

Research Mid-Term Report

Faster Sorting Algorithm by Employing Fundamentals of Statistics

January 19, 2018

Soumya Mishra

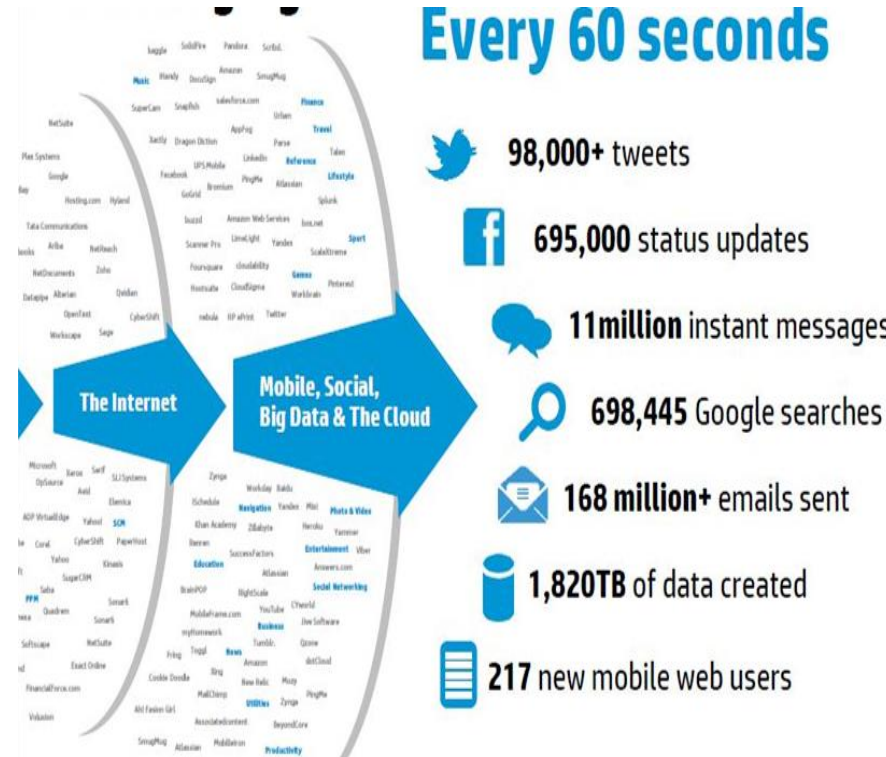
https://github.com/SoumyaMishra8/Mishra-AOS_JR_SortAlg

WHY ARE SORTING ALGORITHMS IMPORTANT?

- 25% - 50% of what computers do require sorting
 - Efficiency, speed, processing data
 - Search Engines (sort through website to find relevant one)
 - Graphical processes (layering is necessary)
 - Government organizations, financial institutions, commercial enterprises (transactions, accounts)

WHAT IS BIG DATA?

- Collected by the Government, NASA, Research Institutions
- Data sets are so large and complex
- IT systems have trouble processing data
- Volume, veracity, variety, and velocity
- Challenge of indexing, searching, transferring data



<https://onlinembapage.com/data-analytics-mba/>

SORTING BIG DATA - Merge Sort

Cases:

