Leveraging Mobile Devices for Distributed Computing

Submitted September 2011, in partial fulfilment of the conditions of the award of the degree Information Technology

Michael D. Jarrett

School of Computer Science University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature ______________________

Date _____/_____/_____

I hereby declare that I have all necessary rights and consents to publicly distribute this dissertation via the University of Nottingham's e-dissertation archive.

Public access to this dissertation is restricted until: 07 / September / 2011

DO NOT COPY

Abstract

As mobile devices become faster and more powerful, equalling the processing power of desktop computers of just 5 years ago, we must consider using them as a distributed computing network to solve complex problems. Can these devices do complex computations in a reasonable time and would it be beneficial to do so? Inspired by the BOINC community I set out to solve that question by building a client-server application that communicated asynchronously to pass data. Once the data had been received, the client application would perform matrix multiplication. The goal is that results would prove positive enough that we can indeed utilise multiple mobile devices to do complex mathematics. Indeed, the results were positive showing that the mobile devices were between a factor of 3 and 4 slower than PCs. Whilst, these results do not appear to be very good, the algorithm implemented could be optimised and better hardware used could easily drop it to a factor of 2. Running ten thousand mobile devices to obtain the same results as a PC may not seem like a good ratio but if a community is introduced as well as competition within, similar to the BOINC community's rewarding credit for data that was returned then a large community could be built around distributed mobile computing to solve real world issues.

DO NOT COPY

Acknowledgment

I would like to thank Dr Steven Bagley for the guidance he gave me on this project and for being a great teacher and teaching me how to program applications for the Android OS. I also would like to thank Katie Behringer for helping me edit this paper, without her there would be no structure to this paper. I would also like to thank Programmer XR for his well written tutorial on communication using Android and JSON.

DO NOT COPY

Contents

1. Introduction 1

2. Related work 1

  2.1 Mobile Parallel Computing 2

  2.2 BOINCOID 2

3. Methodology 3

  3.1 The Client 3

  3.2 The Server 4

  3.3 Analysis 4

4 Description of the work 5

  4.1 Client Side 5

  4.2 Server Side 6

5. Discussion 7

  5.1 Emulated Android Device 7

  5.2 Physical Android (ZTE Racer) 8

  5.3 Pure Java 10

  5.4 Analysis of Data 11

  5.5 Verifying data 12

  5.6 How Can This Be Applied 13

  5.7 The effects on the phone 14

6. Conclusion 14

Works Cited i

Appendix 1 – Code Listing I

  Appendix 1.1 – Main.java I

  Appendix 1.2 – Packet.php V

  Appendix 1.3 – SQL Schema VII

Appendix 2 – Aggregated Log Files IX

  2.1 – Emulate Android Device IX

  2.2 – ZTE Racer (Physical Android device) results XV

  2.3 – Java on PC XXI

DO NOT COPY

1. Introduction

What is stopping us from using our mobile devices as supercomputers? Maybe not just one mobile device, but could we use even a small percentage of the estimated 300 million Google Android and Apple iOS devices (Tsotsis, 2011) simultaneously to predict climate change or discover new medicines based on calculating different ways proteins can fold and mutate? Why not search for radio signals coming from space? Will this be the way we find E.T.? These are just some of the projects that utilise the Berkeley Open Infrastructure for Network Computing (BOINC) distributed computer grid and the original inspiration for distributed mobile computing.

The goal of my research was to determine if the modern mobile devices that are running the Android operating system could in fact be used to solve any of the previously stated problems. Whilst integrating and communicating with BOINC to actually contribute to their projects would have been excellent, I felt I would not have had a satisfactory amount of time to successfully complete the task. I instead opted to build a prototype demonstrating that packets of data could be received from a remote server, processed on the mobile device and then returned to the server, within a reasonable amount of time. To demonstrate this, I have chosen to perform distributed matrix multiplication since the way that the multiplication is performed, multiplying the items in the first matrix row by the items in the second matrix column and summing their products, lends itself well to be broken apart, sending one row and one column to a device on a distributed computing network. Although the focus was on matrix multiplication, other modules could easily be written and included into the program to extend its functionality. For example, an additional method could search for patterns in two separate sequences of elements or a distributed vector tracing algorithm.

For the project I was able to program a client for the Android platform that would request rows and columns from a server, do the appropriate calculations, and return the calculated data to the server to be stored. Using an emulated Android device and a ZTE Racer I was able to demonstrate that matrix multiplication could be carried out on the Android platform. When the results were compared with a computer than ran virtually the same Java code, we find that whilst the mobile devices are able to perform the calculations, they are a factor or three to four slower than that of a PC.

2. Related work

With the introduction of the iPad and a multitude of Android tablets, on top of the 300+ million smartphones already on the market, one would expect to find the field of distributed mobile computing full of researchers searching for a way to leverage the unspent processing power when the devices just idle whilst charging. However, I have found little data showing that researchers are interested in using these devices as a distributed platform. Using search terms such as 'mobile distributing computing' and 'mobile parallel computing', the only relevant

DO NOT COPY

results I discovered were one paper, ‘Mobile Parallel Computing’, and one project, BOINCOID, that specifically address the idea of using Android devices as a distributed computing network. With as little research as this having been done, the topic is begging for more input as these types of mobile devises continue to grow rapidly in popularity.

2.1 Mobile Parallel Computing

In Mobile Parallel Computing, Doolan, et al. described a message passing platform built using Java to pass packets over bluetooth to the Nokia 6630[1] and Nokia 6680[2] (Doolan, Tabirca, & Yang, 2006). The packets that were sent out were rows and columns of matrices that the phones would perform multiplication and addition on, and then return the product back to the server, which in this case was another mobile phone. Once the results are returned, they are displayed, in a horizontal fashion on the screen to the user. They showed that distributing the work amongst the connected devices and processed in parallel showed substantial improvements of processing the data serially. Doolan’s system was designed to have multiple devices connected to the server synchronously and all processing the data in parallel. My design goal is different in that clients will connect to the server asynchronously, retrieve and process that data and then return the data back to the server to be stored.

They were able to prove that their system was cable of running matrix multiplication in $O(n^2)$, which is successful in showing that these now older devices could be used for matrix multiplication. The clock speeds of the Nokias CPUs are roughly 22% of the current generation of mobile devices. This difference, coupled with the architecture improvements over the last five years since the article was published, provides for a large margin for improvement over what they experienced.

2.2 BOINCOID

The other interesting piece of research that was uncovered was the BOINCOID project, which was started and maintained by three Israeli computer science students and an American software engineer. The original goal of the project was essentially to port the BOINC code to the Android platform. What originally drew me into the website were the statements “we intend the BOINCOID users to switch on the processing at night, when their phone is charging and idle” and “mobile devices are designed much much more energy efficient than home PC's. Since battery life is a major concern, the CPU processing is optimized to require the least amount of energy possible. In grand BOINC proportions this has critical implications. It means the same amount of calculations could take a lot less energy. On one hand this means we aid the environment in wasting less electricity, on the other it means we could deploy high-density farms of these devices supporting a lot more processing power, with the same limited resources.” (FAQ, 2009) BOINCOID claims that they were successful in pulling data from the

[1] OS: Symbian OS v8.0, Series 60 v2.0 UI, CPU: 220 MHz ARM926EJ-S processor (GSM Arena)

[2] OS: Symbian OS v8.0a, Series 60 UI, CPU: 220 MHz ARM926EJ-S processor (GSM Arena)

DO NOT COPY

BOINC servers, asynchronously, processing the data, and returning the complete data back to BOINC.

Unfortunately, when starting this research, downloading the project from Sourceforge, compiling and installing it on an Android device, emulated or physical proved to be futile. The webpage and SourceForge page have not been updated since May 2009. I believe this path is a dead-end, although it may be worth returning to and contributing to this project in the future to reinvigorate it.[3]

3. Methodology

Choosing a server-client paradigm using the producer-consumer pattern for this project seemed like the most logical choice. The rationale for this was that the mobile devices, clients, would connect asynchronously to the server and retrieve a JavaScript Object Notation (JSON) packet4. JSON was chosen over XML for one main reason: simplicity. "JSON is much simpler than XML. JSON has a much smaller grammar and maps more directly onto the data structures used in modern programming languages." (JSON, 2011) With smaller grammar, the amount of data sent over the network, whether wireless, Bluetooth or 3G would be significantly smaller than if using XML. This would be beneficial in two ways. First, if the user decided to run the client when not charging the phone, the amount of time the network device would need to be on would be reduced, thus saving energy and allowing more energy for the CPU to process data. Second, users still must be conscience of the amount of data they consume when using 3G and would not be happy if a program that they are volunteering to run costs them money in data usage overages.

After the client receives the JSON packet, the server can then serve the next JSON packet whilst waiting for the client to return the first one. Multiple clients can connect and retrieve packets, go and process them and return them at a later time. During the testing period I did have three connections to the server; one from the virtual device, one from the ZTE Racer and one from the Java application. The server appeared to have no issues sending and receiving the packets to the clients. Since BOINC servers operate in this fashion, it was a good design decision.

3.1 The Client

The client application was to be designed for the Android platform. Being an Android device owner and more familiar with Java, the decision to use the Android platform instead of Apple's iOS platform — which utilises the Objective C programming language — would allow for a more

[3] As I was editing this paper, I returned to BOINCOID's Sourceforge page and was delighted to discover that the project had indeed been started again by other programmers and is now in active development.

[4] Packet is this paper usually is a lose term for a set of data that is transmitted as one unit. This should not be confused with IP or other transmission packet types.

DO NOT COPY

rapid application development. The decision to build the client application using the Android platform, specifically the Android OS 2.2 (Froyo) API Level 8, was based on the fact that about 51% of the Android market is using Froyo. (Google, 2011) This would allow for wide distribution if the development was successful and deemed ready for public consumption. The devices I used were a virtual Android (2.2) device for development and testing purposes and a ZTE Racer[5] mobile phone, updated to Android 2.2, for non-emulated tests.

3.2 The Server

For the server I decided on a Linux, Apache, MySQL, PHP (LAMP) server[6] which would serve JSON packets on an HTTP POST request from the client. The decision to use a LAMP server and PHP was an easy one considering the cost, documentation[7] and other projects that have done similar work being easily accessible online (Programmer XR, 2011). The project described at Programmer XR site gave me great insight on how to set up a PHP server send JSON packets, as well as how to establish an HTTP connection and process the read JSON packet on the Android platform.

3.3 Analysis

After writing the Android application, the server and successfully getting the client and server to pass JSON packets between one and another, I decided to monitor how long the summation was taking. To accomplish this, I created a variable at the beginning of the for loop that was assigned Java's `System.nanoTime()`. At the end of the for loop, I again made the call to `System.nanoTime()` and subtracted to first call to nanoTime from the second.[8] I then output the result, which is system time independent and returns the amount of time that has passed in nanoseconds to a file. I then modified the code to run as a standalone Java application so that I could compare the mobile device performance versus Java on a standard PC. The Java application, like the Android application, made calls to nanoTime and placed the results in a file to be analysed later on.

To analyse the data, I aggregated it all into a list and wrote a Python program that was just a function called that accepted a list and then utilising Python Statistics Library[9] to compute maximum, minimum and total runtimes, mean, standard deviation and median. I was able to look at how long each piece of data took to process, as well as the above calculated information.

[5] OS: Android v2.1 (Eclair), CPU: 600 MHz ARM 11 GPU: Adreno 200, Chipset: Qualcomm MSM7227 (ZTE Blade)

[6] LAMP server in the project had the following specifications: OS: Ubuntu 11.04, CPU: 2.53GHz Intel i5 M450, RAM: 1024MB

[7] http://php.net/manual/en/index.php - Complete manual online, with user contributed code and comments below each class and method.

[8] http://stackoverflow.com/questions/180158/how-do-i-time-a-methods-execution-in-java

[9] http://code.google.com/p/python-statlib/ - Python Statistics Library

DO NOT COPY

4 Description of the work

The client application running on the Android platform is based on Programmer XR tutorial, "From SQL/PHP to XML/JSON to Android ListView!" modified so that the incoming JSON string to Android does not display on the screen in the ListView widget but instead places each item into an array specific to row and column so that multiplication and summation can be performed.

4.1 Client Side

Functionally, if the application was to run just one iteration of the while loop (row multiplied column and added to zero, it can be explained in the following manner. First, the application attempts to establish an HTTP connection with the server to request a name-value pair. The request returns a JSON element similar to: `[{"_ID" : "25", "ROWS" : "1,2", "COL" : "6,5", "ROW_COR" : "1","COL_COR" : "2", "MATRIX_ID" : "1"}]` as a basic HTTP streamed entity. A string based on the JSON object that was received from the HTTP stream is then attempted to be built by creating a buffer reader object and string builder object. The buffer read object is iterated over calling read line and appending the object to the string builder. Once all information in the HTTP stream has been read in, the stream is then closed and the string builder object is converted to a string and assigned to a variable, in this case **`result,`** and then finally is encoded into JSON notation. These steps are required because there is no built-in Java function that can convert a stream object to a string object.

