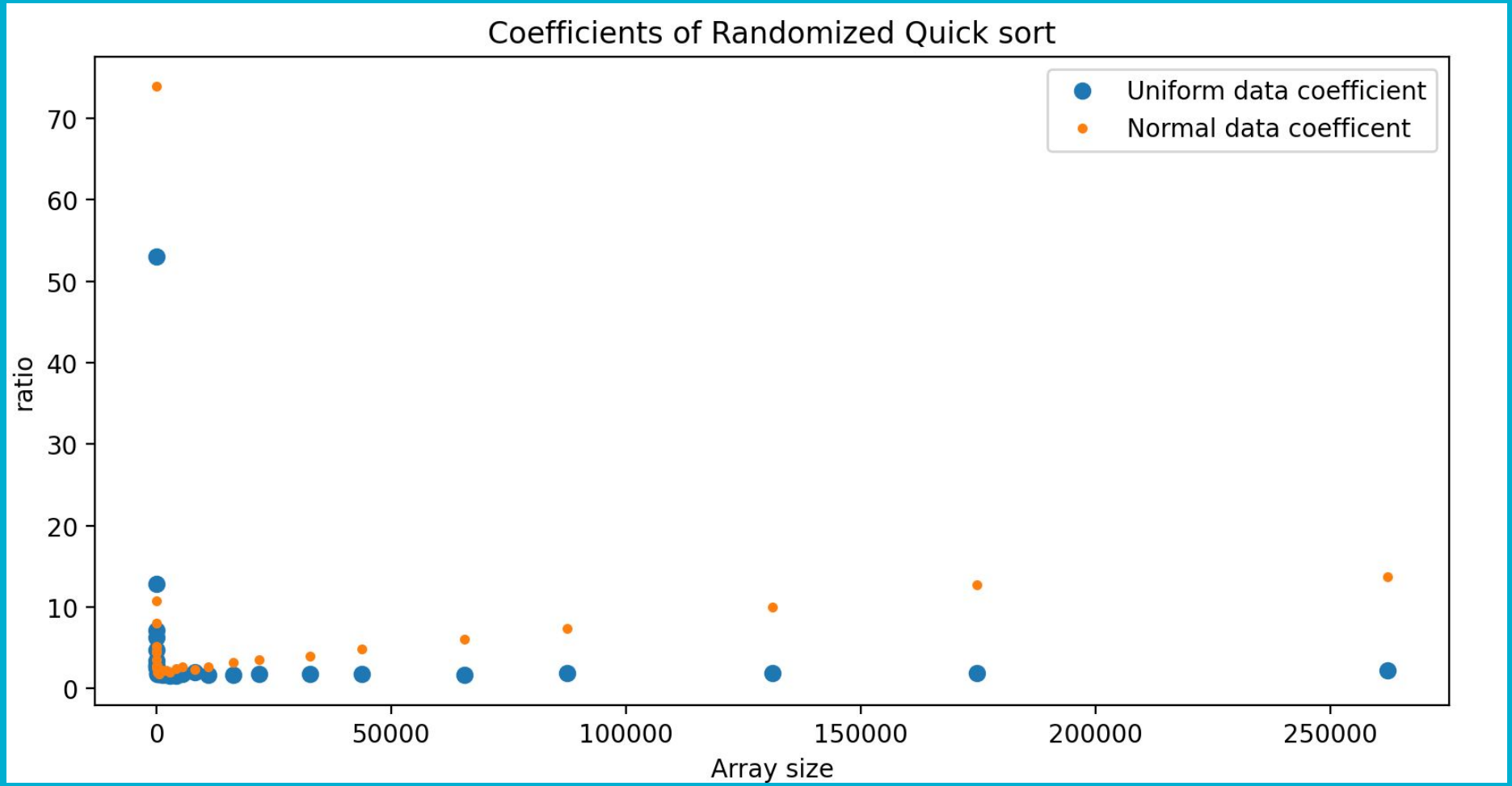
Randomized quick sort

- Here we try to introduce random choice of the pivot to see if it affects the complexity of quick sort and see whether it produces a better result than normal quick sort
- This idea was the increase the probability of getting a good split by introducing randomization
- Randint() was used to generate the random pivotal index


Randomized Quick sort time analysis
1e7
Mean time taken(Uniform distribution)
nlogn
Mean time taken(Normal distribution)
Time in micro seconds
0
2
4
6
8
0
50000
100000
150000
200000
250000
Array size


Coefficient in Time complexity for Randomized quick sort
Coefficient for Uniform distribution
Coefficient for normal distribution
Pure ratio
Array size
0
10
20
30
40
50
60
70
0
50000
100000
150000
200000
250000