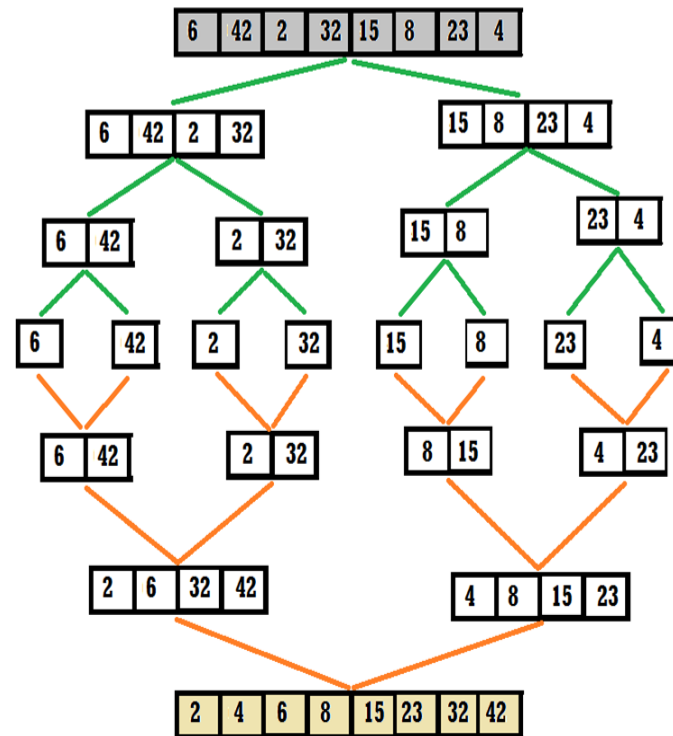
Best case: $O(n \log(n))$

worst case: $O(n \log(n))$

average case: $O(n \log(n))$

Issues:

- comparison-based = too many passes
- each array = own space in memory



SORTING BIG DATA - Bucket Sort

Cases:

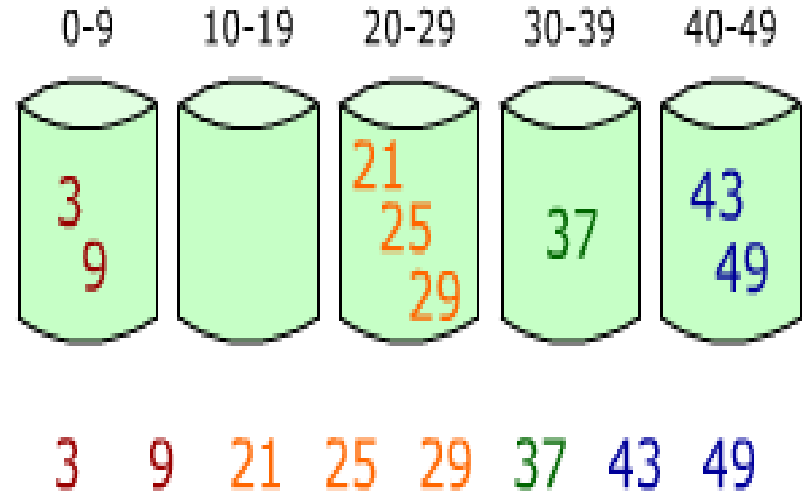
Best case: $O(n+k)$

worst case: $O(n^2)$

average case: $O(n^2)$

Issues:

- wastes memory
- does not work well for data that follows certain types of distributions



SORTING BIG DATA - Insertion Sort

Cases:

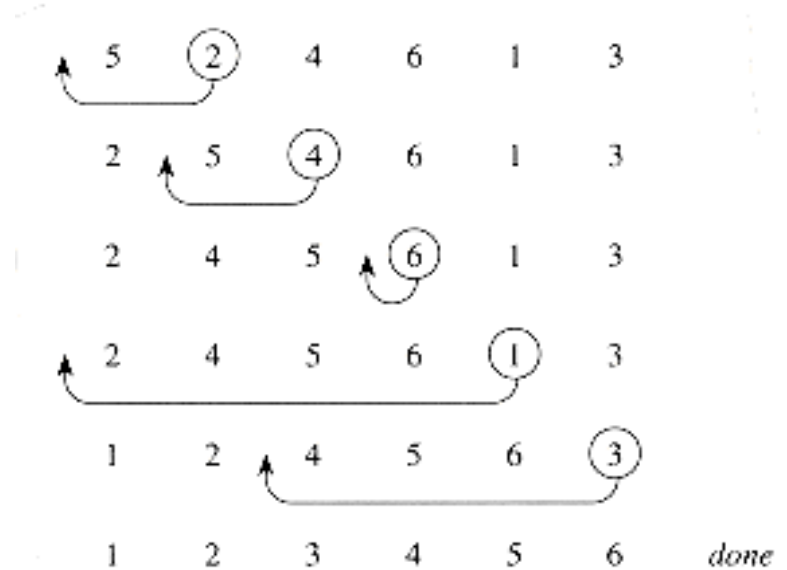
Best case: $O(n)$

worst case: $O(n^2)$

average case: $O(n^2)$

Issues:

- comparison-based = reliant on passes
- uses up a lot of memory



<https://www.ee.ryerson.ca/~courses/coe428/sorting/insertionsort.html>


WHY DO WE NEED NEW SORTING ALGORITHMS?