Once the string has been created from HTTP stream, the program then attempts to create a JSON array from **`result`**. It then selects the first item of the JSON array and pulls out the key values: _ID, ROWS, COL, ROW_COR, COL_COR, and MATRIX_ID. _ID is the primary key for the database from which this set of data came from. The _ID will be passed back to the server once the matrix size has been verified so that the server can remove this set of data from the queue. The ROWS and COL keys are paired with values that are strings which are comma delimited. The program splits these strings up appends them to two separate zero-indexed empty arrays which will later be multiplied and added together to return the desired result. The values paired with keys ROW_COR (row coordinate), COL_COR (column coordinate), and MATRIX_ID are used to associate each row and column calculated value to a specific matrix as well as which position in the matrix the return value is to be placed.

Once all of the data has been processed, the matrix multiplication calculation is finally performed. First, a float variable, **`total`**, is created and assigned with value zero. Then the program iterates down both row and column arrays and multiplies them together and adds the results to the **`total`** variable. Since the row and column length sent to the client has already

DO NOT COPY

been verified as being the same length, the summation that is performed is $\sum_{i=0}^{length} Row(i) * Column(i)$; where i is equal to the index of the row and column and $length$ is the number of elements in the array.

After the client has a result to return to the server it builds a new JSON object containing the calculated result, the matrix id, row and column coordinates. The application than prepares for an HTTP connection that will POST the JSON object to the server to be saved to the remote database. It accomplishes this, much in the same way that it originally received the JSON array that needed to be processed, but for this POST it builds a name value pair to include with the POST so the server make take appropriate action. Once the connection has been established the client makes the HTTP POST and the server saves and notifies that client that the save was successful or not.

When the client accesses the server via an HTTP POST request, the server queries the database and returns a well formed JSON packet. The client then processes the data, as previously described, and does another HTTP POST to return the data to the server which the PHP script then verifies and saves to the database.

When the client starts it begins requesting data to be processed. It will then process the data and request another packet. It will continue doing this until the either the server has no more data packets to be processed or the client is exited. Future functionality that will be added to the program will allow the user to start and stop the request as well as have an option to start the requests as soon as the mobile device is plugged in for charging.

4.2 Server Side

The server application consists of three basic functions: get and display the JSON packet, make a copy of the original data, as well as save computed data and clean up copies. There are three database tables associated with these functions as well. The waiting table contains the rows and columns of data that are waiting to be processed. The copy table contains a copy of the original data that was sent to the Android client. This table is to help minimise loss of data so if a packet is sent out and never returned to the server, the packet can easily be moved from copy back into the waiting queue. And finally, the computed table contains the computed data, the associated matrix id, the row and column coordinates.

When an HTTP POST request is made to the server, the PHP script will query the database table WAITING and return the first result of from the query. The script then encodes the returned row into a JSON packet and sends it to the client. The client then goes through its steps of converting the packet as described previously.

At the point that the client sends another HTTP POST informing the server that it has received valid data to perform the matrix multiplication upon, the server will then make a copy of the

DO NOT COPY

row, using the ID that was sent back in the POST, and remove it from the queue preventing multiple copies of the same data packet to go out and be processed several times.

Finally, once the calculations have been completed on the client side, the third and final POST is sent to the server. The JSON packet that is returned contains the calculated value, as well as the matrix id, row and column coordinates. The final method that is ran by the PHP server queries the computed table and if one or more rows are returned, the result is thrown away and a deletion query is ran on the copy table to shore up the data so not to waste the mobile devices CPU cycles since a value has already been computed.

5. Discussion

Once the clients and servers had been programmed I began to run tests to what the difference in performance was between Android platform and pure Java. I had a wide variety of tests that I ran ranging from multiplying and adding ten single value integers, such as 1 or 2, to testing large floating point numbers[10]. The amount of operands varied between one and 500 in each array. Logging the amount of time each method took to complete the operation on a virtual Android platform, the ZTE Racer and Java running on the server.

Not surprisingly, in four of the six tests, the virtual Android performs the calculations the slowest, followed by the physical Android mobile phone, and pure Java running fastest. On the last two test, the virtual machine performed much better than the physical device. I attribute this to the emulator having access to the host PCs floating point unit. The virtual Android performed 300 multiplications and 300 additions on the large floating point numbers in an average of 3.04 seconds. Pure Java performed the same operation in about 10.36 microseconds, almost a factor of 10^5 differences in speed.

In appendix 2 are the results of tests that were run. Each summation was run between 100 and 500 times on an emulated Android device, the ZTE Racer and a Java application. The data sets included between 10 and 500 items in both the row and column arrays.

5.1 Emulated Android Device

On the first test the emulated Android performed one hundred computations on an array of length 10 with each item being equal to 1 in a total of 5.9775892 seconds. These computations took a maximum time being .0897859306 seconds and the minimum being .0303569333 seconds, with the mean being .0597758929 with a standard deviation of 1.74083754. The median was found to be .0590017100

[10] 98846567400000000000000000000000000000000000.12345678934564 To be exact.

DO NOT COPY

The next test was an array of size 10 with each item equal to 10^{-15} so that we could test very small floating operations. In very small or very large floating point operations we will ignore rounding errors. This dataset was computed 100 times the longest the test took was .796302867 seconds; the shortest was .406564868 seconds for a total of 59.6194577 seconds. The mean was .596194577 with standard deviation 1.24479827 and median of .593639108.

The third test was an array of size 10 with each item being equal to 12345.6789. The computation was run 100 times with the maximum time 1.49526913 seconds, the minimum .408711469 seconds and a total running time of 98.258638 seconds. These results returned a mean of .982586387 with standard deviation 3.32424477 and the median being 10.46520821.

On an array of size 10 with items set to 999999999.999999999 the computation was ran 100 times. This returned results with maximum time being 1.1929505705 seconds, minimum .41376101435 seconds and a total running time of 79.1454254 seconds. The mean time was .791454254 seconds and a standard deviation of 2.20307482 and the median of .809589309.

The fifth test was on an array with items being set to double floating point[11] of size 300 and was computed 300 times. The maximum and minimum running times were 2.99285908 and 1.00283585 seconds respectively, for a total run time of 591.11012 seconds. The mean time was 1.97036710 seconds and standard deviation of 5.8364231. The median run time was 1.97197716 seconds. Not surprising this took longer to run and the results. However, the results produced were much more inconsistent as shown with the standard deviation being .58, around .25 larger than the arrays with less data and smaller values.

The final test that was run 500 times with an array of size 500 items and each item being set to .123456789. This again, produced much more consistent results with the maximum running time of 2.99777041 seconds and the minimum time for calculation taking 1.00022703. The calculation took 998.65549 seconds to complete. The mean run time was 1.99731099 seconds and a standard deviation of 5.9378107. The median time was 1.97502820 seconds. Again, with having 400 more items in each array to compute, the increase in running time is expected. However, the standard deviation of .59 being the largest could show inconsistencies in the way the virtual device handles HTTP requests, string manipulation and mathematics.

The results show the emulated device is consistent when running the test a hundred times. However, when sufficiently large datasets are computed several hundred times the standard deviation grows slightly. It is hard to say precisely what causes this. One could presume it has to deal with the way that the Android platform deals with memory management and when processing large amounts of data repeatedly a few extra CPU cycles must be spent clearing space to store the new values that are calculated.

5.2 Physical Android (ZTE Racer)

[11] This again being exactly: 98846567400000000000000000000000000000000.12345678934564

DO NOT COPY

Running the test on the ZTE Racer produced similar results as the virtual device only faster since the application bytecode is compiled to run natively on Android devices not emulated Android devices running on the Intel x86 instruction set. The following are the results that were found from the physical device.

The first test, like that of the virtual machine, was an array of size 100 with all items in the array being set to 1. The computation was ran 100 times and ran for a total of 48.029008 seconds. The maximum and minimum calculations took .695969383 and .209541962 seconds respectively. The mean runtime was .480290084 seconds and a standard deviation of 1.38126020. The median run time was .497506995.

The second test was an array of size 100 with all items being set to 10^{-15}. The test was run 100 times for a total length of 50.478287 seconds. The longest time for the calculation took .697543379 seconds and the shortest took .302120809. The mean time was .504782870 and a standard deviation of 1.18723042, which is closer than the first test. The median time was .513276461 seconds.

The next test had the arrays set to 12345.6789 with length of 100. Again, the test was run 100 times and took 74.433040 to complete. The maximum time the calculation took was just over one second at 1.09990524 and the minimum calculation took a speedy .308417429. The calculated mean time was .744330364 with a standard deviation of 2.30797032. The median run time was .757982509 seconds very close to the mean.

The fourth test had the same specifications as the emulated device, 100 items that was run 100 times. With the larger numbers, it was not surprising to find that the total runtime was higher at 64.799071 seconds with the maximum taking .979225282 and minimum taking .311429277. The mean and median times were again quite close with .647990710 and .662689976 seconds respectively. The standard deviation on these test results was 2.03415636

The largest, in terms of individual numbers passed to the physical device, was again similar to that of the fifth test of the virtual device. This test too was of length 300 and ran 300 times. The total run time of this test was 908.06126 seconds. Surprising the maximum time to complete the calculation was 3.98926537 seconds, almost a full second longer than the virtual device. The minimum time was 2.01684012 seconds. The mean was 3.02687089 seconds with the standard deviation being similar to the virtual device at 5.8493955[12]. The median was just over .01 higher than the mean at 3.03757019 seconds.

The sixth and final test performed on the physical Android device had the items in the array set to 0.12345678934564 and a length of 500. This test was run 500 times and produced a running

[12] Virtual Device standard deviation was 5.8364231 for the same test

DO NOT COPY

time of 1252.41449016 seconds and maximum and minimum running times of 2.99838487 and 2.00040588 seconds. The mean running time was 2.50482898 seconds with a standard deviation of 2.9569446 and the median running time coming in at 2.50139133 seconds.

5.3 Pure Java

Whist running pure Java to perform these calculations, the raw power of a PC versus a mobile device is shown. The exact same calculations were performed with a Java application. The first test results were a foreshadowing of what the rest of the results would return. Like on the virtual and physical Android devices, an array of length 100 with all items set to 1 was sent to the Java application. The application ran for a total of 697.605 microseconds. The maximum and minimum running times were 9.010 and 5.1790 respectively. The average (mean) time for calculations was 6.9760 microseconds with a standard deviation of 1.1996. The median runtime was 6.7267 microseconds.

The second test, just as in the previous two tests, was an array of size 100 with each item set to 10^{-15} and computed 100 times. Because floating point calculations take longer the total running time of 1087.126 microseconds, or just over 1.08 seconds to complete. The maximum calculation time took 15.8530 microseconds and the minimum took 5.6100 seconds. The mean was 10.87126 with a standard deviation of 3.28356. The median run time was close to the mean at 10.73150 microseconds.

The next test, as with the third test that was run on the physical and virtual devices, was run 100 times with the same settings. It ran for a total 1.056282 seconds with the longest calculation taking 31.6630 microseconds and the minimum taking just 5.6920. The mean calculation time took 10.56282 with a standard deviation of 3.98714. The median time of this test was 9.69153 microseconds.

The fourth test, again, was just an expansion of the previous tests by making the values larger. Like the previous three tests this one was size 100 and was run 100 times. The total run time was .8279100 seconds. The max runtime was 17.3390 microseconds and the min was 6.0730 microseconds. The mean time was 8.27910 microseconds with a standard deviation of 1.56728 and a median runtime of 8.15721 microseconds.

Again, needing to test the limits of the system and the programming language the large double floating point number was inserted to an array 300 times and calculations were made 300 times using the same array and values. The results returned took a total of 30.734320 seconds. The longest calculation ran for 30.9100 microseconds and the shortest returning results in 5.8970 microseconds. The mean return time for the calculations was 10.24477 microseconds with a standard deviation of 6.01824. The median run time was 8.32326 microseconds.

DO NOT COPY

The final test was run 500 times on the Java application. The array size, like before was 500 and the values in the array were set as they were previously to 0.12345678934564. The 500 calculations took a total of 45.304850 seconds. The slowest calculation took 17.7210 microseconds to complete and the fastest took 5.5400 microseconds. The average (mean) wait time for the application to complete is calculation was 9.06097 microseconds with a standard deviation of 2.89712. The median time was 8.45532 microseconds.

5.4 Analysis of Data

On the set of test the emulated device took a total of just over 5.97 seconds, the physical device just over 4.80 seconds and the Java application just over 697.6 microseconds (0.0006976 seconds). With the standard deviation on each test being 1.7, 1.3, and 1.1. The difference between the emulated and physical Android devices is small, which is not surprising since it is a very summation to calculate. However, the run time difference between the PC and the Android devices of about 10^4 is quite surprising. The standard deviations are small enough to show that all the data was group pretty closely together and there were few outliers. This test showed that all the devices can perform a simple summation in about the same amount of time, every time.

On the second set of tests, the virtual device took a little more than 59.6 seconds, the ZTE Racer took a little more than 50.4 seconds, and the pure Java application took a little more than 1.08 seconds. The standard deviation for these test were 1.2, 1.1 and 3.2. This test introduced floating point multiplication and addition into the tests. Floating point is known to take longer on Android devices since most do not have a floating point unit and must run a software emulator internally to perform floating point calculations. The difference between the run time on the Android devices is small and what is expected. The difference between the PC running Java and the Android devices is no longer a factor of 4 but just one difference. This shows that for a small amount of floating point calculations, Android may be a suitable for substitute for calculations.

