PROGRESS REPORT 2 – 12/14/2017

Why are sorting algorithms important?

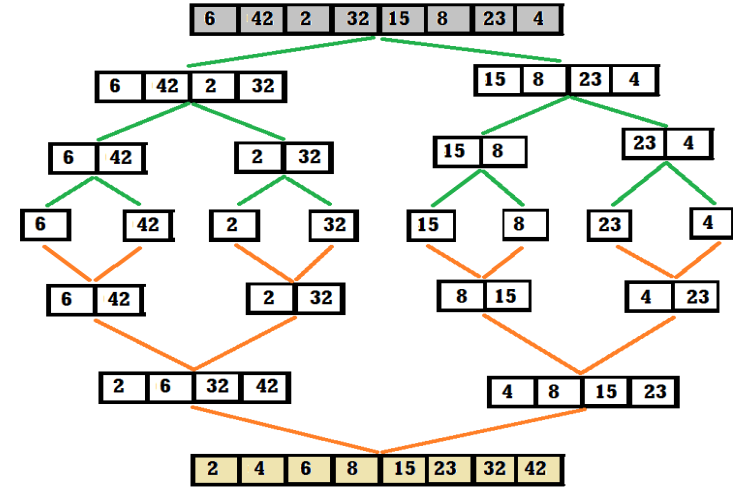
- 25-50% of what computers do require sorting algorithms for
  - Efficiency
  - Speed
  - Processing data
    - Search engines (sorting through all websites to find relevant ones)
    - Graphical processes (layering is necessary)
    - Government organizations, financial institutions, commercial enterprises (Transactions, accounts, processing data)
    - Event driven simulations

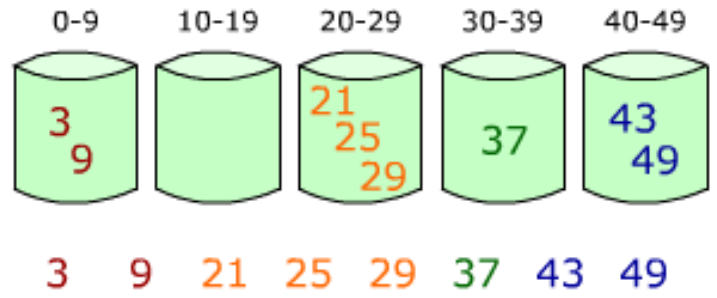
What is Big Data?

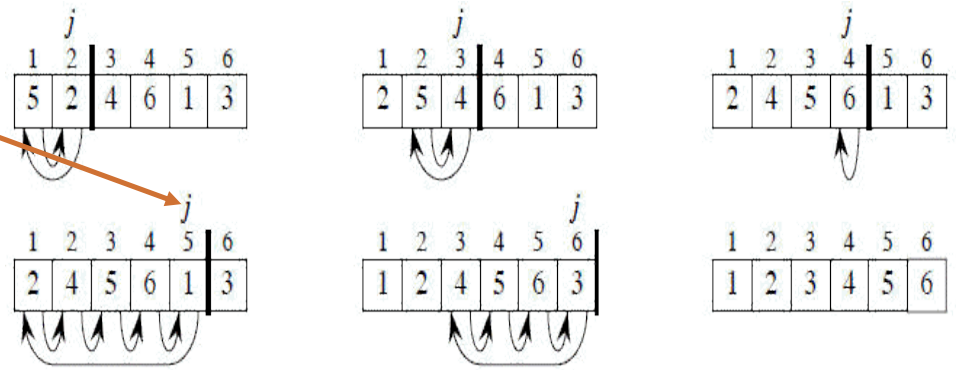
- Big data
  - Collected by government, NASA, research institutions
  - data sets so large and complex that IT systems can not handle them- difficult to process data
  - Volume, velocity, variety, and veracity of data exceed storage or computing capacity
  - challenge of indexing, searching, transferring, and so on all increase exponentially

SORTING BIG DATA

- IMAGINE YOU HAVE **123,456,789** DATA ELEMENTS YOU NEED TO SORT
  - YOU WILL USE SORTING ALGORITHMS (HOPEFULLY)
  - MERGE SORT = 998952457 OPERATIONS
  - BUCKET SORT = 1.52 X 10^16 OPERATIONS
  - INSERTION SORT = 1.52 X 10^16 OPERATIONS
  - *COMPANIES HAVE MANY SERVERS AND SUPERCOMPUTERS SO THEY CAN PERFORM THIS IN MINUTES USING THE SAME ALGORITHMS*







Why is it difficult?