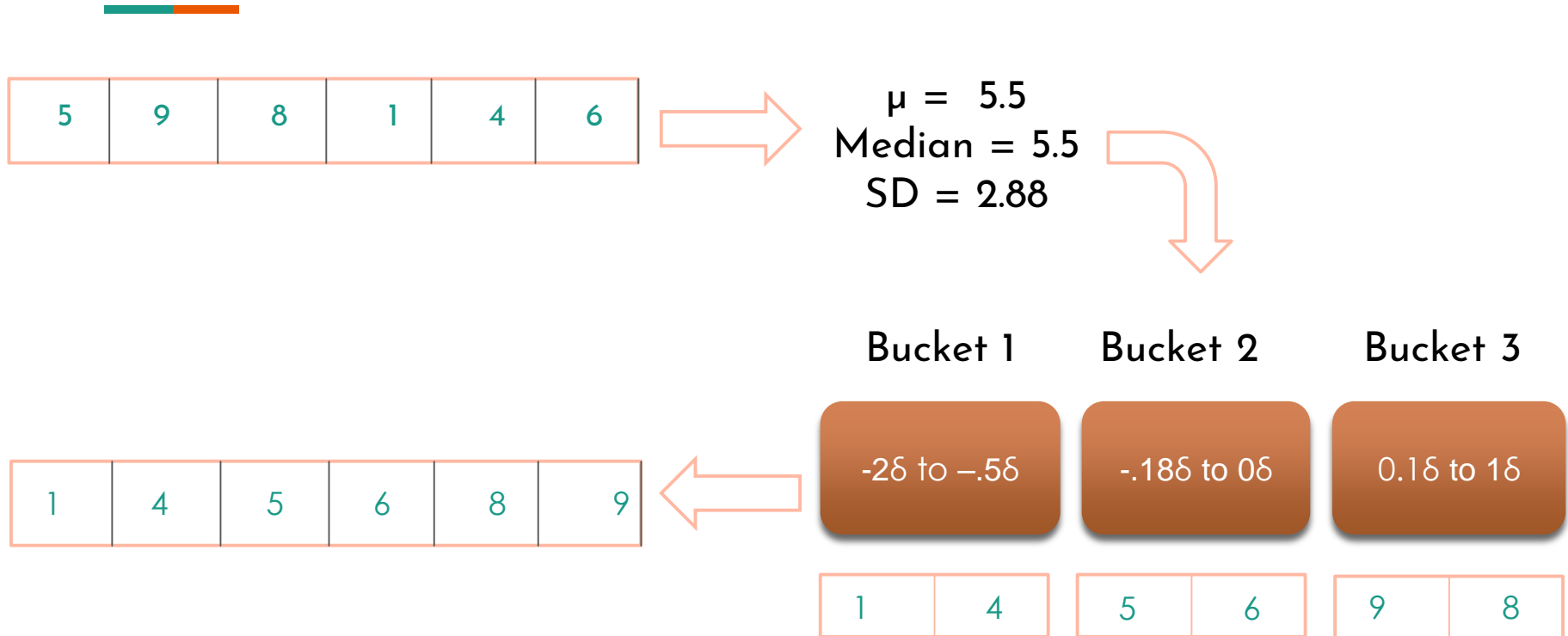
CURRENT SORTING ALGORITHMS REQUIRE TOO MUCH MEMORY, TOO MANY PASSES AND TOO MUCH TIME

- Implement neural networks
- Using more servers, linked lists, multiple parallel processors
- Different ways of distributing data amongst servers
- Implementing both GPUs and CPUs

A BETTER AND FASTER WAY TO SORT

- 
- REDUCE COMPUTE TIME AND RAM REQUIREMENTS
 - USE STATISTICAL ANALYSIS TO DETERMINE BUCKET RANGES BASED ON DATA SET
 - <1000 ELEMENTS IN EACH BUCKET (NUMBER OF ELEMENTS ARE EQUAL)
 - SHOULD BE ABLE TO SORT DATA SETS THAT FOLLOW ANY TYPE OF DISTRIBUTION
 - WORST CASE OF $O(N)$