The next test on the emulated Android device took around 98.2, the Racer at around 74.4, and the application on the PC taking around 1.05 seconds. On these tests the standard deviation was 3.3, 2.3, and 3.9. This test introduced a medium sized floating point number. Surprisingly, the Racer, without a floating point unit did the calculations about 24% faster than the emulated device. Not surprisingly, however, the PC completes the computation almost by a factor of two quicker than the Android devices. The standard deviation here is higher than on the previous two. The time it takes to complete this computation is more spread out. Possibly with a larger dataset there is more paging and memory management that is occurring that may slow down the calculations.

DO NOT COPY

The fourth set of data returned the emulated runtime at about 79.1 seconds, physical at about 64.79 seconds, and the PC running the Java application at about .827 seconds. The standard deviation for these tests 2.2, 2.03 and 1.5. An even larger floating point is introduced in this test. However, all three of the devices that this test was run on complete it faster than it did the third test. These results are baffling; the floating point value provided in the previous and this test data both cause rounding errors when represented in binary. I was unable to explain this phenomenon and returned similar results when tested again. However, small standard deviations for all three tests show that the times are in a tight grouping. These results are in contract with the last test in that the Java application is a factor of three faster than either of the Android devices.

On the fifth test the emulated ran for around 591.1 seconds, physical device ran quite a bit longer at around 908.8 seconds, and the PC around 30.7 seconds. Standard deviation 5.8, 5.8 and 6.0 were calculated for the results. The data provided in this test was a very large and complex floating point number, which was hard to represent in binary and consumes a lot of memory to store. The interesting thing to note here is the amount of time it took the virtual device to perform the calculations compared to the ZTE Racer. This test shows that without a floating point unit to aid in floating point calculations, the performance drops to almost half of the emulated device. The standard deviations on this test are the largest of all the tests that were run. When verifying these tests, similar results for the data were again produced just as with verifying the second test. The Android devices keep within a factor of two when producing the results.

The final test showed that the virtual device emulated ran just over 998.6 second, the physical ZTE Racer ran at just over 1252.4 seconds, and the Java application ran at just over 45.3. The standard deviation for these results was 5.9, 2.9 and 2.89. As with the fifth test, the results were somewhat surprising at first. The virtual device finished the task in about two-thirds the times it took the Racer to finish. Again, this could possibly be attributed to the virtual device having access to the PCs floating point unit. The Android devices again performed at about a factor of two slower than the Java application, but it performed sufficiently enough where distributed computing could take advantage of the multiple devices.

DO NOT COPY

5.5 Verifying data

The server does not do a check on whether or not the calculated data that is returned is valid or not. A suggested way of verifying this data is to send the JSON packet out to at least three different clients. Each client would process the data and return it to the server. The server would then check all three results and if two results are the same, then democratically that result is the same. If all three results are different, then the data would be placed back into the queue to be sent out again for processing. Using the democratic system would help prevent very large, or very small, numbers from suffering from rounding errors.

5.6 How Can This Be Applied

Whilst the difference in speed is surprising, at times a magnitude of 10^4, a better designed algorithm to break up and distribute the work to ten thousand mobile devises than the speed difference would be negated. With an estimate 130 million Android (Tsotsis, 2011) devices, ten thousand is only 0.007% of the total market!

Who would use an application that just does distributed matrix multiplication? A good candidate, specifically for the matrix multiplication part of the application, but the distributed computing aspect could be BOINC. Let us now speculate at the possibly of leveraging just the BOINC community to perform distributed matrix multiplication. According to BOINC Stats, the BOINC community has 2,251,291 registered users running 6,512,022 hosts, an average of 2.89 hosts per user. Although a questionnaire was not sent out to each registered member to ask what mobile device they are using and if they would be willing to run a similar project to BOINC on their mobile device; it should be fair to surmise that half the users would have at least one Android or iOS device.

Now, let's assume that half of the community, 1,125,645 users, have an Android or iOS device, which is a fair assumption since there is an estimate 300+ million of these devices on the planet (Tsotsis, 2011). And again, let us assume that half of the users with an Android or iOS device are willing to contribute to the project. We now have 562,822 mobile devices processing data. Using my current design and algorithm, a quarter of the BOINC community could produce roughly the equivalent of 56 PCs for every operation that is performed.

If we introduce a reward system, similar to BOINC, that rewards users with credit based on the amount of data that was processed, the CPU time used. We could introduce competition for who donated the most processing power and how much credit the have received in total and recently. It may be a good way of encouraging more users to use the program.

Inspired by the BOINC's distributed computing model, my goal was to see if mobile devices could be used as a distributed computing grid. The results, whilst positive, were not as good as I had originally desired them to be. The emulated Android device and the underpowered ZTE Racer could not match the speeds of a PC. With that being said, the tests were not a complete disaster; for moderately sized amounts of information, the mobile devices did perform well, usually with in a factor of two or three. Applying these findings to an open source project, such as BOINCOID, one could design an application that allows mobile devices to be used as distributed computing.

DO NOT COPY

As technology improves and new mobile processors are introduced, such as the recently announced nVidia Tegra 3 (code named Kal-El) a quad-core ARM Cortex-A9 chip with clock speeds of up to 1.5GHz (TabletUpdate Special Report: Nvidia Tegra 3 – codename Kal-El, 2011), and the Apple A5 1GHz dual-core process, the idea of mobile distributed computing will soon become a reality.

5.7 The effects on the phone

During the test the ZTE Racer became extremely hot! Mobile devices have little or no cooling mechanisms which could explain the high temperature of the Racer. Unlike PCs, mobile devices were not built with high data processing in mind, so they have no active cooling devices, such as fans, to dissipate any excess heat. So what does that mean for phones running this application? Well, in future implementations the application would be only allowed to run for a certain amount of time or until the mobile device became too hot to run at optimal speeds. Referencing the Android developers' page about sensors they make reference to a temperature sensor but do not say which, if any, devices actually contain that specific sensor.

There are many reasons that it is not a good idea to run the mobile device at a high temperature too long I will briefly discuss two of them. The first reason to not run mobile devices are a high temperature too long being it may end up having an adverse effect on the battery. That is, "exposing the battery to high temperature and being at full state-of-charge for an extended time can be more damaging than cycling [charges]." (Battery University, 2011) That is to say if the battery is fully charged and the phone is running at a heightened temperature, it could be worse for the battery than discharging and charging it multiple times.

Second, according to Be You Own IT "higher temperatures exacerbate a problem known as Electromigration." (Be You Own IT) Eletromigration is the movement of material carried by ions do to momentum. Electromigration is far beyond the scope of this paper and project and probably best to let electrical engineers to ponder and discuss. Electromigration, however, is analogous to running your hand over a wall with cracked paint. The warmer and moister[13] your hand is, the more likely that more paint will chip away as you run your hand over it. Exacerbated electromigration, in a worst case scenario, could eventually break the connection between one of the devices integrated circuits and the semi-conductor, which would render the mobile device useless.

6. Conclusion

We have seen that using many Androids devices we can successfully create a distributed network for large scale processing. Using asynchronous communication in a client-server

[13] Moisture is not applicable to electromigration but is a good idea to not get any electronics wet!

DO NOT COPY

paradigm and applying a consumer-producer design pattern I was able to successfully send information between the LAMP server and multiple Android devices. The data that the Android devices received, processed in an acceptable amount of time and were able to transmit it back to the server for storage was acceptable, albeit slow. Not too surprising was the fact that the Android devices performed the summations slower than a normal PC. By analysing at the time data, specifically the standard deviations for each set of data, we see that similar calculations should be produced in similar amounts of time. The test west successful and with more extensions and a better algorithm, mobile distributed computing could be available to volunteers world-wide to perform matrix multiplication, sequence seeking, or even find E.T.

Now that we have a successful design, we must now extend it to take note of what effects running strenuous processes on devices that were not originally meant to handle extended processing. Further iterations of my design, or any other, must take in consideration there is no active heat dissipation on most phones, electromigration, and the application must detect whether the mobile device is charging as well as a setting that allows the user to run the application on battery power alone.

Where is the next step? As Apple and Android devices become quicker, the difference in calculation times between mobile devices and their PC counterparts should decrease. If we then apply the same methods that were produced here to something to the BOINCOID project we should see the reality of having a distributed mobile network helping to solve some of the world's hardest questions.

DO NOT COPY

Works Cited

FAQ. (2009, 05 16). Retrieved 06 13, 2011, from Boincoid: http://boincoid.sourceforge.net/faq.html

TabletUpdate Special Report: Nvidia Tegra 3 – codename Kal-El. (2011, June 01). Retrieved August 09, 2011, from TableUpdate: http://tabletupdate.com/tabletupdate-special-report-nvidia-tegra-3-codename-kal-el/

Battery University. (2011). *How to Prolong Lithium-based Batteries*. Retrieved August 06, 2011, from Battery University: http://batteryuniversity.com/learn/article/how_to_prolong_lithium_based_batteries

Be You Own IT. (n.d.). *Why does heat kill electronics?* Retrieved August 5, 2011, from Be Your Own IT: http://www.beyourownit.com/blog/why-does-heat-kill-electronics/

Doolan, D. C., Tabirca, S., & Yang, L. T. (2006). Mobile Parallel Computing. *ISPDC '06 International Syumposium on Parallel and ditrubted computing, 2006.* , (pp. 161-167).

Google. (2011, July 02). *Platform Versions*. Retrieved July 02, 2011, from Android Developers: http://developer.android.com/resources/dashboard/platform-versions.html

GSM Arena. (n.d.). Retrieved 07 10, 2011, from GSM Arena: http://www.gsmarena.com/nokia_6630-811.php

GSM Arena. (n.d.). Retrieved 07 12, 2011, from GSM Arena: http://www.gsmarena.com/nokia_6680-1045.php

JSON. (2011). *JSON: The Fat-Free Alternative to XML*. Retrieved July 24, 2011, from JSON: http://www.json.org/xml.html

Programmer XR. (2011, 04 01). *Android tutorial series: From SQL/ PHP to XML/ JSON to Android ListView!* Retrieved 06 2011, 29, from Programmer XR - Just Someone Who Explains Andriod Code: http://p-xr.com/android-tutorial-series-from-sql-php-to-xml-json-to-android-listview/

SlateDroid1. (n.d.). *Introduction to the Pandigital Novel*. Retrieved 07 18, 2011, from SlateDroid: http://www.slatedroid.com/wiki/index.php/Introduction_to_the_Pandigital_Novel

SlateDroid2. (n.d.). *Samsung S3C6410*. Retrieved 07 19, 2011, from SlateDroid: http://www.slatedroid.com/wiki/index.php/Samsung_s3c6410xh-53

Tsotsis, A. (2011, July 14). *Google Android Now On 130M Total Devices, With 6B App Downloads*. Retrieved July 29, 2011, from TechCrunch: http://techcrunch.com/2011/07/14/google-android-now-on-130-million-total-devices/?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+Techcrunch+%28TechCrunch%29

DO NOT COPY

ZTE Blade. (n.d.). Retrieved 07 05, 2011, from GSM Arena: http://www.gsmarena.com/zte_blade-3391.php

DO NOT COPY

Appendix 1 – Code Listing

Appendix 1.1 – Main.java

```
package uk.ac.notts.mdj.dissertation;

import uk.ac.notts.mdj.pack.R;
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.HttpClient;
import org.apache.http.client.ResponseHandler;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.BasicResponseHandler;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.util.EntityUtils;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import android.app.ListActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.ArrayAdapter;
import android.widget.LinearLayout;
import android.widget.TextView;

public class Main extends ListActivity {
    private static final int MENU_START = 0;
    ArrayList<String> listItems=new ArrayList<String>();
    ArrayAdapter<String> adapter;


    /* Creates the menu items */
    public boolean onCreateOptionsMenu(Menu menu) {
        menu.add(0, MENU_START, 0, "Start");
        return true;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
        setContentView(R.layout.listplaceholder);
        matrixmult();
    }
```

DO NOT COPY

```
    /* Handles item selections */
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
    case MENU_START:
    matrixmult();
            return true;
        }
        return false;
    }

    public void matrixmult() {
        boolean process = true;

        LinearLayout linlay = (LinearLayout) this.findViewById(R.id.llayout);
        final int N = 20;
        int count = 0;
        final TextView[] theResults = new TextView[N];

        while (process) {
    String result = null;
       InputStream entitystream=null;
       try{
               ArrayList<NameValuePair> nameValuePairs =
                           new ArrayList<NameValuePair>();
               HttpClient httpclient = new DefaultHttpClient();
               HttpPost httppost =
                   new HttpPost("http://10.0.2.2/Dissertation1/packet.php");
               httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
               HttpResponse response = httpclient.execute(httppost);
               HttpEntity entity = response.getEntity();
               entitystream = entity.getContent();
       }
       catch(Exception e){
               Log.e("log_tag", "Error in http connection "+e.toString());
       }


       //convert response to string
       try{
               BufferedReader reader = new BufferedReader(
                      new InputStreamReader(entitystream,"iso-8859-1"),8);
               StringBuilder stringbuilder = new StringBuilder();
               String line = null;
               while ((line = reader.readLine()) != null) {
                       stringbuilder.append(line + "\n");
               }
               entitystream.close();
               result = stringbuilder.toString();
       }
       catch(Exception e){
               Log.e("log_tag", "Error converting result "+e.toString());
       }

       //parse json data
       try{
               JSONArray jArray = new JSONArray(result);
```