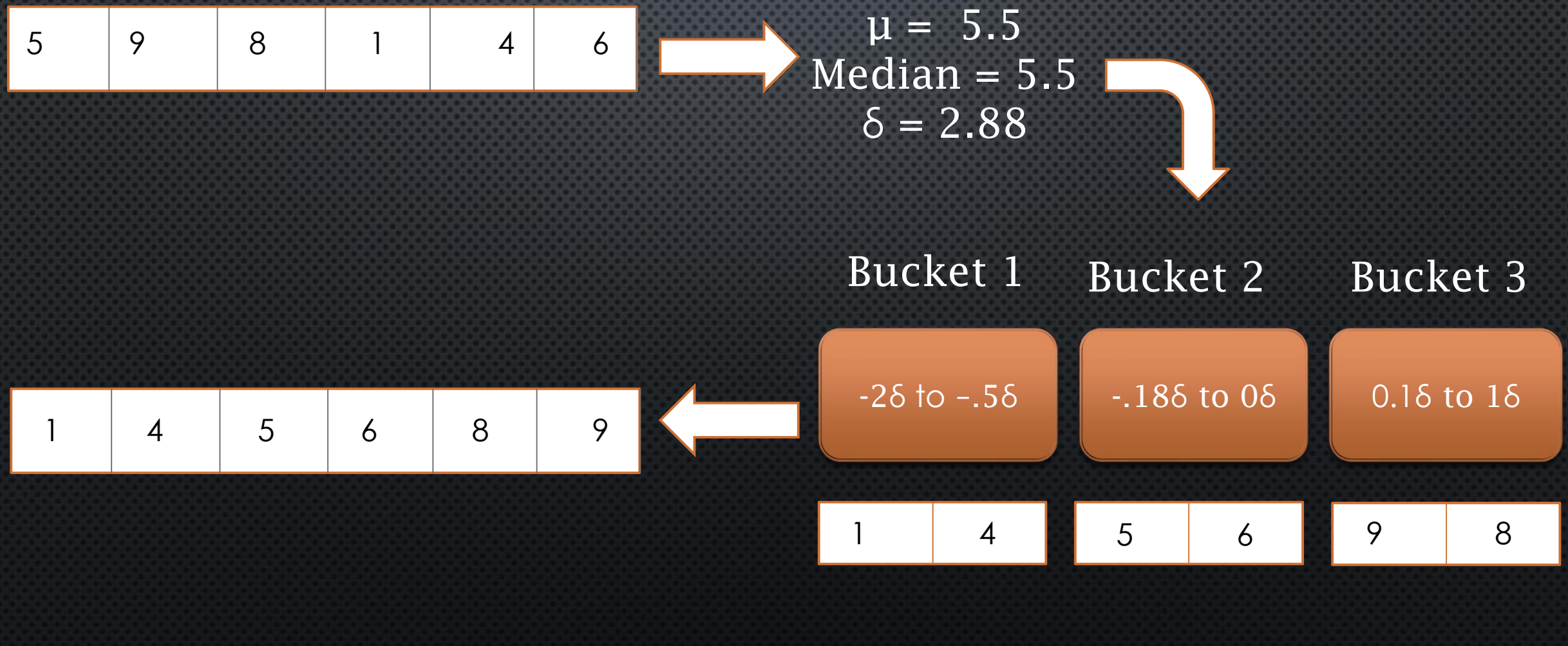
- Too many passes (recursive iterations)
- Each pass allocates its own heap in the memory (each pass has its own space)
- Merge sort – many recursions
- Bucket sort – memory is wasted, distributed data = uneven distribution amongst buckets (although this is a smart idea)
- Insertion sort – solely based on comparing each element to every other one