A VISUAL REPRESENTATION



VARIABLES, CONSTANTS, CONTROLS

- ❖ **Independent variable** – using statistical analysis to set the parameters of the bucket
- ❖ **Dependent variable** – time complexity using Big-O notation; time it takes for algorithm to sort n elements
- ❖ **Constants**
 - Memory available
 - Laptop and processor
 - Bucket sort

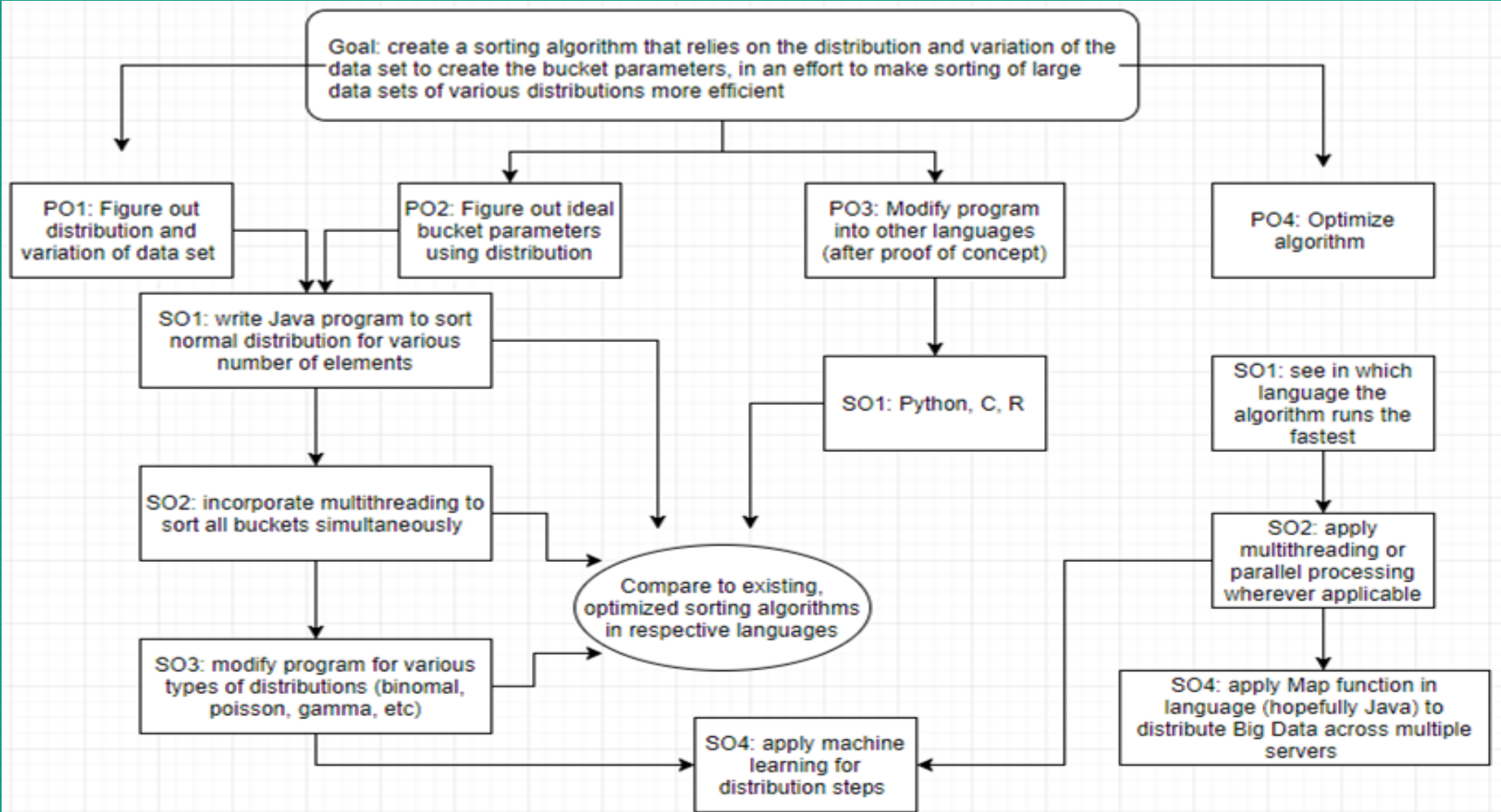
VARIABLES, CONSTANTS, CONTROLS



❖ Constants (continued)

