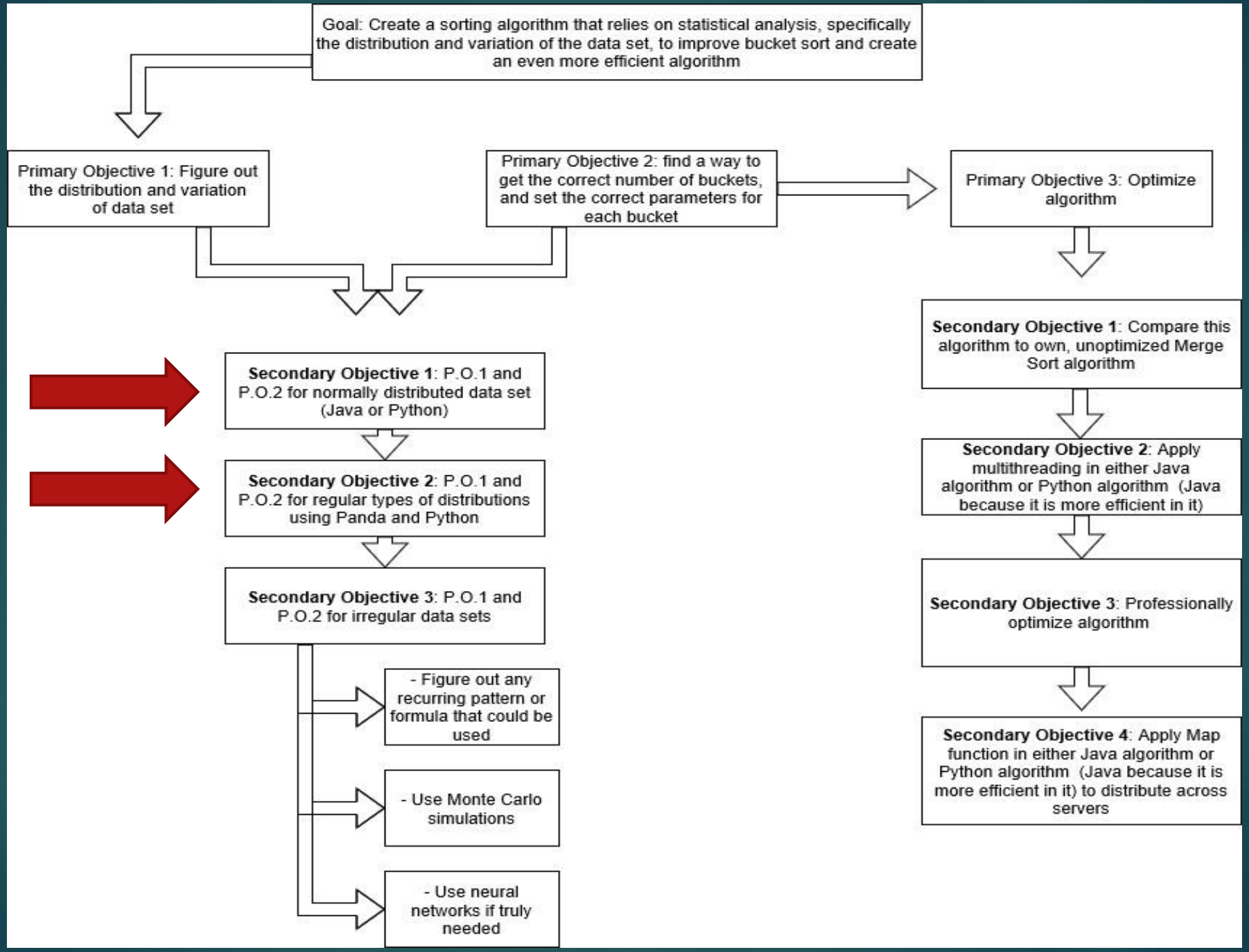
Background

Goal – Program a faster sorting algorithm using statistical analysis

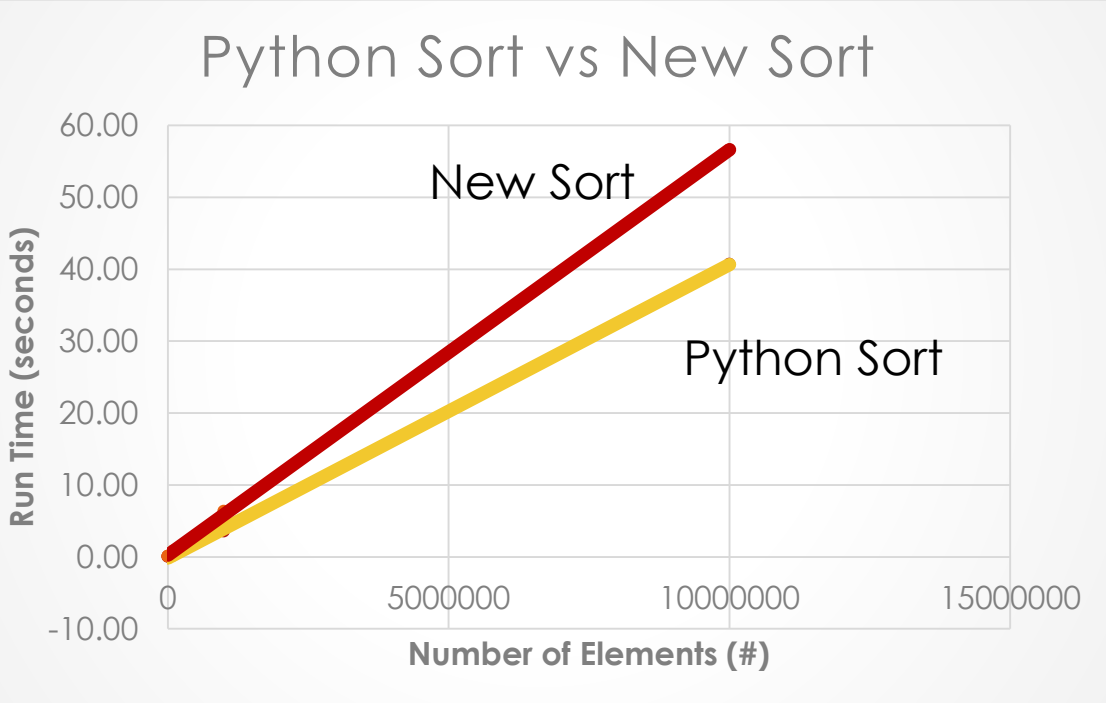
Progress –

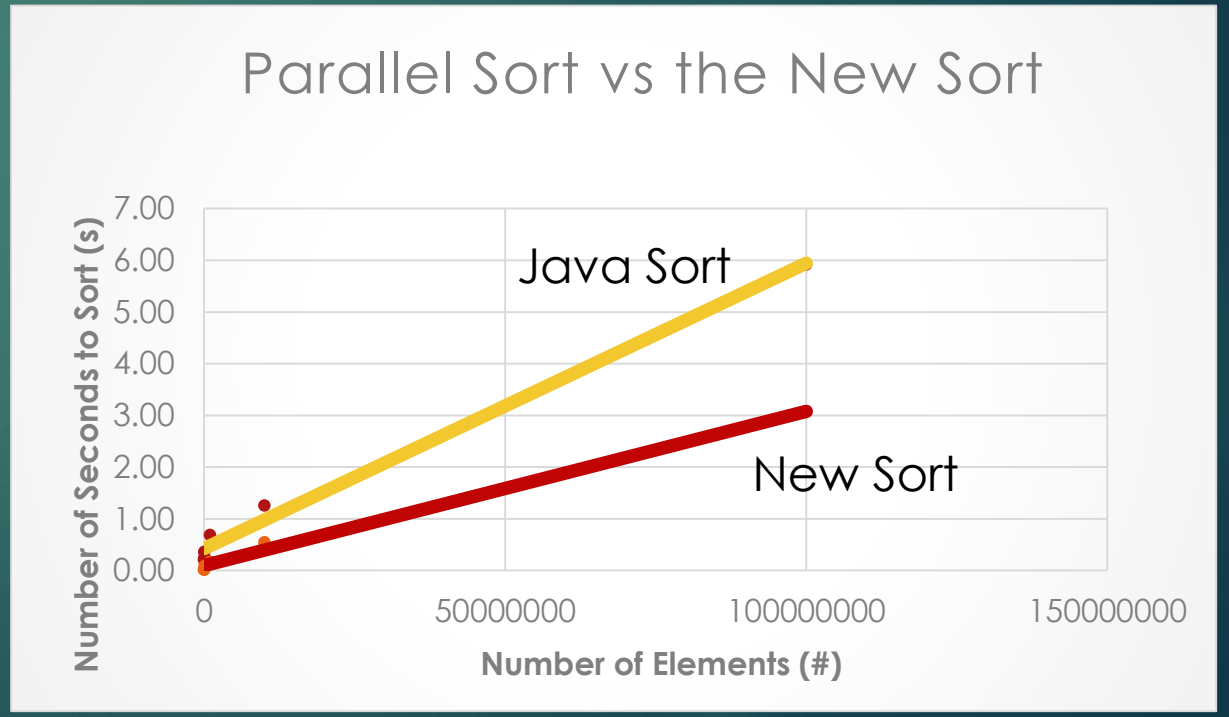
- Written code in **Java** – collected preliminary data (next slide)
  - Compared to **Merge Sort**, **Bucket Sort**, **Parallel Sort**
- Written code in **Python** – collected preliminary data (next slide)
  - Compared to **built-in Python sort**


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

Elements	Python Sort – Total Time (s)	New Sort – Total Time (s)
1000	0.03	0.06
10000	0.06	0.12
100000	0.18	0.81