DO NOT COPY

```
                JSONObject json_data = jArray.getJSONObject(0);
                int process_ID = Integer.parseInt(json_data.getString("_ID"));
                String row_temp [];
                row_temp = json_data.getString("ROWS").split(",");

                String col_temp[];
                col_temp = json_data.getString("COL").split(",");

                String row_cor = json_data.getString("ROW_COR");
                String col_cor = json_data.getString("COL_COR");
                String matrix_id = json_data.getString("MATRIX_ID");
                String httppostStr =
              "http://10.0.2.2/Dissertation1/packet.php?updateID="+process_ID;

                double total = 0;
                if (row_temp.length == col_temp.length) {
                 //Tell server to remove this data packet from the queue
                 try {
                     DefaultHttpClient hc=new DefaultHttpClient();
                     ResponseHandler <String> res=new BasicResponseHandler();
                     HttpPost postMethod=new HttpPost(httppostStr);
                     //String response=
                     hc.execute(postMethod,res);
                     }
                    catch (Exception e){
                     Log.e("log_tag", "Error in http connection
                              "+e.toString());
                    }

                    for (int j = 0; j<row_temp.length; ++j) {
                     total += Double.parseDouble(row_temp[j]) *
                              Double.parseDouble(col_temp[j]);
                    }
                }
              try {
                JSONObject json = new JSONObject();
                try {
                     json.put("data", total);
                     json.put("matrix_id", matrix_id);
                     json.put("row_cor", row_cor);
                     json.put("col_cor", col_cor);
                     }
                catch (Exception e){
                    Log.e("Error in HTTP", "HTTP Request error:
                      "+e.toString());
                }

                HttpClient httpclient = new DefaultHttpClient();
                HttpPost httppost =
                     new HttpPost("http://10.0.2.2/Dissertation1/packet.php");
                try {
                      List<NameValuePair> nameValuePairs =
                         new ArrayList<NameValuePair>(2);

                     // Add two POST request variables
                     nameValuePairs.add(new BasicNameValuePair("data",
                                          Double.toString(total)));
                    nameValuePairs.add(new BasicNameValuePair("payload",
                                         json.toString()));
                    httppost.setEntity(
                               new UrlEncodedFormEntity(nameValuePairs));
           // Execute HTTP Post Request
                   HttpResponse response =
                                httpclient.execute(httppost);
                   String responsestr =
                                EntityUtils.toString(response.getEntity());
                        try {
                          final TextView rowTextView = new TextView(this);
                          rowTextView.setText("Returned Result: " +
                                               Double.toString(total));
                          linlay.addView(rowTextView);
                          theResults[count] = rowTextView;
                          count++;
                       }
                       catch (Exception e) {
                           count = 0;
                       }
            }
            catch (Exception e){
                Log.e("Error in HTTP Request", "HTTP Request error:
                      "+e.toString());
            }
        }
        catch (Exception e){
            Log.e("log_tag", "Error in http connection "+e.toString());
        }
    }
    catch(JSONException e){
        try {
             final TextView rowTextView = new TextView(this);
             rowTextView.setText("Nothing data returned from server");
             linlay.addView(rowTextView);
             theResults[count] = rowTextView;
            }
            catch (Exception e1) {
                count = 0;
            }
        Log.e("log_tag", "Error parsing data "+e.toString());
        break;
       }
     }
   }
}
```

DO NOT COPY

Appendix 1.2 – Packet.php

```
<?php
    if (isset($_GET['updateID'])) { $update = $_GET['updateID']; }
    if (isset($_POST['data'])) { $compdata = $_POST['data']; }

    mysql_connect("127.0.0.1","root","crimson");
    mysql_select_db("Food");

function get_json() {
        $sql=mysql_query("select * from WAITING") or die(mqsql_error());
        while($row=mysql_fetch_assoc($sql)) {
            $output[]=$row;
            break;
        }
        print(json_encode($output));
        mysql_close();
}

    if (!isset($_REQUEST) || empty($_REQUEST)) { get_json(); }

    else {
        //Android client received a valid copy of the data
        if (isset($update) && (!isset($compdata))) {
            //Move row from available packs to backup
            $query = mysql_query("select * from Copy where
                                  MATRIX_ID='$update'");
                    //Verifies that the entry isn't already there.
            if(mysql_num_rows($query)==0) {
                $sql=mysql_query("INSERT INTO Copy (MATRIX, ROWS, COL,
                                  ROW_COR, COL_COR, MATRIX_ID)
                                  SELECT _ID, ROWS, COL, ROW_COR, COL_COR,
                                  MATRIX_ID FROM WAITING WHERE
                                  _ID = '$update'") or die(mqsql_error());
                if ($sql) {
                        //If copy was successful,
                              remvoe the row from the WAITING table
                    $sql=mysql_query("DELETE FROM WAITING WHERE
                                 _ID='$update'") or die(mysql_error());
                }
            }
        }

        else if (!isset($update) && (isset($compdata))) {
            $_POST['payload'] = stripslashes($_POST['payload']);
               //Required if you have magic_quotes on.
            $payload = $_POST['payload'];
            $payloadObj = json_decode($payload);

            /*Check for data, check the sql was good
              (matrix_ID). Move back to queue
            or delete from copy */
            $check_sql = mysql_query("select * from COMPUTED WHERE
                                    MATRIX_ID='$payloadObj->matrix_id'
                                    and row_cor='$payloadObj->row_cor'
```

```
                                                      and col_cor='$payloadObj->col_cor'");

              $numResults = mysql_num_rows($check_sql);
              if ($numResults >= 1) {
                              $delete_sql = mysql_query("delete from Copy WHERE
                                     MATRIX_ID='$payloadObj->matrix_id'
                                     and row_cor='$payloadObj->row_cor' and
                                                      col_cor='$payloadObj->col_cor'");
              } else {
                               $insert_sql = mysql_query("insert into COMPUTED VALUES
                                  (Null,'$payloadObj->data', '$payloadObj->matrix_id',
                                      '$payloadObj->row_cor', '$payloadObj->col_cor')");
                              echo json_encode($payloadObj);
                      }
           }
           mysql_close();
     }
?>
```

DO NOT COPY

Appendix 1.3 – SQL Schema

```
-- phpMyAdmin SQL Dump
-- version 3.3.10deb1
-- http://www.phpmyadmin.net
--
-- Host: localhost
-- Generation Time: Sep 04, 2011 at 06:37 AM
-- Server version: 5.1.54
-- PHP Version: 5.3.5-1ubuntu7.2

SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;

--
-- Database: `Dissertation`
--
-- --------------------------------------------------------
--
-- Table structure for table `COMPUTED`
--

CREATE TABLE IF NOT EXISTS `COMPUTED` (
  `_ID` int(11) NOT NULL AUTO_INCREMENT,
  `DATA` varchar(100) NOT NULL,
  `MATRIX_ID` int(11) NOT NULL,
  `ROW_COR` int(11) NOT NULL,
  `COL_COR` int(11) NOT NULL,
  PRIMARY KEY (`_ID`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 AUTO_INCREMENT=46 ;

--
-- Dumping data for table `COMPUTED`
--
-- --------------------------------------------------------
--
-- Table structure for table `Copy`
--

CREATE TABLE IF NOT EXISTS `Copy` (
  `_ID` int(11) NOT NULL AUTO_INCREMENT,
  `MATRIX` int(11) NOT NULL,
  `ROWS` varchar(100) NOT NULL,
  `COL` varchar(100) NOT NULL,
  `ROW_COR` int(20) NOT NULL,
  `COL_COR` int(20) NOT NULL,
  `MATRIX_ID` int(11) NOT NULL,
  PRIMARY KEY (`_ID`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 AUTO_INCREMENT=140 ;
```

```
--
-- Dumping data for table `Copy`
--

-- --------------------------------------------------------

--
-- Table structure for table `WAITING`
--

CREATE TABLE IF NOT EXISTS `WAITING` (
  `_ID` int(11) NOT NULL AUTO_INCREMENT,
  `ROWS` longtext NOT NULL,
  `COL` longtext NOT NULL,
  `ROW_COR` int(11) NOT NULL,
  `COL_COR` int(11) NOT NULL,
  `MATRIX_ID` int(11) NOT NULL,
  PRIMARY KEY (`_ID`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 AUTO_INCREMENT=166 ;

--
-- Dumping data for table `WAITING`
--
```

DO NOT CO

Appendix 2 – Aggregated Log Files

2.1 – Emulate Android Device

Test 1

```
Array Size:  100
Array Contains Elements = 1
Max: .897859306
Min: .303569333
Total: 59.7758928918
Mean: .597758929
Standard Deviation: 1.74083754
Median: .590017100
Results:
[7.32802440, 6.17368243, 3.77354693, 5.81214948, 3.56440001, 5.29310017, 4.68800098,
7.77744893, 3.62399405, 6.84435989, 3.09953554, 4.13481564, 5.86729115, 6.93123103,
5.85858372, 7.10562513, 8.97859306, 8.59443005, 6.17546874, 3.45005542, 5.74523068,
6.41005206, 8.80886979, 8.16844136, 8.66747250, 3.75643240, 6.01653579, 8.32338261,
7.51816305, 3.62556736, 6.88025085, 6.55687704, 7.05581628, 7.62977203, 6.63781347,
7.90902663, 7.68901437, 5.25940999, 4.47844554, 5.51102970, 3.86116083, 7.39562840,
5.80157426, 8.80706667, 5.84854881, 8.49538119, 6.38497250, 3.59317960, 4.22066430,
8.91316938, 7.75844311, 5.89577344, 3.09845107, 4.86729954, 5.47594097, 4.24902162,
4.71187754, 5.33348803, 3.92450458, 3.28419539, 6.51887240, 8.12907138, 3.03569333,
3.56062111, 6.02563326, 5.51557083, 5.51855717, 6.48654034, 7.82405398, 4.29523501,
3.69871216, 6.59651317, 7.36593292, 8.02657135, 3.33023195, 8.76264808, 4.31387336,
5.13610361, 8.72405306, 8.93129707, 4.29944619, 4.75054189, 7.11513124, 3.31331488,
4.98117856, 7.27227373, 4.49810733, 6.43319021, 6.39586107, 5.34853486, 5.50800778,
7.43204806, 4.98919104, 7.28111128, 8.88827470, 3.33489456, 7.32447705, 6.63327527,
3.96552138, 6.55648083]
```

Test 2

```
Array Size:  100
Array Contains Elements = .000000000000001
Max: 7.96302867
Min: 4.06564868
Total: 596.194577
Mean: 5.96194577
Standard Deviation: 1.24479827
Median: 5.93639108
Results:
[7.60901905, 4.16790240, 4.15344146, 4.91639973, 4.73474197, 5.40746115, 7.68112949,
5.97482444, 5.37981655, 7.68245182, 7.68271895, 7.01576494, 6.32954241, 7.54652711,
7.35823065, 7.65372652, 6.05235421, 5.18265365, 6.30148476, 4.99232171, 4.41753029,
```

DO NOT COPY

```
7.11068886, 6.76480091, 5.45664542, 4.62565699, 4.29402099, 4.36331678, 6.83807087,
6.38813536, 4.07460859, 4.08558006, 6.26801306, 6.59292060, 5.52021854, 7.78176029,
5.02005732, 7.39173287, 6.27731001, 4.06564868, 5.32126180, 5.01030279, 5.24338770,
5.64460103, 4.22807989, 5.41469636, 7.63555296, 4.13723907, 4.25992062, 5.42737865,
6.62881237, 7.45902565, 6.34216766, 6.01209043, 6.69405739, 7.00958608, 6.17813844,
6.15640429, 7.07146368, 7.85659833, 6.98062098, 4.81569716, 7.45679411, 4.32694455,
6.10815208, 7.93045875, 7.89691315, 7.82619404, 5.93330253, 7.69931458, 6.97090532,
7.46695964, 4.84241909, 4.64733685, 6.86436982, 6.59943112, 7.88852266, 4.48503947,
7.78259096, 4.65517342, 5.93017558, 4.76780864, 5.25677642, 5.34901105, 4.28874264,
6.60443582, 4.48808342, 7.48112906, 5.57333454, 4.13495208, 5.18344954, 7.22910823,
4.33135832, 4.96640776, 5.88751922, 4.32540876, 4.91067307, 5.80755942, 6.34191790,
5.30656177, 7.96302867]
```