- Merge sort
- Array.parallelSort
- Multiple processing units (6 node network)

❖ Control – run time for bucket sort and merge sort for the same number of elements that are being tested



PROJECT STATUS

- 1) Programmed “backbone” code in Java
 - a) talked to Mr. Kumar about project
- 2) Compared runtime of Java code to runtime of Parallel Sort
- 3) Programmed code in Python - same code as Java
- 4) Compared runtime of Python code to runtime of QuickSort in Python
- 5) Tried to fix Python code using NumPy and Pandas - collected data

PROJECT STATUS

6) Started examining how to simplify algorithm to make it applicable to different types of distributions

- a) integration technique

- b) box and whisker plot technique - suggested by Mr. Writer

 - i) talked to AOS Alumni about integration technique

7) Currently writing programs in Eclipse using Apache Commons to see how much more efficient integration technique is

8) Plan to write program in Mathematica for box and whisker plot technique (and possibly integration technique as well)

OVERVIEW OF JAVA CODE

1. Obtain the MIN, MAX, MEAN of the data set by implementing a for loop
2. Create a 2D array for buckets - each bucket has 1000 elements
3. Use formula for z-score to determine position of bucket index along a range
4. When a bucket reached 1000 elements, data point would be sent over to next bucket
5. Merge sort is used to sort each bucket
6. Arrays are concatenated and then given to user

OVERVIEW OF JAVA CODE

- ❖ Worked well for the normally distributed data fed into the program
 - performance will deteriorate with different distributions
 - efficiency keeps decreasing (from being 98% more efficient to being 47% more efficient as number of data elements increase)
 - there must be some underlying problem within the code that increases the run time - for loops?

COLLECTED DATA

Effect of Setting Bucket Parameters on Run Time

Elements	New Sort (s)	Merge Sort (s)	Parallel Sort (s)	Bucket Sort (s)
1000	0.01	0.22	0.26	0.19
10,000	0.01	0.36	0.35	0.42
100,000	0.07	0.47	0.26	0.52
1000,000	0.15	1.18	0.61	0.74
10,000,000	0.42	5.6	0.99	1.34
100,000,000	3.26	10.31	5.81	4.65
1,000,000,000	HEAP ERROR			

OVERVIEW OF PYTHON CODE

- ❖ Similar to Java code in the method and the way to calculate bucket index
- ❖ Being compared to Insertion Sort - default Python sorting algorithm
- ❖ Does not work as expected because of execution delays in Python itself
- ❖ The two for loops are causing run times to increase substantially

COLLECTED DATA

Comparing run times of Insertion Sort and New Sort

Elements	Python Sort			New Sort		
	Start Time (s)	End Time (s)	Total Time (s)	Start Time (s)	End Time (s)	Total Time (s)
1000	16.29	16.32	0.03	27.11	27.18	0.06
10,000	13.76	13.82	0.06	24.13	24.25	0.12
100,000	38.08	38.26	0.18	10.49	11.30	0.81
1,000,000	4.98	8.52	3.54	42.36	48.70	6.34
10,000,000	8.01	48.71	40.69	47.59	104.15	56.57

COLLECTED DATA

Comparing run times of Insertion Sort and New Sort

Elements	Python Sort			New Sort - with NumPy Array		
	Start Time (s)	End Time (s)	Total Time (s)	Start Time (s)	End Time (s)	Total Time (s)
1000	57.98	57.98	0.00	4.02	4.02	0.00
10,000	4.13	4.14	0.02	48.44	48.58	0.14
100,000	25.69	25.78	0.10	13.70	14.93	1.23
1,000,000	15.06	18.03	2.98	0.59	15.80	16.40

- Run time has significantly decreased from before
- Can time still be cut using numpy array for the 2D list as well?