1000000	3.54	6.34
10000000	40.69	56.57

Elements	Java Sort – Total Time (s)	New Sort – Total Time (s)
1000	0.26	0.01
10000	0.35	0.01
100000	0.26	0.07
1000000	0.61	0.15
10000000	0.99	0.42
100000000	5.81	3.26





Issue

Even though the **same code** is being used in Python and Java, **run times** for both are completely **different** (data on the previous slide – circled in red)

- Sort took a shorter time on Java than Python – **WHY?**

Took a look at the what was taking a long time as program ran in Python

- The two **FOR LOOPS** (the most crucial part) takes a significantly long time while in Java they do not affect the run time significantly

The first FOR LOOPS (circled in green on slide below) – figure out the **max and mins** of data set – necessary to create **end parameters**

Python

```
for i in range (upperlimit):
    Min=min(Min,intBucket[i]) #comparing each value to get min and max
    Max=max(Max,intBucket[i])
    sum = sum+intBucket[i] #summing all elements
```

Java

```
for (int i = 0; i < upperLimit; i++){
    iMin = Math.min(iMin, intBucket[i]);
    iMax = Math.max(iMax, intBucket[i]);
    iSum+=intBucket[i];
}
```

The second FOR LOOPS (circled in purple on slide below) – determine **which bucket** each element belongs to