Test 3

```
Array Size:  100
Array Contains Elements = 12345.6789
Max: 14.9526913
Min: 4.08711469
Total: 982.58638
Mean: 9.82586387
Standard Deviation: 3.32424477
Median: 10.46520821
Results:
[6.62140787, 12.83906857, 8.84227328, 5.42257310, 12.30159767, 13.77759300,
11.30091560, 5.73090707, 6.51972826, 4.20257715, 11.91574638, 6.76862122, 4.96814404,
6.70173724, 10.46386937, 14.12240538, 12.60851717, 14.65091917, 8.38873762, 9.45992787,
5.99934035, 4.30503011, 13.28823747, 13.99377163, 14.41826608, 12.88697390, 6.08607536,
12.99912986, 14.04630620, 6.55619055, 14.94039168, 10.80391078, 4.08711469,
12.27405738, 11.72470791, 14.95269139, 8.42300405, 10.92059867, 4.45146639,
10.67711427, 6.31657169, 5.89398870, 6.47299854, 13.76955257, 10.32665618, 6.74435084,
11.29576004, 10.56085308, 10.77695956, 7.13818816, 7.23658930, 8.51413140, 5.93855770,
6.60500075, 14.54911051, 14.48625685, 12.22879167, 12.55568948, 4.54206241, 9.86236639,
13.31655920, 13.64352168, 12.71396084, 8.03709861, 9.33689136, 10.67550491, 5.86147022,
11.75800469, 12.50637805, 14.12608201, 4.57767752, 11.95344023, 4.78866458,
14.07467019, 7.20991885, 5.95916452, 9.00354376, 6.58766386, 11.45916130, 12.22671673,
12.50161394, 12.15601673, 13.23987557, 12.55264351, 14.19081967, 8.19822943,
12.28711969, 6.04615244, 10.07950180, 12.09422045, 11.31186192, 4.55506354,
10.19743147, 7.09892637, 8.97887399, 6.86059010, 12.88737141, 4.85009698, 14.48709853,
6.94070507]
```

Test 4

```
Array Size:  100
Array Contains Elements = 999999999.999999999
Max: 11.929505705
```

DO NOT COPY

Min: 4.1376101435
Total: 791.454254
Mean: 7.91454254 (~.659 sec) 6.594637116E8
Standard Deviation: 2.20307482
Median: 8.09589309
Results:
[5.79499194, 5.02574296, 6.91698058, 11.30458982, 7.88133662, 8.30060627, 9.88903534, 6.22658803, 5.44546430, 4.19132777, 11.17563195, 10.52964708, 10.24698773, 8.51036539, 8.13957482, 5.26985186, 4.51249414, 4.35900122, 4.53802313, 4.16423329, 5.92246263, 6.35189842, 10.44632498, 8.12216858, 10.20120508, 6.57226462, 5.60564102, 5.58286508, 10.08067860, 9.91173122, 9.53985318, 4.13761014, 9.88968092, 10.43895444, 8.52193790, 5.89345832, 10.29461104, 9.92705161, 5.01163543, 4.80485042, 8.78072702, 5.62605214, 9.94691263, 5.46248038, 5.11969704, 9.23797581, 10.88931073, 9.94039180, 6.79712999, 9.79679250, 10.25904422, 9.74511746, 9.72636956, 6.57800972, 8.29079343, 10.40832989, 10.24558642, 10.80383926, 6.70356252, 11.92950571, 7.08507352, 9.15656833, 4.81453702, 6.67913137, 7.53372853, 8.16731548, 6.50639336, 7.00864458, 6.14470385, 5.77564225, 11.39426296, 11.74997470, 10.24412840, 7.00408525, 11.67807162, 11.61942832, 7.52734495, 6.92038546, 7.12761971, 6.31872666, 6.47832840, 6.53127827, 8.96647033, 7.59433237, 4.65765481, 7.05843629, 8.70648533, 5.65151580, 8.22620586, 9.47549178, 10.03766090, 4.62130884, 8.09180184, 5.04593621, 8.87121019, 9.68576864, 9.94600116, 9.16525520, 9.96137026, 6.25898955]

<u>Test 5</u>
Array Size: 300
Array Contains Elements =
98846567400000000000000000000000000000000.12345678934564
Max: 2.99285908
Min: 1.00283585
Total: 591.11012
Mean: 1.97036710
Standard Deviation: 0.58364231
Median: 1.97197716
Results:
[2.88003476, 1.62955072, 1.42558047, 2.07535942, 1.93792261, 2.85082220, 2.20957684, 2.57359628, 1.09298731, 2.06710956, 2.49750103, 1.67454768, 1.71785121, 2.89779374, 2.74680004, 1.12051874, 1.66273378, 2.62180068, 1.76519424, 2.34164377, 1.81127952, 1.83253490, 2.01260051, 1.60999930, 1.14778268, 1.30029952, 2.56333782, 2.18256244, 1.34685196, 2.93625835, 2.53803478, 2.73204599, 2.41856382, 2.34555409, 1.78239717, 1.38620205, 1.94373395, 1.81244067, 2.20349633, 1.04350008, 2.39019359, 1.95188537, 1.59968531, 1.74076159, 2.07854506, 1.03756776, 2.38482937, 1.25369606, 1.68245345, 1.28467987, 1.28005653, 1.24908834, 2.35424553, 1.82883536, 1.08055792, 1.07287300, 1.20495346, 2.99285908, 1.26451574, 2.29955915, 1.19186893, 1.20684201, 1.50068831, 2.45236503, 1.31359238, 1.74100971, 1.39518723, 1.21198565, 1.51761967, 1.00830211, 2.74853640, 1.60964470, 1.97314377, 2.11081155, 1.99371360, 2.50409876, 1.27024976,

DO NOT COPY

```
2.57662826, 2.33593563, 1.29348665, 2.11061464, 1.69609642, 1.71036634, 1.52404581,
1.95884384, 1.47858872, 1.71964613, 1.10431231, 2.32403490, 1.80028247, 2.85496072,
2.82118362, 2.11451193, 2.76451433, 2.51067120, 2.39451587, 2.88893358, 2.33554036,
2.67568984, 1.54415098, 2.04951221, 2.91857595, 1.84067721, 1.52380308, 1.63285836,
1.29411108, 1.19854134, 2.23541232, 2.67231222, 1.21169026, 2.27975335, 2.20868067,
1.22656229, 1.82037502, 1.95981107, 1.74488891, 1.18497801, 1.98589234, 2.32993160,
1.31715938, 2.35318738, 2.92034905, 2.05761473, 1.36441613, 1.30692919, 2.85678144,
2.17785160, 1.07324692, 2.90270240, 2.13153720, 1.48685035, 1.74711710, 1.19348635,
1.57717289, 1.27001476, 2.93774740, 1.12380394, 1.75351678, 1.03390091, 2.48107858,
1.02750936, 1.72922987, 2.51222379, 1.90482225, 1.76230405, 1.61964745, 2.42582932,
1.20160136, 1.89559893, 2.93886276, 2.49221680, 2.00854977, 2.07444534, 1.19540012,
2.44153408, 2.97026898, 1.10654694, 2.11644056, 1.30634080, 2.99200156, 1.88597683,
2.46938935, 2.73010855, 2.11129474, 2.13955431, 1.89384767, 1.14640914, 1.19427298,
2.23546582, 1.75713334, 2.71170432, 2.06384249, 2.55132871, 2.33100025, 2.39567912,
2.77735180, 2.31084975, 1.49722741, 2.02915270, 2.11199270, 2.81431207, 2.06032871,
2.20922557, 1.42342542, 1.24612558, 2.87157775, 2.72306119, 2.22929549, 1.72178279,
1.08464808, 1.97702676, 1.60373623, 1.69742384, 2.83523519, 1.24592526, 2.71227211,
2.95192952, 2.91002267, 1.46400993, 2.52691927, 1.19766152, 2.47040621, 1.19538930,
1.64914880, 2.65495323, 1.49538389, 1.31023476, 1.81346469, 2.22708511, 2.79200393,
1.52438923, 1.70068721, 2.65896864, 1.84351071, 2.46786912, 2.93342448, 2.50213004,
2.63903073, 2.55780427, 1.98824978, 2.58432632, 1.29389188, 2.74822118, 2.38709906,
1.84682995, 2.66518600, 1.13804417, 2.32716972, 1.35875804, 1.64222203, 1.54088538,
1.27450684, 1.07017390, 1.19954173, 2.82249609, 1.60626912, 2.82843146, 1.07471013,
1.44277190, 2.09163367, 2.69797385, 1.55502593, 2.97017996, 2.75354778, 1.15523213,
2.95668668, 2.82552904, 1.99400207, 2.42413821, 1.62165566, 2.27053065, 2.88609870,
1.92641116, 1.47443403, 2.52023148, 1.76150423, 2.32172859, 2.33766250, 1.48829960,
2.57465431, 2.26972213, 2.14390701, 2.53993472, 2.97843036, 2.26961164, 2.57064062,
2.60515842, 2.87122597, 1.21403463, 2.18750886, 1.68597354, 1.18749643, 1.03413919,
1.68277953, 1.99532644, 1.32752700, 2.21875425, 1.57416953, 2.20425253, 1.34704544,
2.48001230, 2.69092089, 1.57978667, 2.18995492, 1.97170269, 1.49953767, 2.81703678,
1.50707592, 1.34331052, 1.90004397, 1.00283585, 1.01217937, 2.21051388, 1.40309230,
1.51920310, 1.70393337, 2.33015281, 1.03413770, 2.61945502, 2.49158637]
```

<u>**Test 6**</u>

Array Size: 500
Array Contains Elements = 0.12345678934564
Max: 2.99777041
Min: 1.00022703
Total: 998.65549
Mean: 1.99731099
Standard Deviation: 0.59378107
Median: 1.97502820
Results:

```
[1.82136543, 1.59003535, 2.14073751, 2.17883293, 1.01844872, 2.84170491, 2.55649238,
```

DO NOT COPY

```
1.09601548, 1.97466523, 1.15955838, 2.35301912, 1.85759362, 1.46461870, 2.12101313,
1.77952075, 1.35455885, 2.50602365, 1.58653985, 2.78253461, 1.70216942, 2.87780457,
2.36055253, 2.91142294, 1.06337085, 2.98405173, 2.17732134, 1.44683703, 1.15419397,
1.84890800, 2.82138873, 1.24781975, 2.79592775, 2.57905620, 2.98794217, 2.41564828,
1.75255684, 2.65357603, 2.80044773, 2.34848521, 1.64352332, 2.55704031, 2.98302127,
1.54095206, 1.64143286, 2.90542187, 1.38128247, 1.90005126, 1.39702263, 1.49011236,
1.10042421, 1.59306010, 2.61799746, 1.89224473, 2.42560729, 2.51744928, 2.12912863,
2.07411163, 1.97181286, 1.66867233, 1.18614677, 2.33763783, 1.00022703, 2.37520378,
2.57931747, 2.37785128, 1.00731618, 1.00871186, 2.99777041, 1.16366428, 2.72437294,
2.84130390, 2.53878443, 2.05356817, 1.68902065, 2.21221479, 1.04340394, 1.95328772,
2.25832540, 1.98298780, 2.54159352, 1.88315838, 1.25073469, 2.54884945, 2.78673336,
2.85268382, 2.07473537, 2.88541065, 1.89524744, 2.86023276, 2.02431277, 1.60675053,
2.74344347, 2.92002027, 2.54416804, 2.33784140, 1.47832661, 1.67624858, 2.84852447,
1.88185667, 1.81912394, 2.14317520, 1.32769699, 1.39237431, 2.94088347, 2.54200346,
1.49074666, 1.67314698, 1.16313441, 2.05191892, 2.10837830, 1.78489132, 2.91437815,
1.17080703, 1.49206435, 1.77963969, 1.11149489, 1.72429668, 1.49326817, 1.39307686,
1.05325071, 2.04477275, 1.88818445, 2.71478795, 1.38694549, 1.05130764, 2.51596773,
1.91311490, 1.60742897, 2.98824623, 1.48765111, 1.45539892, 1.96108103, 2.06389145,
2.83780260, 1.63034070, 1.27086859, 2.29650083, 2.56078894, 1.86755305, 1.97503553,
1.55615776, 1.14733515, 1.66423394, 1.50408667, 2.20012485, 2.85926789, 1.31130020,
1.11237554, 1.17471845, 2.27666248, 1.19798172, 1.16278224, 2.20225126, 1.91761668,
2.10469073, 2.43922091, 1.14243371, 1.66107479, 2.74754260, 1.96972854, 2.19937044,
1.96005869, 1.16271078, 1.60990168, 1.02493635, 1.71873850, 1.96453038, 1.29744507,
2.10705009, 2.49152575, 2.27680376, 2.35658219, 1.14578574, 2.65803863, 1.29860946,
1.53799405, 2.54601980, 1.26433601, 2.56753403, 1.90718058, 2.81929458, 1.76377968,
1.06592420, 2.90382575, 1.81157816, 2.63449208, 2.70176526, 1.26914520, 2.55587955,
1.26555217, 2.42160630, 2.04682812, 2.96398098, 2.85971385, 2.98757426, 1.48773917,
2.90898727, 2.43939020, 1.94502614, 1.79920938, 2.33166649, 1.57403116, 1.78034055,
2.89828612, 1.33281469, 1.43612994, 2.93055491, 1.42744255, 2.55804260, 2.20525957,
1.64325133, 1.09928758, 2.68011690, 2.32091675, 1.26146425, 2.33854149, 2.57714642,
2.10314067, 2.78531695, 2.15977460, 1.03499093, 1.95363521, 2.46731221, 2.25487635,
1.28943009, 1.46164620, 1.27657039, 2.53044677, 1.04728338, 1.54574184, 2.09560971,
2.88484275, 1.15781299, 1.85585842, 1.56914673, 2.02304567, 2.01438063, 2.93828798,
2.99226646, 1.58467274, 1.34754813, 2.25894919, 1.68911178, 2.56280573, 2.50161959,
2.54178726, 2.42547586, 1.98714049, 2.58171308, 1.52841795, 2.72075193, 2.28993404,
1.72342481, 1.93720553, 2.68462672, 1.65652759, 2.42778548, 1.51290383, 2.59249855,
2.91073504, 1.31315351, 2.58461007, 1.95524304, 2.97942958, 2.51143312, 2.07541485,
1.66066931, 1.28023654, 1.12193243, 1.80118625, 2.11439946, 2.76653455, 2.75368624,
2.04696475, 2.95144178, 1.67259045, 2.66941750, 2.38283260, 2.76663275, 1.84041596,
1.36731419, 2.87836227, 1.39992295, 2.74152355, 1.10443276, 2.77117402, 1.71440581,
1.69473611, 1.78153657, 1.89172800, 1.92276964, 2.94481503, 2.59191247, 2.60113158,
2.80395622, 1.72950746, 2.03425203, 1.93100496, 1.13412560, 2.80717069, 1.44967771,
2.65678205, 1.88960814, 2.10811816, 1.38065716, 1.84196448, 2.36690010, 2.15514909,
2.30128844, 2.74788219, 1.45120397, 1.93458819, 1.34628228, 2.54160320, 1.52970472,
```