COLLECTED DATA

Python - Sort with Complete NumPy (Sort, Arrays)					
Elements	Time for NumPy Array (s)	Time for Stat Collection (s)	Time for Bucketing (s)	Total Time (s)	Sort
1000	0	0.000	0.023	0.023	0.000
10,000	0	0.000	0.157	0.162	0.005
100,000	0.031	0.000	1.661	1.926	0.265
1,000,000	0.267	0.013	16.530	45.109	28.580

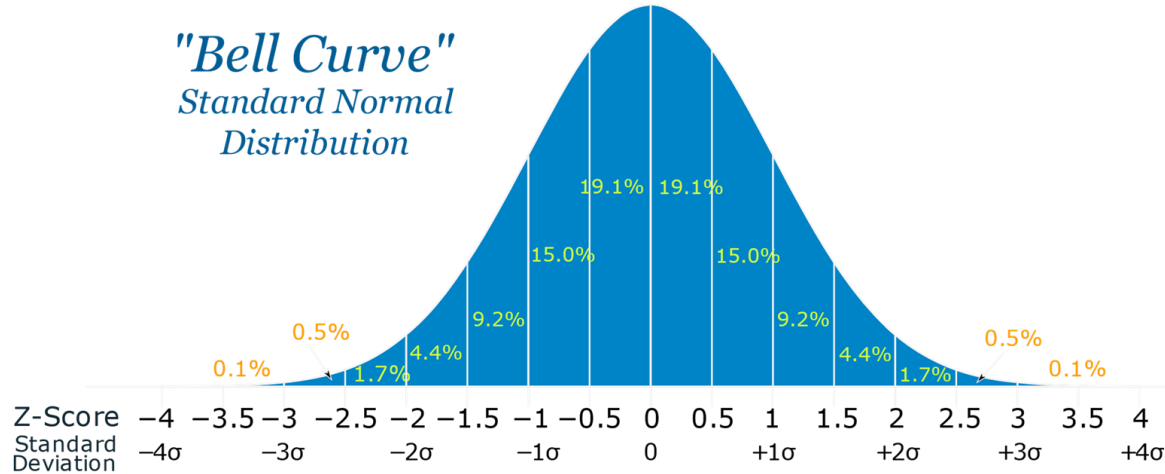
- Time to create numpy array is **gradually increasing**
- Time for **stat collection is negligible**
- Why don't the run times improve as much as expected?

Insertion Sort

ANOTHER WAY TO APPROACH THE ISSUE

- ❖ current Java algorithm will work for normally distributed data (proof of concept)
 - will not work for other types of distributions
 - need to create a more universal approach
 - minimize the computations needed to assign number to bucket
 - minimize the time it takes to collect statistics
 - need to figure out a way to use correct formula to sort given data based on distribution without wasting too much time

INTEGRATION TECHNIQUE



Key Features

- Distribute numbers evenly between buckets
- Use area under the curve to determine to determine bucket index
- Low resources for sorting each bucket