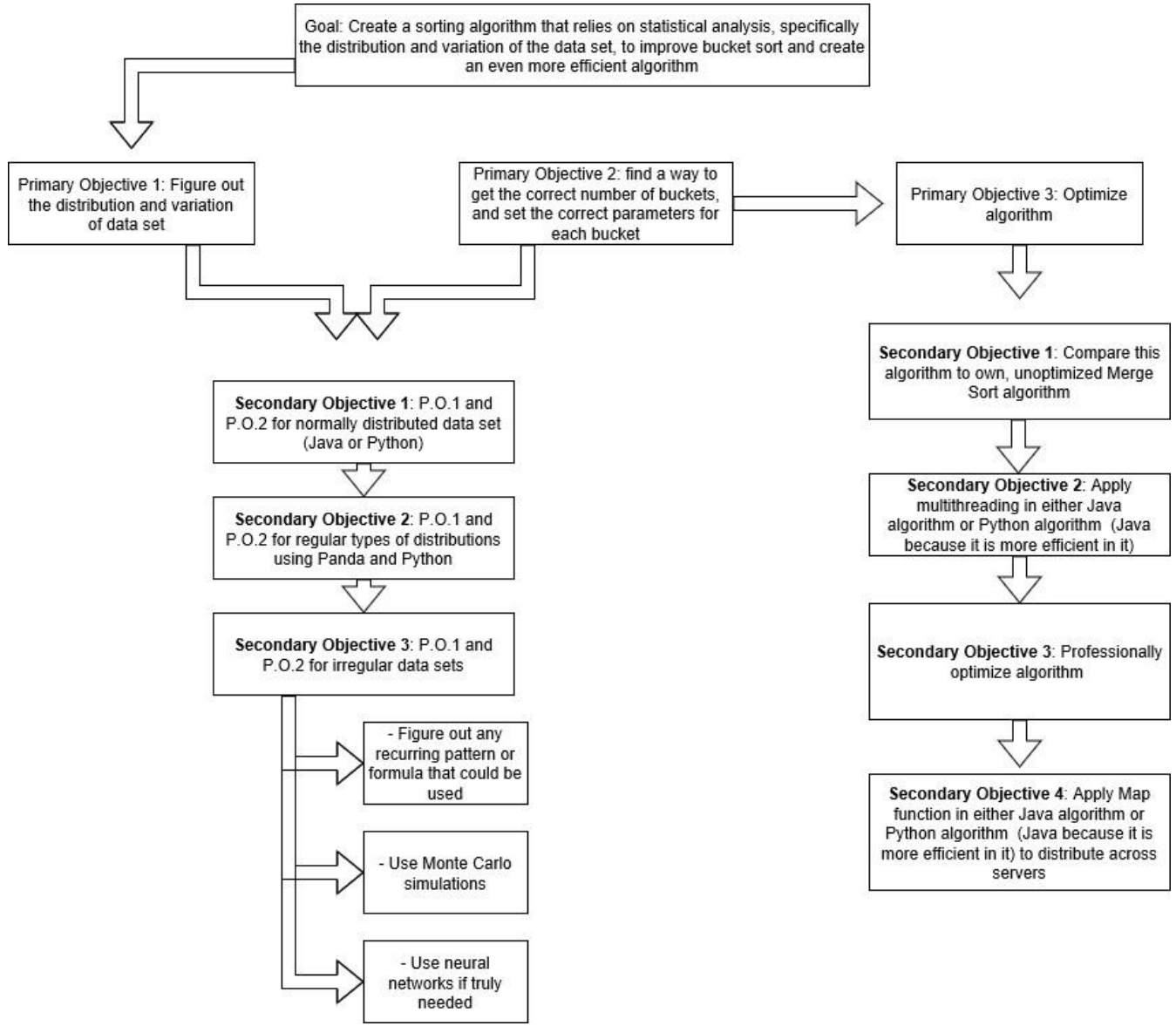
Too much RAM required, too many passes, too much time