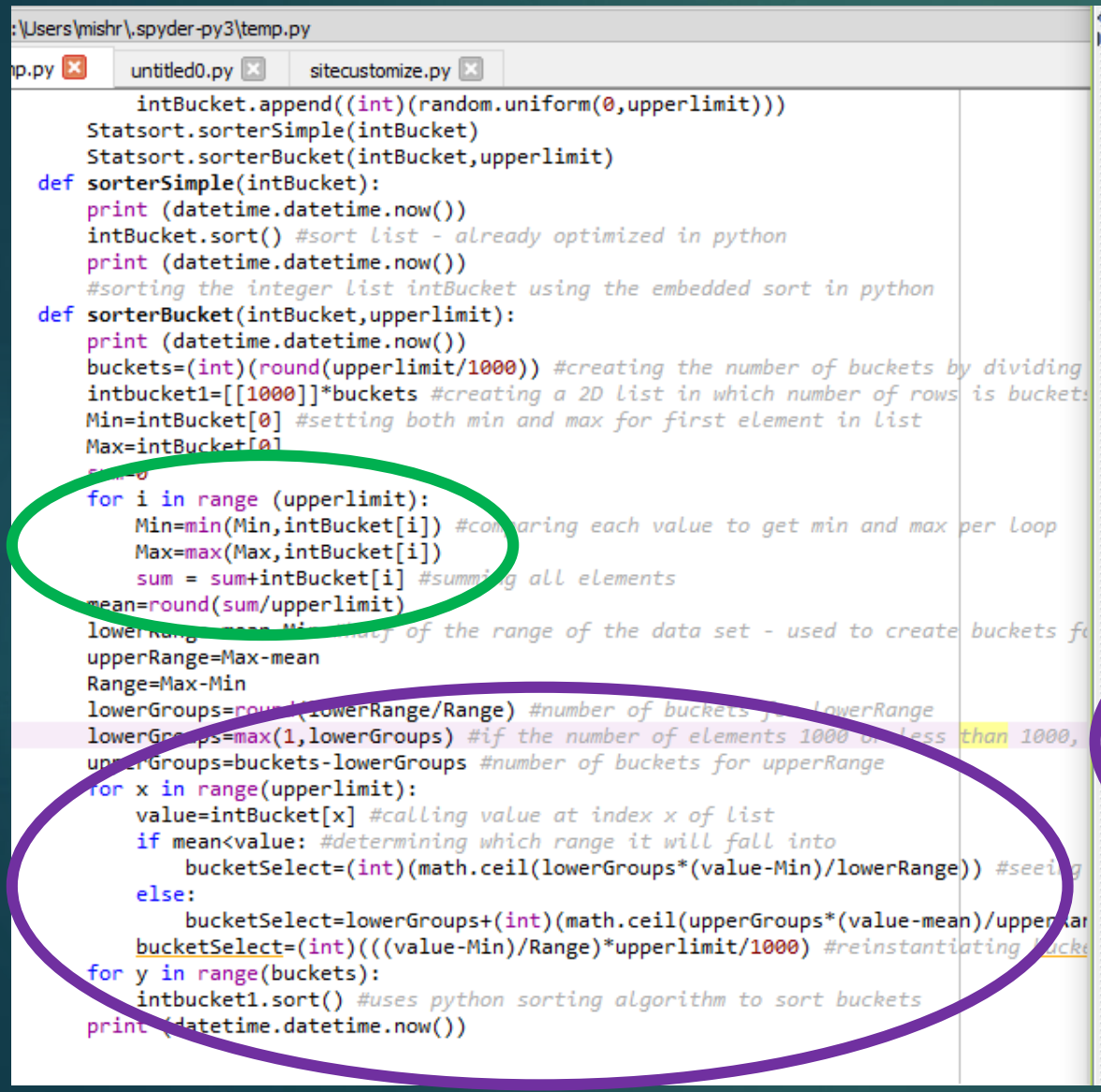
Python

```
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
    else:
        bucketSelect=lowerGroups+(int)(math.ceil(upperGroups*(value-mean)/upperRar
    bucketSelect=(int)(((value-Min)/Range)*upperlimit/1000) #reinstantiating bucke
for y in range(buckets):
    intbucket1.sort() #uses python sorting algorithm to sort buckets
print (datetime.datetime.now())
```

Java

```
for (int i = 0; i < upperLimit; i++){
    int iValue = intBucket[i];
    if (iMean > iValue) {
        bucketSelect = (int)Math.ceil(lowerGroups * (iValue - iMin)/lowerRange);
    }
    else {
        bucketSelect = lowerGroups + (int)Math.ceil(upperGroups * (iValue - iMean)/u
    }
    bucketSelect = (int)((iValue - iMin)/iRange)*upperLimit/1000;
}
for (int i = 0; i < buckets; i++){
Arrays.parallelSort(intBucket1[i]);
}
debug_msg("End sorterBucket " + LocalDateTime.now());
```