Operations analysis of Randomized quick sort
1e7
Minimum operation(Uniform)
Maximum operation(Uniform)
Minimum operation(Normal)
Maximum operation(Normal)
Number of operations
0
2
4
6
8
0
50000
100000
150000
200000
250000
Array size

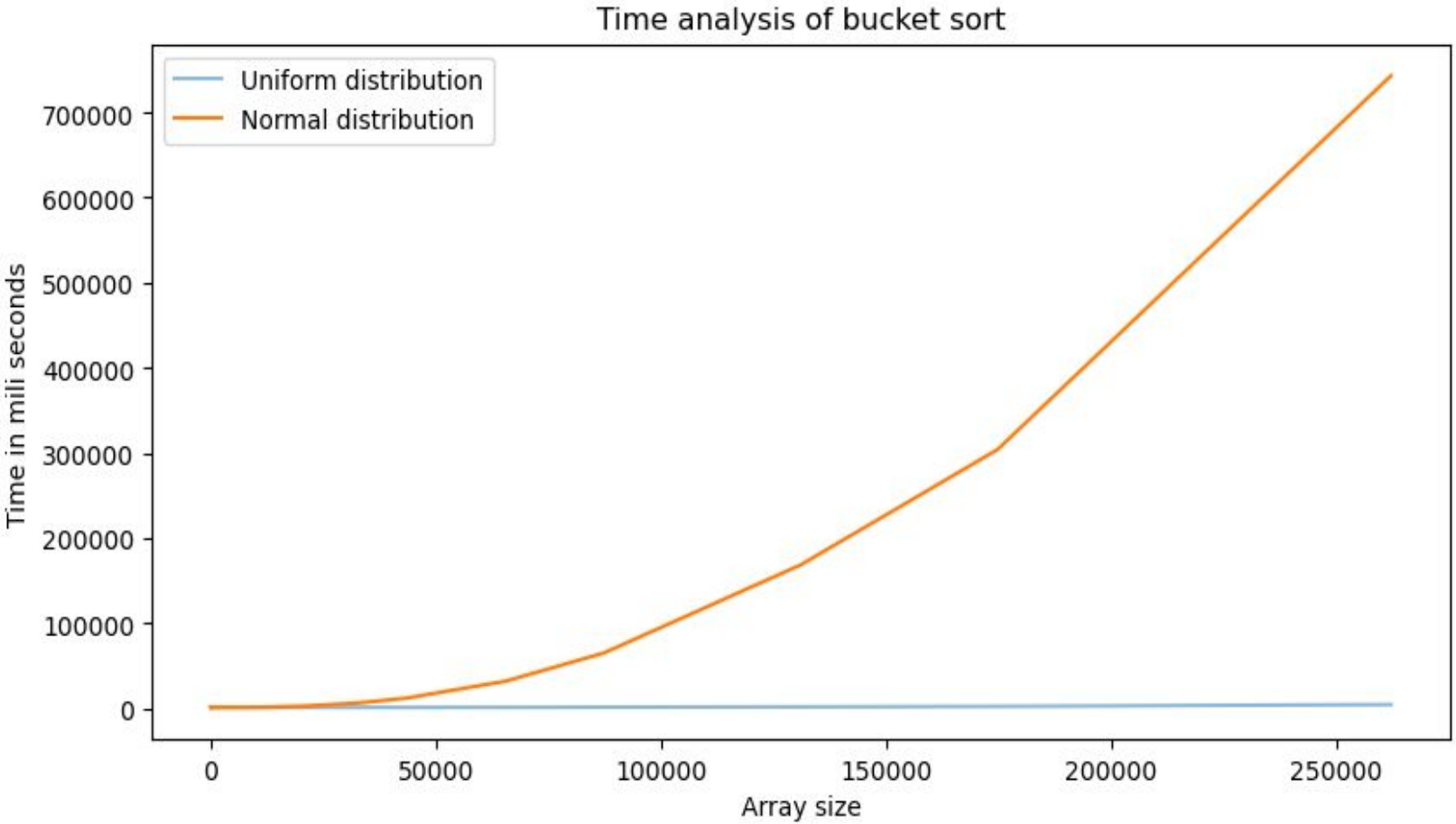

Coefficients of Randomized Quick sort
Uniform data coefficient
Normal data coefficent
ratio
Array size
0
10
20
30
40
50
60
70
0
50000
100000
150000
200000
250000

Observations

- We see that randomization improves the probability of getting a better split more but it still does not give us any deterministic complexity with the choice of pivot
- Even in randomization the normal dataset did not perform very well due to its unique bell shaped curve
- But it can be noted that randomization improved the complexity for normal distribution compared to the uniform distribution. More experiments need to be performed before drawing a exact conclusion about the above hypothesis