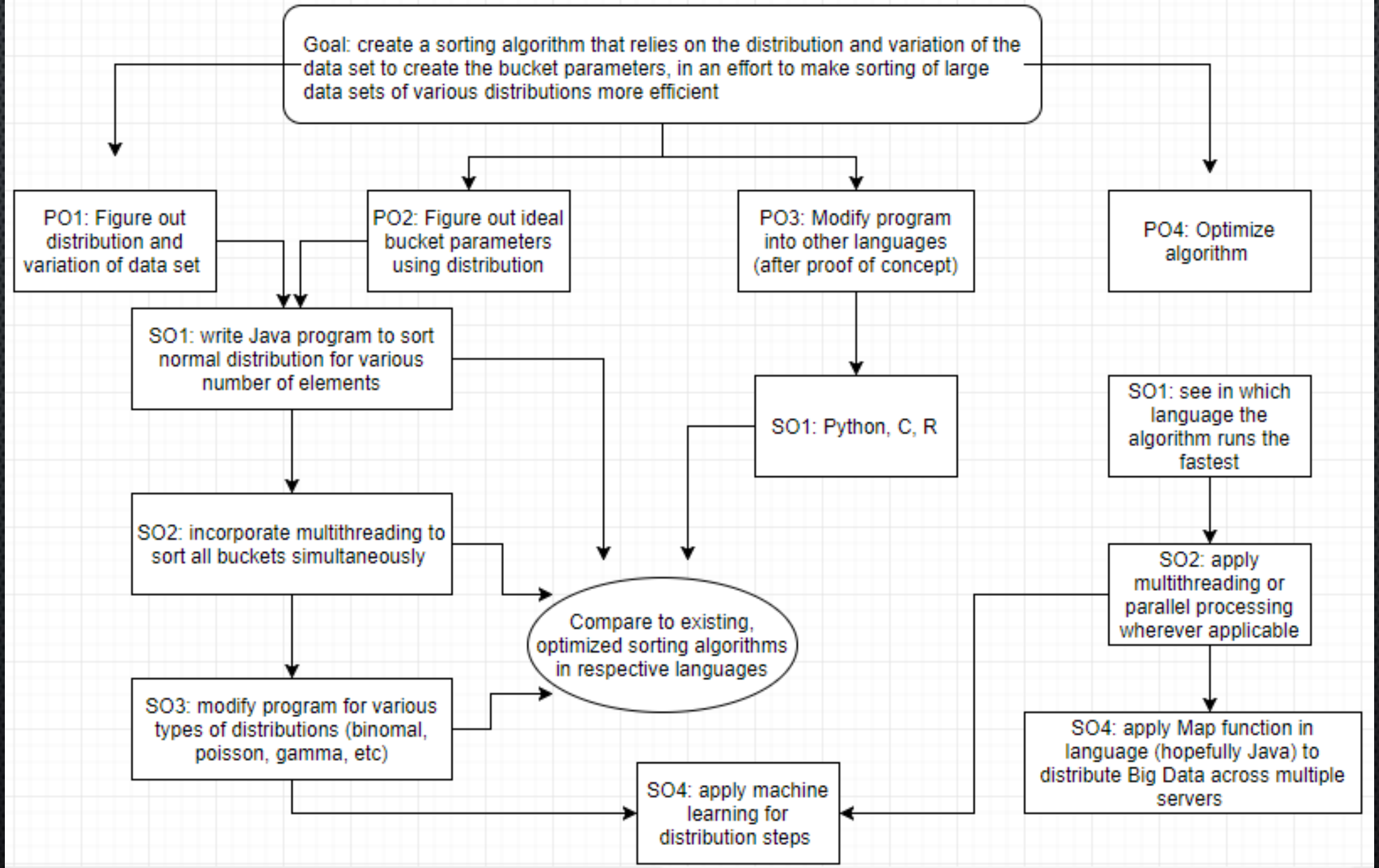
A BETTER WAY TO SORT

- 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




Goal: Create a sorting algorithm that relies on statistical analysis, specifically the distribution and variation of the data set, to improve bucket sort and create an even more efficient algorithm
Primary Objective 1: Figure out the distribution and variation of data set
Primary Objective 2: find a way to get the correct number of buckets, and set the correct parameters for each bucket
Primary Objective 3: Optimize algorithm
Secondary Objective 1: P.O.1 and P.O.2 for normally distributed data set (Java or Python)
Secondary Objective 2: P.O.1 and P.O.2 for regular types of distributions using Panda and Python
Secondary Objective 3: P.O.1 and P.O.2 for irregular data sets
- Figure out any recurring pattern or formula that could be used
- Use Monte Carlo simulations
- Use neural networks if truly needed
Secondary Objective 1: Compare this algorithm to own, unoptimized Merge Sort algorithm
Secondary Objective 2: Apply multithreading in either Java algorithm or Python algorithm (Java because it is more efficient in it)
Secondary Objective 3: Professionally optimize algorithm
Secondary Objective 4: Apply Map function in either Java algorithm or Python algorithm (Java because it is more efficient in it) to distribute across servers