DO NOT COPY

```
2.12380816, 1.23750276, 1.04538855, 2.17107127, 1.48709684, 1.73919601, 2.56494141,
1.99141234, 1.34955770, 2.79707745, 1.83074796, 2.63506615, 2.59051959, 2.39524957,
1.22739472, 1.33417402, 2.83551672, 2.85636674, 1.08923741, 2.41889841, 1.29972045,
1.34802867, 1.18744279, 1.28773447, 2.62081619, 2.17661660, 1.75883652, 2.11322078,
2.23051891, 2.29108159, 1.63934892, 2.13191981, 1.00800194, 2.64990588, 1.98956517,
1.04386673, 2.74624125, 1.28423309, 1.49196374, 2.32986274, 2.79171264, 1.40653944,
1.00917197, 1.22004303, 1.13907208, 2.53724763, 1.11545170, 1.26659714, 2.19175908,
2.42691176, 1.38799007, 1.43386948, 1.20613068, 2.25937265, 1.16723909, 1.40562817,
2.14005128, 1.35331371, 2.11494304, 1.54159516, 2.37954195, 1.03327440, 1.04272597,
2.96780330, 1.01101589, 2.09384699, 2.55129683, 1.38601501, 1.19988072, 2.70138426,
1.41121208, 1.56316053, 1.43461785, 1.06842424, 2.95384007, 2.88567700, 2.67621637,
1.30748121, 2.41726515, 2.31310512, 2.83993299, 2.87081365, 1.45741114, 1.67709765,
1.29442116, 1.08830777, 1.85642446, 1.69448484, 1.46145593, 2.58869997, 2.79984784,
2.49512218, 2.85747457, 2.88463612, 1.52254367, 2.61946731, 1.32959072, 2.16898013,
1.36253381, 2.88078837, 1.98480707, 2.05060447, 1.21059380, 2.15325594, 1.06494034,
1.75738259, 2.36876592, 2.69220887, 2.61766537, 2.52881006, 1.84979388, 2.74605666,
1.55093918, 1.15307353, 1.47133047, 1.06665322, 2.23700128, 2.16238808, 1.88956753,
1.76681651, 2.64628930, 1.92923795, 2.50785502, 1.14096698, 2.25075899, 1.93263843,
1.91522599, 1.56899485, 2.61987855, 1.25104554, 2.48741614, 1.95777507, 2.84989199,
2.42977949, 2.83310520, 1.03918319, 1.90988919, 1.11747029, 2.14934758, 1.17739636,
1.64551786, 1.88275809, 2.46774809, 1.91891190, 1.12563930, 1.02692407, 2.52712373,
1.38374034, 2.78185962, 1.65500736, 2.81408118, 2.13795387, 1.53278227, 2.13404584,
2.27254866, 1.99286662, 2.84106760, 2.42768273, 2.97361154, 2.88219952, 2.05160350,
1.12881479, 1.26430474, 1.56285315, 2.83571164, 2.67513280, 2.70449263, 1.82965022,
2.61644551, 1.52229404, 1.95981262, 2.71631885, 1.66611186, 1.90911076, 2.36570085,
2.35412140, 2.55002844, 2.57891933, 1.63973004, 1.18199798, 1.76714054, 2.43214790,
1.77538483, 2.48502318, 2.21183611]
```

DO NOT COPY

2.2 – ZTE Racer (Physical Android device) results

Test 1

```
Array Size:  100
Array Contains Elements = 1
Max: 6.95969383
Min: 2.09541962
Total: 480.29008
Mean: 4.80290084
Standard Deviation: 1.38126020
Median: 4.97506995
Results:
[6.18233545, 5.50731796, 3.63191847, 3.09390348, 5.09712317, 3.80842831, 2.18136441,
4.65426600, 5.97306931, 2.09541962, 3.10695440, 3.54683843, 4.18486648, 4.28085270,
6.51813892, 5.63787666, 2.69266176, 4.50841682, 6.67881676, 5.24426536, 2.38014451,
6.31284480, 5.89032520, 6.93004427, 6.84101434, 3.24720467, 5.62454785, 3.36602224,
6.27075440, 6.39309953, 5.63785771, 2.52743040, 4.48833874, 6.35103430, 6.24390754,
4.12708341, 5.08498716, 5.35056739, 5.27573814, 4.05675903, 6.38619087, 6.95969383,
6.32967080, 3.58125455, 3.98410076, 5.27208239, 3.72378398, 4.06116473, 3.98240798,
5.99102565, 3.43879079, 3.55329206, 3.87118463, 5.52234223, 6.25964878, 3.34181684,
5.26402682, 4.74713200, 6.19950914, 4.16627728, 5.40991073, 6.48491875, 5.20564807,
4.62348390, 2.25063527, 6.74296447, 6.86609566, 4.66934428, 3.64638589, 5.53983197,
4.13594495, 5.75482580, 5.69135755, 3.83600149, 3.27034699, 3.55883695, 6.21736944,
4.64387508, 5.82996853, 6.09430897, 4.21951826, 2.65911424, 5.80040967, 6.82534230,
6.16929802, 2.17270057, 6.44473207, 3.85012871, 6.40410006, 5.05555383, 2.41767620,
4.22123646, 2.30477454, 6.70749263, 3.01758585, 5.33423433, 4.97171399, 6.06237512,
4.64142077, 2.87868535]
```

Test 2

```
Array Size:  100
Array Contains Elements = .000000000000001
Max: 6.97543379
Min: 3.02120809
Total: 504.78287
Mean: 5.04782870
Standard Deviation: 1.18723042
Median: 5.13276461
Results:
[5.96954319, 3.55327469, 3.65706947, 3.82275406, 6.28938753, 4.40802287, 6.25970704,
3.21053378, 4.17794241, 4.00718409, 3.61825317, 6.97543379, 3.44182023, 5.59944634,
6.32485496, 6.62356502, 6.82336165, 4.04072985, 3.45818371, 6.71144502, 6.33556233,
6.84269931, 3.90810386, 4.50116418, 3.75466290, 3.81993571, 5.30782866, 5.46059078,
5.42915860, 6.64587710, 4.29034458, 6.76922428, 5.13271815, 6.67392958, 6.80006345,
```

DO NOT COPY

6.00931959, 5.01161015, 5.30184928, 3.25911470, 5.21341507, 6.71051040, 3.52457993, 4.13061291, 6.66342974, 4.79423520, 5.34549111, 3.02120809, 6.96981991, 6.76585340, 6.49952676, 5.42577752, 3.60751955, 6.76085696, 3.60618588, 3.58100743, 5.43264635, 3.53193305, 3.28151372, 5.18670484, 4.37388342, 6.07768460, 6.42670533, 6.04983188, 4.00800633, 4.84232275, 4.51458921, 6.51038702, 5.85121536, 4.78591111, 3.41283919, 5.12737636, 3.87046838, 4.72735149, 4.81590251, 4.64697466, 3.14561501, 4.19733621, 3.36002074, 3.95249752, 3.88051378, 5.79087035, 6.01583112, 5.20476489, 4.75136648, 5.67371589, 6.72166287, 5.40547443, 5.17226039, 5.59795083, 5.52954073, 5.12183670, 4.41979564, 5.47478001, 5.63435677, 3.18448031, 3.30380332, 5.67195410, 4.63426635, 6.54701615, 6.07061240]

<u>Test 3</u>
Array Size: 100
Array Contains Elements = 12345.6789
Max: 10.9990524
Min: 3.08417429
Total: 744.33040
Mean: 7.44330364
Standard Deviation: 2.30797032
Median: 7.57982509
Results:
[7.57369309, 3.43863603, 10.89087360, 8.88988084, 3.89374144, 4.97790855, 6.92943938, 9.85457457, 6.54687500, 7.26071865, 10.73267107, 5.81107506, 10.90307884, 9.41021052, 10.41788824, 10.06426844, 5.69766841, 5.99960089, 3.89113679, 8.97263229, 5.31971651, 6.14491982, 5.42670273, 9.17230675, 9.68326552, 8.52565221, 8.16752535, 9.61743711, 3.55918274, 5.56212942, 8.63255626, 7.73168886, 7.78761232, 10.05386444, 8.91884682, 9.93742271, 10.66791500, 5.69107653, 10.87240005, 10.35950716, 3.08417429, 6.64483751, 9.63864193, 7.64693669, 7.84418638, 5.78577357, 7.11583291, 8.24580561, 5.64701155, 10.55983159, 5.12198476, 7.31193229, 5.24324393, 5.04928537, 6.54564922, 8.04474195, 9.43120503, 7.81919725, 9.67853337, 7.39930849, 8.30010914, 7.00575611, 3.28983414, 7.76969755, 10.36398891, 7.72896715, 4.04964285, 9.85389303, 9.85589453, 8.33225275, 10.93435350, 9.05286903, 5.31712771, 4.98312700, 6.56001134, 6.06639873, 6.49646480, 10.23420531, 8.16569761, 10.91423033, 6.11200140, 4.12217288, 10.40418123, 4.67914167, 4.60535696, 6.19431503, 6.19993628, 8.34896281, 4.60334032, 4.86871956, 3.72662916, 10.99905246, 6.06671611, 5.09958761, 10.87436926, 8.90002244, 10.93804022, 7.16763371, 4.18195483, 3.11729464]

<u>Test 4</u>
Array Size: 100
Array Contains Elements = 999999999.999999999
Max: 9.79225282
Min: 3.11429277
Total: 647.99071
Mean: 6.47990710

DO NOT COPY

Standard Deviation: 2.03415636
Median: 6.62689976
Results:
[7.96523638, 5.09711601, 9.18501462, 7.02260822, 3.30861101, 9.13385946, 6.36894798, 5.45304654, 7.07052484, 7.55490150, 4.77836069, 8.71633795, 3.72064016, 4.10238626, 4.98934628, 7.33667285, 9.79225282, 6.83854661, 3.58554383, 9.40989415, 8.86126958, 4.30501927, 4.75446723, 4.56845067, 8.07378019, 5.05689840, 4.12211028, 3.61448479, 4.40424735, 6.01739421, 7.76990532, 9.00850213, 7.19496297, 7.76588431, 5.34980889, 5.65291289, 4.48854689, 7.78627592, 8.06224481, 8.03327530, 6.89839964, 8.84277681, 7.49010462, 9.34109203, 4.91322854, 8.78361814, 4.38306891, 7.38841499, 4.54009917, 8.09567231, 8.12962427, 8.62847177, 3.56516694, 9.39754289, 5.94538994, 6.62493685, 4.52507368, 5.68915419, 6.83374056, 3.40462054, 3.91744138, 3.63288252, 9.13887582, 3.15596770, 8.55721228, 9.06875625, 9.27646034, 4.58221357, 7.67692428, 5.35879668, 7.55635724, 7.10836704, 6.33025577, 8.56967107, 9.49423367, 3.11429277, 3.83987977, 4.65068452, 6.78010400, 3.17994876, 3.47997969, 9.65062703, 4.67309093, 4.94474770, 9.09644967, 5.90177220, 9.60825247, 5.97732260, 9.17134057, 8.73090814, 8.57812686, 7.92923120, 7.06625946, 3.23666538, 6.25999955, 6.02950749, 5.54242072, 8.37600228, 3.90575934, 5.10243501]

<u>Test 5</u>
Array Size: 300
Array Contains Elements =
98846567400000000000000000000000000000000.12345678934564
Max: 3.98926537 - 3.04338083E9 (~ 3 sec)
Min: 2.01684012
Total: 908.06126
Mean: 3.02687089
Standard Deviation: 0.58493955
Median: 3.03757019
Results:
[2.32735578, 3.56194485, 2.68125035, 3.25071681, 3.33722042, 2.87205762, 3.45435302, 3.53772143, 2.93402339, 3.91765130, 2.03175860, 2.44272232, 2.19974797, 2.50175818, 2.66293607, 3.56848923, 2.03280589, 2.29231683, 3.90916956, 2.15404702, 3.29176015, 3.92883350, 2.38557012, 3.88946204, 3.74996912, 3.60696781, 2.33737672, 3.89126995, 3.12579103, 3.27725709, 3.31060049, 3.32358642, 3.86819814, 3.78042928, 2.09904590, 3.75214805, 3.28232224, 2.54835273, 2.40490595, 2.38162204, 3.58940337, 3.58015559, 3.09705530, 3.84178491, 3.84981139, 3.21310860, 2.09601214, 3.30800810, 2.17742608, 3.98019919, 2.31377196, 3.13131471, 2.41395473, 2.19327975, 3.09639309, 2.21442578, 2.93090658, 3.84373519, 2.97854386, 2.98574176, 3.36297822, 2.68097246, 2.89024459, 3.23003207, 3.12223368, 2.97145642, 2.19719754, 3.27400421, 3.67788973, 2.92300924, 3.27625871, 2.83341150, 2.56730003, 2.27469200, 3.13604391, 3.19889885, 2.74396998, 3.19287089, 2.46943542, 3.21154042, 3.04705463, 2.63763028, 3.62810004, 3.73366265, 3.80519238, 2.19318506, 2.81350771, 2.28386348, 2.98896215, 3.91153770, 2.21385123, 3.58292397, 3.98500941, 3.59935172, 3.95684947, 2.65577329, 2.58550101, 3.03823839,