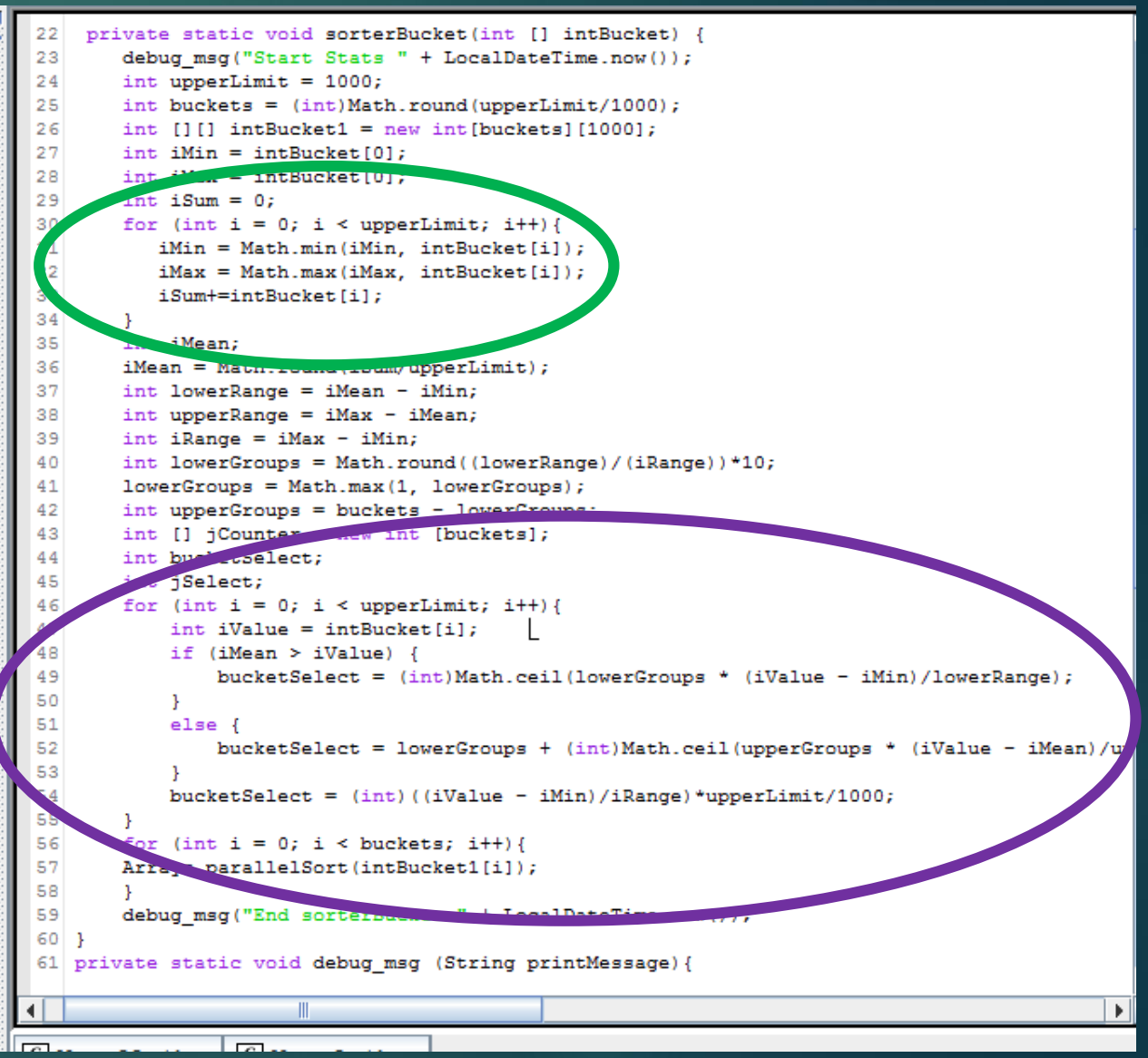
Python Code on Spyder



```
:\Users\mishr\.spyder-py3\temp.py
np.py | untitled0.py | sitecustomize.py

        intBucket.append((int)(random.uniform(0,upperlimit)))
    Statsort.sorterSimple(intBucket)
    Statsort.sorterBucket(intBucket,upperlimit)
def sorterSimple(intBucket):
    print (datetime.datetime.now())
    intBucket.sort() #sort list - already optimized in python
    print (datetime.datetime.now())
    #sorting the integer list intBucket using the embedded sort in python
def sorterBucket(intBucket,upperlimit):
    print (datetime.datetime.now())
    buckets=(int)(round(upperlimit/1000)) #creating the number of buckets by dividing
    intbucket1=[[1000]]*buckets #creating a 2D list in which number of rows is buckets
    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)
    lowerRange=mean-Min #half of the range of the data set - used to create buckets fo
    upperRange=Max-mean
    Range=Max-Min
    lowerGroups=round(lowerRange/Range) #number of buckets for lowerRange
    lowerGroups=max(1,lowerGroups) #if the number of elements 1000 or less than 1000,
    upperGroups=buckets-lowerGroups #number of buckets for upperRange
    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
        else:
            bucketSelect=lowerGroups+(int)(math.ceil(upperGroups*(value-mean)/upperRar
        bucketSelect=(int)(((value-Min)/Range)*upperlimit/1000) #reinstantiating bucke
    for y in range(buckets):
        intbucket1.sort() #uses python sorting algorithm to sort buckets
    print (datetime.datetime.now())
```