Goal: create a sorting algorithm that relies on the distribution and variation of the data set to create the bucket parameters, in an effort to make sorting of large data sets of various distributions more efficient
PO1: Figure out distribution and variation of data set
PO2: Figure out ideal bucket parameters using distribution
PO3: Modify program into other languages (after proof of concept)
PO4: Optimize algorithm
SO1: write Java program to sort normal distribution for various number of elements
SO2: incorporate multithreading to sort all buckets simultaneously
SO3: modify program for various types of distributions (binomal, poisson, gamma, etc)
SO4: apply machine learning for distribution steps
SO1: Python, C, R
SO1: see in which language the algorithm runs the fastest
SO2: apply multithreading or parallel processing wherever applicable
SO4: apply Map function in language (hopefully Java) to distribute Big Data across multiple servers
Compare to existing, optimized sorting algorithms in respective languages

Progress upto Progress Report 1

- Created backbone of algorithm in Java
  - Technical errors in preliminary code took time to figure out
- Collected data when compared to Parallel Sort/Merge Sort
- Transferring code into Python and learning Panda
- Started communicating with Ms. Kumar

Progress upto now

- Finished programming normal distribution code in python
- Encountered many problems while trying to reduce run time
  - Emailed Ms. Kumar about these issues
  - Tried to figure out solutions using numpy arrays and various numpy methods
- Changed plan due to net progress and time
- Learning how to multithread in java
  - Compared current algorithm to merge sort, bucket sort

PYTHON PROGRAM

```
1  import math
2  import random
3  import datetime
4  class Statsort:
5      #program to compare python embedded sort and new sort being created
6      #main - creating integer normal distribution list of varying lengths, print out times
7      def main():
8          upperlimit=10000000
9          intBucketf=[]
10         for i in range (upperlimit):
11             intBucketf.append((int)(random.uniform(0,upperlimit)))
12         Statsort.sorterSimple(intBucketf)
13         Statsort.sorterBucket(intBucketf,upperlimit)
14     def sorterSimple(intBucket):
15         print (datetime.datetime.now())
16         intBucket.sort() #sort list - already optimized in python
17         print (datetime.datetime.now())
18         #sorting the integer list intBucket using the embedded sort in python
```

- Creating random normal distribution data set
- Sorting using built-in python sort (quicksort)

PYTHON PROGRAM

```
def sorterBucket(intBucket,upperlimit):
    print (datetime.datetime.now())
    buckets=(int)(round(upperlimit/1000)) #creating the number of buckets by dividing list length (elements) by 1000 - elements in each bucket
    intbucketi=[[1000]]*buckets #creating a 2D list in which number of rows is buckets and columns is 1000
    print (datetime.datetime.now())
    Min=intBucket[0] #setting both min and max for first element in list
    Max=intBucket[0]
    sum=0
    for i in range (upperlimit):
        Min=min(Min,intBucket[i]) #comparing each value to get min and max per loop
        Max=max(Max,intBucket[i])
        sum = sum+intBucket[i] #summing all elements
    Mean=round(sum/upperlimit)
    Range=Max-Min
    print (datetime.datetime.now())
    lowerRange=Mean-Min #half of the range of the data set - used to create buckets for each range
    upperRange=Max-Mean
```

- Creating 2D list in which elements are going to be stored

- Obtaining Min, Max, Sum through use of FOR LOOP #1

PYTHON PROGRAM

```
lowerRange=Mean-Min #half of the range of the data set - used to create buckets for each range
upperRange=Max-Mean
lowerGroups=round(lowerRange/Range) #number of buckets for lowerRange
lowerGroups=max(1,lowerGroups) #if the number of elements 1000 or less than 1000, number of groups will be 1 instead of 0
upperGroups=buckets-lowerGroups #number of buckets for upperRange
print (datetime.datetime.now())
for x in range(upperlimit):
    value=intBucket[x] #calling value at index x of list
    if Mean<value: #determining which range it will fall into
        bucketSelect=(int)(math.ceil(lowerGroups*(value-Min)/lowerRange)) #seeing where in the lowerRange does this element lie in - distance fro
    else:
        bucketSelect=lowerGroups+(int)(math.ceil(upperGroups*(value-Mean)/upperRange)) #seeing where in the upperRange does element lie - distanc
    bucketSelect=(int)(((value-Min)/Range)*upperlimit/1000) #reinstantiating bucketSelect
print (datetime.datetime.now())
for y in range(buckets):
    intbucket1.sort() #uses python sorting algorithm to sort buckets
print (datetime.datetime.now())
```