DO NOT COPY

```
3.57374575, 2.49784751, 2.12813345, 2.57249374, 3.04376797, 2.17308726, 2.48898383,
2.85118854, 2.92124232, 2.93580512, 3.29751276, 3.73752129, 3.91980014, 2.53981950,
3.11741405, 3.89671251, 3.07920233, 3.06629546, 3.00837352, 3.74187283, 3.38641668,
2.24517116, 3.24020986, 3.12961811, 2.90324866, 3.15187751, 2.23187650, 3.86639114,
3.57224689, 3.83431636, 2.35612332, 3.37387925, 2.33252431, 3.19752493, 3.01559897,
3.02933097, 2.32209004, 2.85461531, 3.88086878, 3.82498679, 3.59793235, 3.75909528,
2.97520777, 2.87973110, 2.24628349, 3.27769436, 2.70370974, 2.67646170, 3.58010605,
2.13132787, 3.39746171, 2.93921423, 2.51506271, 2.49326627, 3.56460870, 2.15762638,
3.55589926, 3.27472297, 3.57303336, 2.91571341, 3.45084215, 2.36000099, 2.58205954,
2.99700898, 2.22629579, 3.48764854, 3.55185562, 2.87655705, 3.77436602, 3.52550187,
2.02702078, 2.04206818, 2.27800060, 3.62043156, 3.66551090, 3.49533030, 3.04022671,
3.49037622, 3.42177865, 2.63968526, 3.25649484, 3.91565497, 2.58636395, 2.62228635,
3.57022276, 3.36267162, 3.49352931, 2.82653351, 3.58825653, 3.89026872, 3.91883107,
2.96010118, 2.10445123, 3.76765406, 2.08216393, 2.92157684, 2.99849805, 3.91963307,
2.58559693, 3.15694854, 3.98309217, 3.50709979, 2.85759572, 3.26933016, 3.83970647,
2.69309038, 3.75295463, 3.43745310, 2.13548030, 3.96057308, 2.70405005, 2.66602872,
3.34476606, 2.27662700, 2.97924675, 3.51731341, 2.62939061, 3.94677533, 3.77248627,
2.62700846, 2.05044630, 2.68958548, 2.71310941, 2.04608196, 2.29943298, 3.75351298,
2.17717151, 3.39828008, 3.21909410, 2.10431212, 2.18113153, 2.85883965, 2.86109592,
3.96442320, 2.28136793, 3.81422374, 2.05036843, 3.87361145, 3.46808156, 2.15285827,
3.26131122, 2.63448342, 3.43044521, 3.97822106, 3.69165641, 3.02821800, 2.65071921,
2.48244718, 3.54243200, 2.60278624, 2.04976523, 3.68474251, 3.68663173, 2.52998342,
2.62839084, 3.45415246, 2.44155366, 3.02911211, 2.70659330, 2.43026185, 3.95732112,
3.48264814, 3.22575578, 3.21389418, 2.21333936, 3.03817321, 2.67812137, 3.82672035,
3.10836854, 2.25746335, 3.51273901, 2.42375308, 3.16887052, 2.55079981, 2.14263217,
3.19959992, 2.34589849, 3.02998923, 2.01684012, 2.47567877, 2.13229576, 3.49966569,
2.62034027, 2.49589655, 2.95843082, 3.66042258, 3.29424724, 2.26303775, 2.89168311,
2.38106114, 2.72532361, 2.27582243, 3.88964762, 3.44084393, 3.98926537, 2.27209751,
3.47996368, 3.05227296, 3.71059300, 2.56426272, 2.16372407, 3.36137843]
```

Test 6

Array Size: 500
Array Contains Elements = 0.12345678934564
Max: 2.99838487
Min: 2.00040588
Total: 1252.41449016
Mean: 2.50482898
Standard Deviation: 0.29569446
Median: 2.50139133
Results:

```
[2.98208271, 2.26404823, 2.23826092, 2.77462645, 2.01622896, 2.79119105, 2.81173372,
2.05223338, 2.08443241, 2.97850771, 2.97153147, 2.11047787, 2.62397775, 2.48814590,
2.24937184, 2.66104391, 2.77880842, 2.33381111, 2.49447122, 2.82117664, 2.23802328,
2.55785548, 2.46397201, 2.63957176, 2.48735114, 2.11417726, 2.73081000, 2.45121203,
```

DO NOT COPY

```
2.59386624, 2.91286532, 2.67298997, 2.46970562, 2.82293195, 2.24528109, 2.10251812,
2.49466133, 2.22178782, 2.62356233, 2.85285479, 2.72020861, 2.97431063, 2.12243371,
2.32200576, 2.29728013, 2.54624595, 2.52912534, 2.85652456, 2.00292352, 2.00333552,
2.38931369, 2.25043406, 2.53332058, 2.56901573, 2.60598714, 2.61876972, 2.49469964,
2.93339804, 2.78244981, 2.38779759, 2.53774066, 2.62532067, 2.76601298, 2.52323323,
2.99243862, 2.32785084, 2.91085826, 2.76678116, 2.06110936, 2.09406282, 2.51215786,
2.54100045, 2.52418787, 2.60527966, 2.20043431, 2.03179609, 2.25843117, 2.02352779,
2.64132471, 2.48013017, 2.05128905, 2.79784879, 2.72649306, 2.21401648, 2.46634677,
2.03290324, 2.18099932, 2.91324034, 2.92991860, 2.99786353, 2.03151084, 2.84467405,
2.82904008, 2.85360777, 2.88275328, 2.61683024, 2.10996878, 2.11784950, 2.25878563,
2.34456305, 2.55024916, 2.39054442, 2.21660250, 2.97145896, 2.90294690, 2.73161852,
2.37199699, 2.00669634, 2.29765157, 2.66377102, 2.19218374, 2.16506532, 2.78648839,
2.80446910, 2.48783005, 2.49299739, 2.54670449, 2.49650710, 2.16829870, 2.91485822,
2.04552459, 2.21574469, 2.92005266, 2.59603769, 2.74670073, 2.10341534, 2.87495045,
2.86499663, 2.92073397, 2.42476514, 2.46079713, 2.03223537, 2.52717319, 2.00159480,
2.06081246, 2.85274443, 2.81261472, 2.07138767, 2.60163807, 2.03544822, 2.23068512,
2.84227536, 2.50075202, 2.83020496, 2.94679624, 2.26272151, 2.56043368, 2.86947410,
2.71171825, 2.40673674, 2.60240653, 2.91138334, 2.58406010, 2.01575104, 2.82891194,
2.78103578, 2.55364751, 2.62149433, 2.04367225, 2.65364725, 2.67963367, 2.76342735,
2.64595027, 2.19256152, 2.79037194, 2.25525994, 2.28297909, 2.97891903, 2.95101447,
2.75156323, 2.09902932, 2.92607972, 2.08378005, 2.82700994, 2.25875560, 2.39331821,
2.68979054, 2.04425693, 2.47512050, 2.71452826, 2.02599522, 2.98473632, 2.07600350,
2.65968656, 2.72019177, 2.49624923, 2.88942921, 2.26306012, 2.83171804, 2.44468205,
2.45942639, 2.32261169, 2.59382413, 2.13040541, 2.54252595, 2.98625615, 2.43532648,
2.78313281, 2.78324487, 2.50485665, 2.93650526, 2.93148362, 2.13967433, 2.12356632,
2.81782681, 2.17551678, 2.83222714, 2.13152576, 2.97175447, 2.50036179, 2.69110088,
2.16154614, 2.97617370, 2.60451340, 2.29963739, 2.45781023, 2.89361656, 2.37785946,
2.99838487, 2.90021564, 2.62816155, 2.29388347, 2.78838865, 2.53941863, 2.99145005,
2.84779059, 2.66491105, 2.82497528, 2.91455160, 2.98609876, 2.31006033, 2.03104085,
2.73669959, 2.16190682, 2.44492540, 2.22519752, 2.41858171, 2.07319110, 2.78003621,
2.34697195, 2.82189177, 2.58370953, 2.79826847, 2.83259832, 2.79181717, 2.39104406,
2.43910491, 2.14928183, 2.58398117, 2.62788670, 2.44496435, 2.32541382, 2.26612864,
2.22891777, 2.39331186, 2.63305153, 2.08113531, 2.13241643, 2.98264982, 2.01637320,
2.65347887, 2.26437221, 2.03065123, 2.40815011, 2.19125298, 2.30602118, 2.36923102,
2.60742601, 2.64795915, 2.44083265, 2.54013076, 2.19730457, 2.23002976, 2.24252684,
2.09597813, 2.20881642, 2.66679038, 2.11621210, 2.23095219, 2.45715155, 2.54504000,
2.39542864, 2.65864929, 2.03235645, 2.84567504, 2.23291250, 2.07515188, 2.58865824,
2.75447693, 2.79788058, 2.93061100, 2.35967340, 2.79063819, 2.22149998, 2.62364181,
2.46529182, 2.11747629, 2.99745047, 2.32575510, 2.66017774, 2.07982416, 2.18410045,
2.00415093, 2.87515185, 2.10100069, 2.03411249, 2.07329041, 2.48798448, 2.12889254,
2.45775953, 2.91904989, 2.81145579, 2.91962594, 2.55597001, 2.36432336, 2.70550835,
2.06281518, 2.18443116, 2.54685813, 2.24972059, 2.23898396, 2.88912954, 2.93821252,
2.43833781, 2.82807574, 2.74101143, 2.62978313, 2.81134602, 2.70342737, 2.31339178,
2.03640688, 2.10080301, 2.25277849, 2.19154258, 2.97277573, 2.25956288, 2.97199236,
```

DO NOT COPY

```
2.99121375, 2.82561945, 2.48037087, 2.31966861, 2.55382283, 2.97027166, 2.45231567,
2.98567500, 2.41299950, 2.47919436, 2.56228920, 2.50428353, 2.66590138, 2.77851534,
2.68322838, 2.77428833, 2.09980010, 2.79768036, 2.17881931, 2.49372655, 2.03359169,
2.15796600, 2.08242816, 2.80120394, 2.93286426, 2.22508163, 2.68511776, 2.67795265,
2.91760065, 2.31522794, 2.32457266, 2.87178381, 2.93752588, 2.10933434, 2.59554412,
2.99163782, 2.00040588, 2.33789576, 2.28788424, 2.39878020, 2.77935397, 2.58581931,
2.58001052, 2.56238555, 2.48268311, 2.15103375, 2.52326124, 2.34201899, 2.70143701,
2.54260964, 2.41990119, 2.70134331, 2.90127097, 2.90505679, 2.94351437, 2.43289738,
2.09235071, 2.01353867, 2.27058975, 2.39453031, 2.78033388, 2.97962832, 2.56703028,
2.17207036, 2.76323512, 2.72561700, 2.26703625, 2.05086872, 2.95427329, 2.27724123,
2.39608237, 2.21465769, 2.11085342, 2.19615437, 2.57166103, 2.02042163, 2.65512395,
2.80212921, 2.47055062, 2.91288483, 2.01669848, 2.72012270, 2.73984821, 2.24189282,
2.84151204, 2.28548438, 2.95889462, 2.52102076, 2.11168547, 2.95462676, 2.05078223,
2.57308310, 2.89094606, 2.47679884, 2.18450887, 2.34715490, 2.16834015, 2.98604257,
2.34402055, 2.79386424, 2.20324709, 2.40124664, 2.73947271, 2.48086617, 2.14267896,
2.76998063, 2.59506989, 2.33725629, 2.26092388, 2.33855144, 2.91009478, 2.60511038,
2.44168177, 2.04883097, 2.00613549, 2.07724353, 2.33712666, 2.98587095, 2.43648770,
2.01146963, 2.30205322, 2.53952325, 2.45657224, 2.40201645, 2.16353869, 2.60473372,
2.72842944, 2.75098703, 2.36425849, 2.19487582, 2.58321794, 2.47471153, 2.35500476,
2.65519876, 2.65395104, 2.14493526, 2.43612433, 2.38897880, 2.92505772, 2.63907748,
2.52298132, 2.66785458, 2.03761963, 2.37187604, 2.77619139, 2.73560898, 2.79188183,
2.68637986, 2.69001979, 2.54172290, 2.44276860, 2.47059043, 2.81881425, 2.17339643,
2.70393330, 2.45050156, 2.36552158, 2.97540330, 2.80489418, 2.92235973, 2.23281462,
2.48559060, 2.03127444, 2.25749176]
```