Bucket Sort

- Bucket sort is a seemingly linear time sorting algorithm but can give 0(n^2) algorithm on worst case
- Bucket sort uses insertion sort to sort the buckets so if all elements are in the same bucket then the complexity will be n square
- Insertion sort is used since it is difficult to determine the number of elements in each bucket resulting in a linked list to be used and insertion sort is only suitable for sorting the linked lists

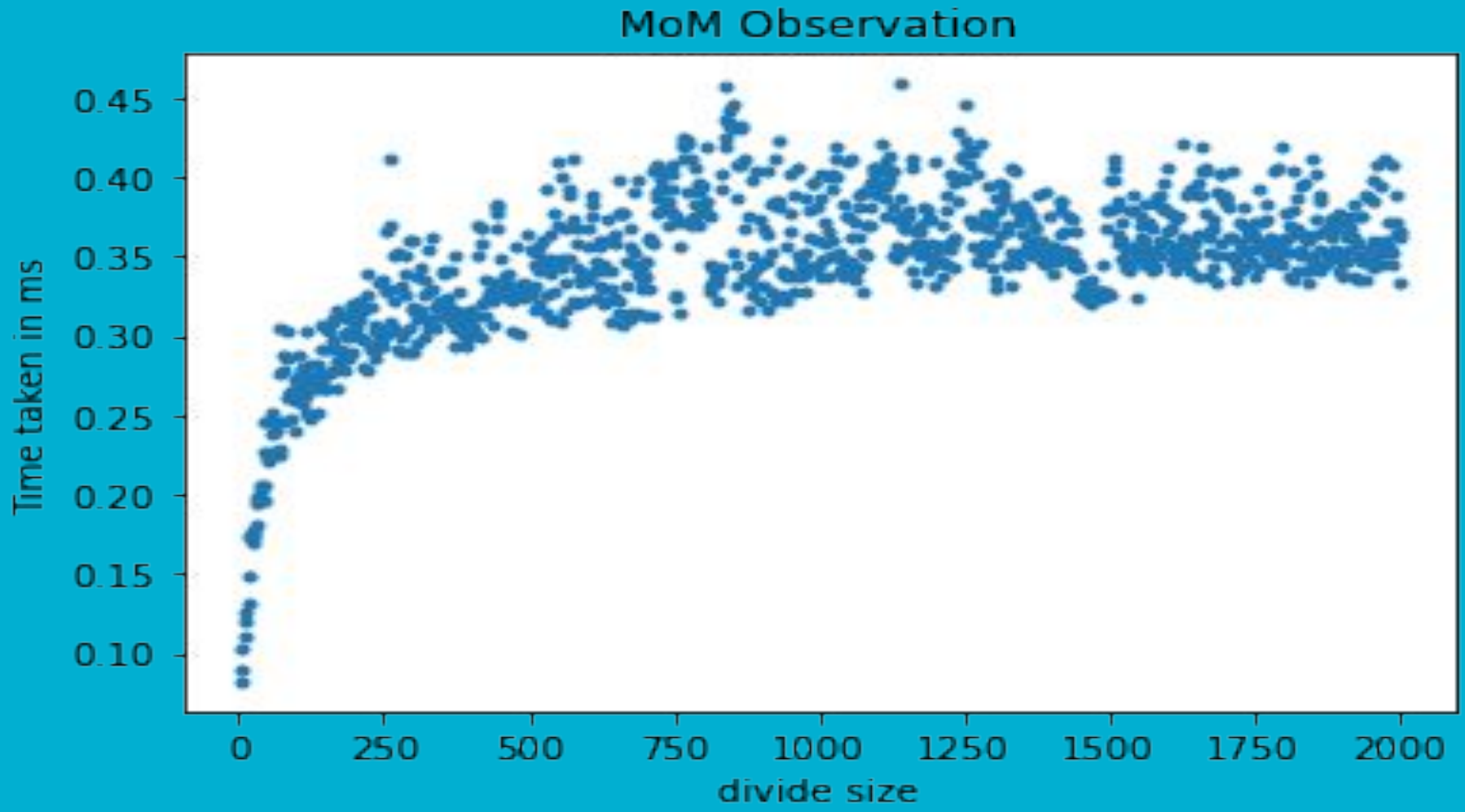

Time analysis of bucket sort
Uniform distribution
Normal distribution
Time in mili seconds
700000
600000
500000
400000
300000
200000
100000
0
0
50000
100000
150000
200000
250000
Array size