- Creating buckets

- Assigning elements to their respective buckets based on distance from mean

DATA COLLECTED (IT DOES NOT WORK)

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
10000	13.76	13.82	0.06	24.13	24.25	0.12
100000	38.08	38.26	0.18	10.49	11.30	0.81
1000000	4.98	8.52	3.54	42.36	48.70	6.34
10000000	8.01	48.71	40.69	47.59	104.15	56.57

DATA CONTINUED – WHY IS IT SLOWER THAN JAVA?

	Parallel Sort (Optimized)				New Sort				Percent Faster (%)*
Elements	Trial 1 (s)	Trial 2 (s)	Trial 3 (s)	Average (s)	Trial 1 (s)	Trial 2 (s)	Trial 3 (s)	Average (s)	
1000	0.31	0.20	0.27	0.26	0.01	0.02	0.02	0.01	95.40
10000	0.31	0.36	0.37	0.35	0.02	0.01	0.02	0.01	96.04
100000	0.24	0.25	0.28	0.26	0.05	0.08	0.07	0.07	74.03
1000000	0.62	0.68	0.51	0.61	0.16	0.16	0.13	0.15	74.82
10000000	0.48	1.25	1.24	0.99	0.26	0.54	0.47	0.42	57.32
100000000	5.71	5.91	5.82	5.81	3.31	3.06	3.40	3.26	44.01
10000000000	HEAP ERROR								

- HERE, IN JAVA, PROGRAM RUNS SIGNIFICANTLY FASTER THAN PARALLEL SORT, MERGE SORT, AND BUCKET SORT
- WHY IS THE PYTHON SIGNIFICANTLY SLOWER AND INEFFICIENT

FURTHER INVESTIGATION

Trials	Time for Stat Collection (s)	Time for Bucketing (s)	Total Time (s)
1000	0.018	0.000	0.018
10000	0.030	0.052	0.082
100000	0.319	0.441	0.760
1000000	2.362	3.506	5.945
100000000	24.26	46.93	79.01

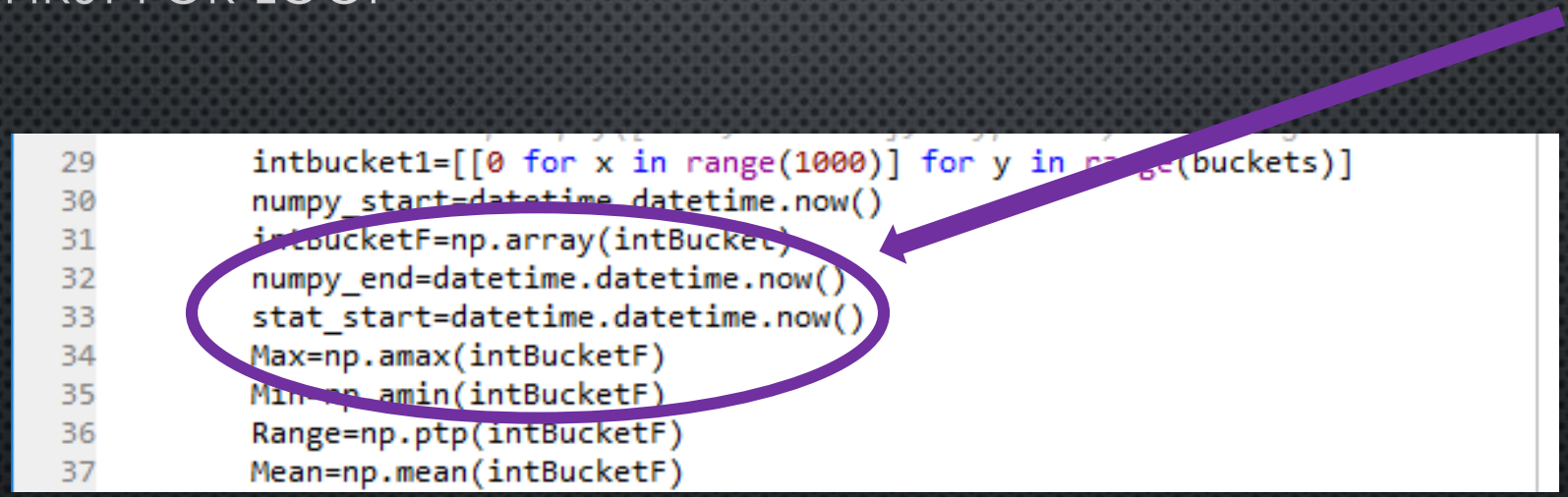
- Time for collecting MIN, MAX, SUM, MEAN increases significantly from trials 4 to 5