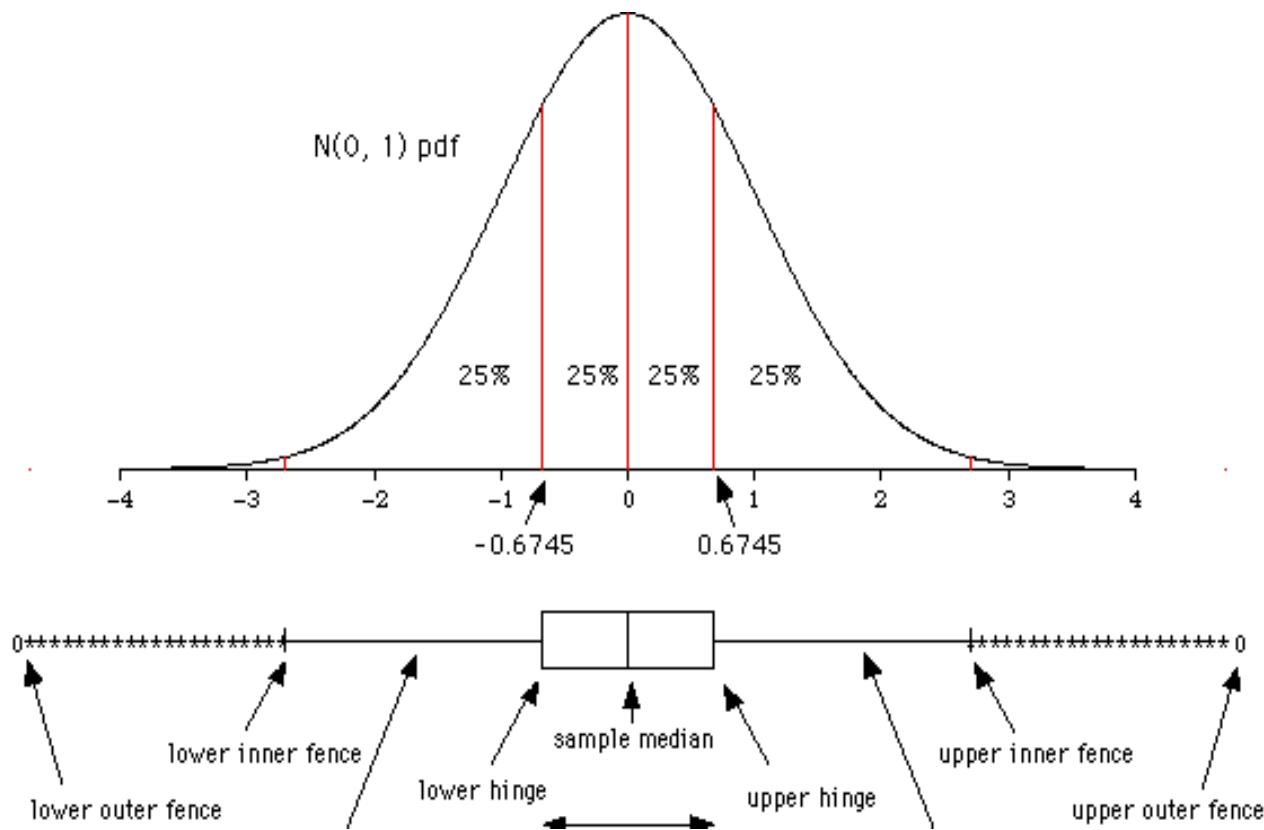
Formula for bucket index

$$\frac{\frac{1}{2\pi\sigma} \int_{\text{min.value}}^x e^{\frac{-1}{2}(\frac{x-\mu}{\sigma})^2} dx}{1000/n} + 1$$

LIMITATIONS OF TECHNIQUE

- Can be efficient
 - other types of distributions rely on degrees of freedom and slopes of probability
 - computation time may increase
 - if the degrees of freedom and parameters are given, then this could work
 - there are ways to estimate these parameters using least squares and other methods = too much time
- Can implement a new technique using box and whisker plots - suggested by Mr. Writer

BOX-AND-WHISKER PLOT TECHNIQUE



- instead of quartiles
 - quintiles
 - deciles
- can have each section cover same area (0.01) and then proceed with algorithm

SUMMARY

- ❖ Java code = 50% more efficient than Parallel Sort, Merge Sort, Bucket Sort
- ❖ Python code = 50% less efficient than Quick Sort (default Python sorting algorithm)
 - NumPy arrays did help solve some problems
 - using Quick Sort = inefficient
- ❖ To make Java code more universal, and to reduce the number of computations, can implement
 - integration technique
 - box-and-whisker plot technique

FUTURE GOALS

- ❖ Write code for Integration Technique in Eclipse
 - can write it in Mathematica as well
- ❖ Write code for Box-and-Whisker Plot Technique in Mathematica
- ❖ Collect data for both techniques
 - compare them with Parallel Sort, Merge Sort, Bucket Sort
 - compare them with each other
- ❖ Apply multithreading to the bucketing aspect to reduce run time

REFERENCES

1. Wang, L., et al. (2012). G-Hadoop: MapReduce across distributed data centers for data-intensive computing. *Future Generation Computer Systems*. doi:10.1016/j.future.2012.09.00
2. Monaknov, A. (2016). Composable multi-threading for python libraries.
3. Goel, N., Laxmi, V., Saxena, A. (2015). Handling multithreading approach using java. *International Journal of Computer Science Trends and Technology*. 3.2.
4. Hai, H., Guang-hui, J., & Xiao-tian, Z. (2013). A fast numerical approach for Whipple shield ballistic limit analysis. *Acta Astronautica*. 93. 112-120. Retrieved from <http://dx.doi.org/10.1016/j.actaastro.2013.06.014>
5. Dehne, F., & Zaboli, H. (2016). Parallel sorting for GPUs. *Emergent Computation*. 293 – 302. doi:10.1007/978-3-319-46376-6_12