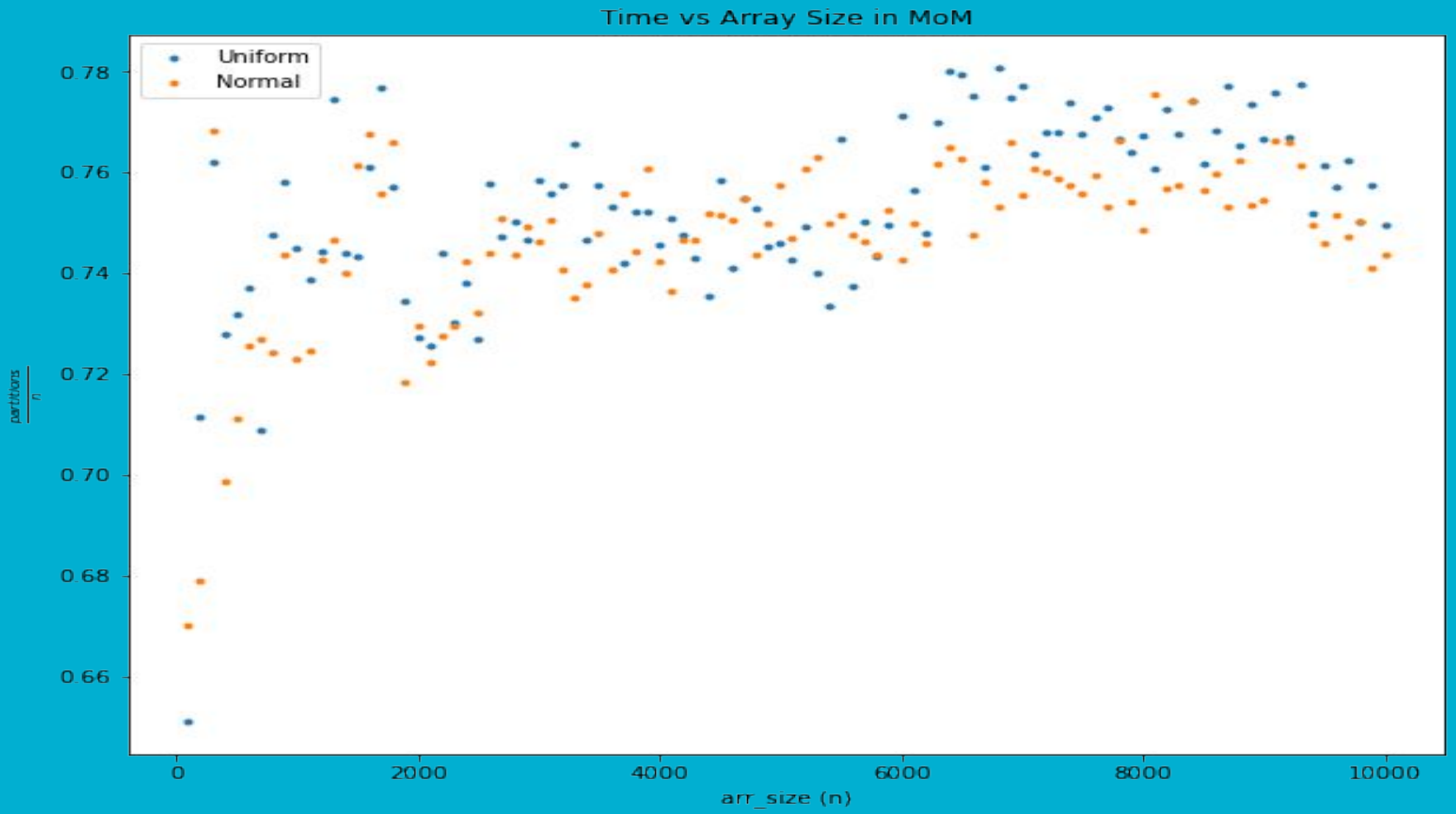
Observations

- The normal dataset gives worse complexity simply because the fact that all the data points are concentrated around the mean. So some of the buckets contains more elements while other do not contain any
- As a result this leads to a very bad complexity of 0(n^2)
- Uniform dataset have a better probability of getting one element in each bucket than the normal datasets as a result having a order n complexity

Median of Medians

- Median of medians is an algorithm designed to choose a better pivot for quick sort like algorithm like median or rank finding and also quick sort
- The goal was to choose a better pivot in order n i.e. 0(n) time to reduce the time complexity
- The median of medians does not provide a deterministic algorithm but surely reduces the worst case from 0(n^2)
- Here we experiment with the size of the buckets in median of medians algorithm and also with the uniform and normal dataset


MoM Observation
Time taken in ms
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0
250
500
750
1000
1250
1500
1750
2000
divide size


Time vs Array Size in MoM
Uniform
Normal
0.78
0.76
0.74
0.72
0.70
0.68
0.66
partitions / n
0
2000
4000
6000
8000
10000
arr_size (n)

Observations

- The choice of the divide size of median of medians algorithm is crucial as other than size 5 the algorithm does not remain linear making the MOM algorithm useless
- We can also see that the choice of pivot using median of medians actually improves the quick sort algorithm significantly compared to the randomization strategy.