- Time for assigning elements to respective buckets increases significantly during same trials

These are the two places where **FOR LOOPS** are incorporated; other elements in the program are simple, one-step functions

POSSIBLE SOLUTIONS???

- NUMPY ARRAYS
  - HAVE EMBEDDED MIN, MAX, RANGE, MEAN FUNCTIONS WHICH CAN ELIMINATE THE FIRST FOR LOOP



MORE DATA

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
10000	4.13	4.14	0.02	48.44	48.58	0.14
100000	25.69	25.78	0.10	13.70	14.93	1.23
1000000	15.06	18.03	2.98	0.59	15.80	16.40

- Run time has significantly decreased from before so the FOR LOOP was indeed affecting run time
- Can time still be cut using numpy array for the 2D list as well?

MORE DATA

Python - Sort with Complete NumPy (Sort, Arrays)						
Elements	Trials	Time for NumPy Array (s)	Time for Stat Collection (s)	Time for Bucketing (s)	Total Time (s)	Sort
1000	AVG	0	0.000	0.023	0.023	0.000
10000	AVG	0	0.000	0.157	0.162	0.005
100000	AVG	0.031	0.000	1.661	1.926	0.265
1000000	AVG	0.267	0.013	16.530	45.109	28.580

- Time to create numpy array is **gradually increasing** – **may cause problems when data set increases** by factor of 100
- Time for **stat collection is negligible** – suggests that numpy array **does make it more efficient** in this aspect
- Time for bucketing with using numpy array
- Why don't the run times improve as much as expected?
  - **QUICKSORT** IS BEING USED TO SORT BUCKETS – **EMBEDDED PYTHON PROGRAM**

IS PYTHON THE RIGHT LANGUAGE?

```
48          for x in range(upperlimit):
49              value=intBucketF[x] #calling value at index of list
50              if Mean<value: #determining which range it will fall into
51                  bucketSelect=(int)(math.ceil(bucketSelectLower*(value-Min))) #seeing wher
52              else:
53                  bucketSelect=lowerGroups+(int)(math.ceil(bucketSelectUpper*(value-Mean)))
54              bucketSelect=(int)(((value-Min)/Range)*upperlimit/1000) #reinstantiating buck
55          bucketing_end=datetime.datetime.now()
56          for y in range(buckets):
57              intbucket1.sort(intbucket1[y])
58          print(np.matrix(intbucket1))
```

- CHECKED TO SEE IF PROGRAM WAS ACTUALLY WORKING THE WAY IT WAS
  - TRIED MULTIPLE WAYS TO PRINT OUT THE 2D LIST AND THE FINAL SORTED ARRAY BUT UNSUCCESSFUL
    - MAYBE DUE TO FAULTY PROGRAMMING
    - DUE TO "BEGINNER'S LEVEL" IN THIS LANGUAGE

DOES THIS PROGRAM EVEN WORK?

- IT DID WORK MAGNIFICENTLY IN JAVA (ADVANCED LEVEL PROGRAMMING SKILL IN LANGUAGE)
- DATA COLLECTED IN JAVA PROVES THAT IT DOES WORK
- PROOF OF CONCEPT IS ESSENTIAL TO MAKE SURE EFFORTS ARE NOT WASTED
- WILL JUST CONTINUE WITH COMPLETING FULL PROGRAM IN JAVA
  - ***PROOF OF CONCEPT***

Next steps

- Apply multithreading – sort buckets all at the same time
- Modify algorithm to sort various types of distributions
  - Binomial
  - Poisson
  - Weibull
  - Log normal
  - Student's t
  - Hypergeometric
    - merely the **FORMULA** needs to change
- Create a tool that will find distribution – if needed