Java Code on JGrasp



```
22  private static void sorterBucket(int [] intBucket) {
23      debug_msg("Start Stats " + LocalDateTime.now());
24      int upperLimit = 1000;
25      int buckets = (int)Math.round(upperLimit/1000);
26      int [][] intBucket1 = new int[buckets][1000];
27      int iMin = intBucket[0];
28      int iMax = intBucket[0];
29      int iSum = 0;
30      for (int i = 0; i < upperLimit; i++){
31          iMin = Math.min(iMin, intBucket[i]);
32          iMax = Math.max(iMax, intBucket[i]);
33          iSum+=intBucket[i];
34      }
35      int iMean;
36      iMean = Math.round(iSum/upperLimit);
37      int lowerRange = iMean - iMin;
38      int upperRange = iMax - iMean;
39      int iRange = iMax - iMin;
40      int lowerGroups = Math.round((lowerRange)/(iRange))*10;
41      lowerGroups = Math.max(1, lowerGroups);
42      int upperGroups = buckets - lowerGroups;
43      int [] jCounter = new int [buckets];
44      int bucketSelect;
45      int jSelect;
46      for (int i = 0; i < upperLimit; i++){
47          int iValue = intBucket[i];
48          if (iMean > iValue) {
49              bucketSelect = (int)Math.ceil(lowerGroups * (iValue - iMin)/lowerRange);
50          }
51          else {
52              bucketSelect = lowerGroups + (int)Math.ceil(upperGroups * (iValue - iMean)/u
53          }
54          bucketSelect = (int)((iValue - iMin)/iRange)*upperLimit/1000;
55      }
56      for (int i = 0; i < buckets; i++){
57      Arrays.parallelSort(intBucket1[i]);
58      }
59      debug_msg("End sorterBucket " + LocalDateTime.now());
60  }
61  private static void debug_msg (String printMessage){
```

How I plan to solve these problems in Python

- I will use a Pandas Dataframe
  - store the values of unsorted list – use built in MIN and MAX functions to eliminate the first for loop – reduce run time by approximately half
- Learn more Pandas (or NumPy depending on which one is best suited for this project)
  - figure out a way to either eliminate second for loop or reduce the number of functions in the for loop

QUESTIONS

1) Why is it that the for loops are taking a significantly longer time in Python than in Java despite the code being similar?
2) Would using a DataFrame reduce my runtime in Python if I implement the built-in functions to eliminate at least the first for loop?
3) Would using NumPy be more beneficial than using Pandas?
4) Should I also code my program in different languages (C and R along with Java and Python) so that I can compare the run times and see what exactly is causing my program run differently in Python?