DO NOT COPY

2.3 – Java on PC

Test 1

```
Array Size:  100
Array Contains Elements = 1
Max: 9010.0
Min: 5179.0
Total: 697605.0
Mean: 6976.0
Standard Deviation: 1199.6
Median: 6726.7
Results:
[8991.0, 6246.0, 6947.0, 7470.0, 6048.0, 7631.0, 6652.0, 7303.0, 8885.0, 7894.0, 7338.0,
5363.0, 5887.0, 5901.0, 8710.0, 8963.0, 7652.0, 6491.0, 5612.0, 7311.0, 5769.0, 5658.0, 9010.0,
7927.0, 7521.0, 5447.0, 6072.0, 8529.0, 8087.0, 8685.0, 8269.0, 5501.0, 8077.0, 5185.0, 6566.0,
7226.0, 8899.0, 8157.0, 7202.0, 5351.0, 6124.0, 6348.0, 7126.0, 8744.0, 8160.0, 6726.0, 6275.0,
8625.0, 8711.0, 8944.0, 7497.0, 8613.0, 8811.0, 5292.0, 8757.0, 5418.0, 6263.0, 6045.0, 5842.0,
5549.0, 5757.0, 6501.0, 7812.0, 6079.0, 8713.0, 5568.0, 5975.0, 5204.0, 7657.0, 7075.0, 6662.0,
6799.0, 8538.0, 7517.0, 7874.0, 7069.0, 5779.0, 5523.0, 5572.0, 6482.0, 7993.0, 7305.0, 5371.0,
5737.0, 6724.0, 5213.0, 6574.0, 6135.0, 7208.0, 6153.0, 6254.0, 6410.0, 7798.0, 5425.0, 5179.0,
8085.0, 5990.0, 8194.0, 6481.0, 8917.0]
```

Test 2

```
Array Size:  100
Array Contains Elements = .000000000000001
Max: 15853.0
Min: 5610.0
Total: 1087126.0
Mean: 10871.26
Standard Deviation: 3283.56
Median: 10731.50
Results
[10520.0, 6810.0, 10730.0, 10332.0, 7602.0, 14728.0, 7301.0, 10672.0, 14612.0, 8663.0,
14599.0, 15853.0, 10066.0, 15685.0, 6198.0, 6232.0, 5865.0, 11591.0, 14854.0, 5706.0,
12704.0, 14465.0, 14614.0, 5738.0, 10177.0, 15852.0, 12530.0, 9965.0, 6973.0, 14220.0,
9500.0, 9554.0, 12652.0, 14806.0, 15241.0, 9278.0, 13657.0, 8519.0, 6240.0, 9260.0, 6163.0,
10247.0, 14047.0, 15316.0, 13297.0, 10395.0, 13971.0, 8603.0, 6821.0, 11615.0, 7748.0,
8610.0, 11836.0, 12400.0, 5867.0, 8965.0, 11160.0, 6337.0, 8880.0, 9057.0, 5673.0, 13701.0,
8183.0, 9877.0, 10785.0, 5610.0, 5650.0, 13744.0, 7999.0, 12107.0, 12165.0, 7187.0, 14685.0,
13519.0, 13631.0, 13949.0, 12403.0, 14600.0, 15030.0, 15385.0, 11333.0, 8339.0, 6074.0,
14949.0, 9867.0, 11087.0, 5921.0, 5920.0, 15797.0, 13887.0, 6166.0, 14857.0, 15421.0,
14949.0, 14715.0, 12384.0, 9072.0, 12272.0, 13361.0, 8973.0]
```

DO NOT COPY

Test 3

```
Array Size:  100
Array Contains Elements = 12345.6789
Max: 31663.0
Min: 5692.0
Total: 1056282.0
Mean: 10562.82
Standard Deviation: 3987.14
Median: 9691.53
Results:
[8996.0, 14576.0, 9632.0, 8755.0, 13089.0, 6557.0, 5968.0, 13565.0, 12380.0, 14566.0, 9670.0,
5835.0, 13093.0, 11142.0, 6042.0, 12311.0, 9334.0, 9628.0, 12311.0, 14298.0, 5861.0, 8058.0,
11602.0, 12160.0, 10442.0, 10379.0, 14628.0, 10752.0, 8939.0, 7513.0, 8857.0, 11522.0,
8791.0, 10823.0, 11090.0, 11860.0, 8636.0, 7254.0, 8200.0, 6276.0, 8511.0, 8973.0, 9160.0,
31663.0, 8099.0, 13233.0, 13275.0, 9471.0, 5866.0, 14597.0, 11190.0, 10175.0, 12266.0,
7055.0, 8326.0, 11642.0, 8946.0, 6323.0, 7051.0, 9793.0, 7941.0, 10416.0, 13392.0, 13022.0,
14475.0, 6333.0, 6140.0, 9021.0, 11495.0, 14523.0, 7809.0, 9409.0, 14705.0, 11998.0, 14830.0,
7588.0, 14617.0, 9725.0, 8325.0, 5692.0, 13771.0, 11334.0, 8659.0, 8446.0, 8173.0, 12530.0,
10120.0, 7095.0, 8848.0, 30383.0, 7211.0, 8090.0, 13775.0, 8314.0, 13740.0, 14482.0, 11849.0,
13575.0, 6850.0, 6575.0]
```

Test 4

```
Array Size:  100
Array Contains Elements = 99999999.999999999
Max: 17339.0
Min: 6073.0
Total: 827910.0
Mean: 8279.10
Standard Deviation: 1567.28
Median: 8157.21
Results:
[7693.0, 7763.0, 8475.0, 6073.0, 6417.0, 8980.0, 6818.0, 9103.0, 8412.0, 6768.0, 9918.0,
9745.0, 13359.0, 6733.0, 7494.0, 7523.0, 8000.0, 9101.0, 8495.0, 7197.0, 8183.0, 6668.0,
6574.0, 8949.0, 6945.0, 7591.0, 6906.0, 6290.0, 17339.0, 6907.0, 7804.0, 9069.0, 8304.0,
8982.0, 8762.0, 6153.0, 12379.0, 7895.0, 8992.0, 8982.0, 7613.0, 9005.0, 9772.0, 7495.0,
7550.0, 9736.0, 10100.0, 8147.0, 6996.0, 9448.0, 7795.0, 8420.0, 8282.0, 9658.0, 7914.0,
9238.0, 8502.0, 8385.0, 9475.0, 6109.0, 9148.0, 8666.0, 8140.0, 8785.0, 9864.0, 7780.0, 6886.0,
6510.0, 6610.0, 9517.0, 7029.0, 8335.0, 7961.0, 7832.0, 6911.0, 7952.0, 9557.0, 9187.0, 7207.0,
6570.0, 7951.0, 6694.0, 8194.0, 9123.0, 9315.0, 7423.0, 8716.0, 9606.0, 7094.0, 7159.0, 9228.0,
9750.0, 6528.0, 9030.0, 6838.0, 9335.0, 6776.0, 8665.0, 7246.0, 9411.0]
```

Test 5

```
Array Size:  300
```

DO NOT COPY

```
Array Contains Elements  =
988465674000000000000000000000000000000000.12345678934564
Max: 30910.0
Min: 5897.0
Total: 3073432.0
Mean: 10244.77
Standard Deviation: 6018.24
Median: 8323.26
Results:
[8998.0, 8636.0, 5957.0, 7351.0, 29538.0, 22532.0, 6351.0, 9538.0, 6819.0, 9724.0, 6798.0,
9354.0, 8971.0, 7400.0, 6024.0, 21049.0, 6788.0, 9646.0, 30531.0, 6144.0, 7712.0, 6162.0,
6704.0, 7470.0, 6951.0, 16828.0, 7258.0, 7147.0, 8457.0, 6027.0, 6971.0, 9672.0, 9538.0,
9594.0, 9143.0, 27887.0, 5913.0, 6967.0, 7504.0, 7553.0, 8478.0, 6863.0, 8905.0, 8263.0,
9832.0, 9060.0, 7012.0, 8646.0, 9552.0, 7314.0, 6664.0, 8629.0, 9463.0, 9103.0, 24439.0,
9784.0, 7085.0, 7200.0, 7388.0, 6328.0, 6569.0, 6955.0, 9087.0, 7844.0, 21998.0, 6394.0,
21156.0, 6882.0, 19057.0, 6736.0, 8426.0, 7757.0, 9289.0, 8561.0, 8342.0, 9068.0, 9056.0,
19233.0, 8978.0, 7416.0, 7903.0, 8160.0, 8291.0, 7777.0, 9672.0, 6834.0, 7282.0, 9941.0,
7435.0, 8819.0, 6106.0, 7367.0, 8331.0, 6461.0, 9674.0, 9158.0, 8402.0, 6937.0, 22630.0,
6026.0, 6376.0, 12397.0, 11859.0, 8376.0, 6151.0, 9935.0, 6250.0, 7508.0, 18529.0, 7438.0,
9467.0, 7342.0, 8597.0, 8922.0, 6993.0, 12074.0, 8617.0, 10755.0, 6806.0, 22322.0, 9735.0,
9845.0, 8373.0, 9014.0, 7071.0, 7443.0, 8430.0, 5897.0, 9469.0, 22358.0, 9933.0, 9142.0,
5941.0, 7301.0, 8046.0, 7984.0, 9047.0, 8049.0, 8913.0, 7566.0, 7604.0, 9322.0, 6737.0, 7671.0,
6974.0, 6699.0, 9053.0, 7021.0, 8245.0, 6802.0, 8629.0, 7299.0, 6340.0, 8312.0, 6532.0, 8975.0,
6563.0, 7879.0, 15461.0, 14183.0, 7620.0, 29215.0, 6068.0, 9184.0, 7078.0, 26457.0, 8945.0,
8725.0, 6877.0, 8272.0, 6756.0, 6345.0, 11132.0, 10633.0, 7814.0, 7810.0, 9641.0, 6436.0,
8371.0, 6045.0, 30581.0, 8690.0, 8365.0, 8145.0, 8706.0, 9192.0, 7354.0, 6726.0, 8643.0,
9699.0, 7088.0, 6851.0, 8649.0, 9350.0, 8016.0, 9086.0, 7288.0, 9408.0, 6037.0, 5917.0, 5961.0,
9770.0, 8455.0, 30205.0, 7182.0, 6716.0, 8492.0, 9078.0, 20743.0, 6746.0, 29713.0, 23942.0,
7749.0, 7452.0, 9931.0, 6337.0, 25120.0, 7493.0, 21825.0, 28914.0, 8501.0, 7589.0, 9946.0,
9021.0, 29790.0, 6901.0, 7125.0, 22885.0, 9544.0, 6871.0, 28070.0, 8888.0, 7866.0, 6212.0,
25310.0, 19542.0, 6001.0, 8802.0, 5922.0, 7070.0, 8393.0, 8356.0, 6863.0, 7624.0, 9663.0,
8547.0, 6079.0, 9326.0, 7587.0, 21990.0, 7572.0, 25266.0, 9438.0, 17484.0, 8722.0, 6989.0,
6946.0, 6479.0, 8438.0, 8895.0, 8219.0, 7320.0, 22600.0, 6867.0, 6427.0, 8299.0, 8073.0,
27019.0, 9281.0, 7094.0, 12216.0, 24019.0, 5990.0, 8415.0, 6515.0, 7071.0, 22525.0, 7415.0,
9585.0, 29979.0, 9301.0, 5991.0, 7045.0, 6445.0, 7239.0, 20805.0, 6868.0, 9316.0, 8658.0,
8820.0, 7802.0, 6692.0, 5963.0, 7219.0, 6553.0, 7244.0, 6519.0, 30910.0, 28663.0, 8701.0]
```

Test 6

```
Array Size:  500
Array Contains Elements = 0.12345678934564
Max: 17721.0
Min: 5540.0
Total: 4530485.0
Mean: 9060.97
```

DO NOT COPY

```
Standard Deviation: 2897.12
Median: 8455.32
Results:
[7410.0, 8539.0, 15955.0, 8735.0, 7764.0, 16220.0, 7114.0, 7169.0, 13573.0, 10396.0, 14607.0,
6833.0, 14051.0, 9194.0, 16085.0, 6093.0, 8814.0, 7363.0, 7531.0, 9109.0, 8138.0, 14630.0,
5743.0, 12704.0, 6398.0, 5780.0, 7669.0, 17721.0, 6927.0, 7243.0, 6036.0, 9969.0, 6416.0,
9359.0, 7839.0, 6398.0, 10286.0, 5855.0, 9272.0, 8871.0, 7295.0, 8011.0, 7296.0, 5721.0,
9193.0, 5918.0, 9891.0, 7259.0, 9453.0, 8905.0, 9882.0, 9029.0, 17227.0, 6453.0, 14753.0,
7007.0, 16986.0, 9686.0, 7350.0, 16814.0, 7877.0, 8331.0, 5883.0, 7555.0, 8674.0, 6927.0,
8050.0, 9303.0, 6916.0, 12841.0, 9106.0, 8264.0, 12313.0, 9232.0, 5967.0, 8313.0, 6883.0,
7162.0, 9160.0, 9902.0, 7079.0, 7290.0, 6511.0, 9940.0, 5987.0, 5766.0, 5959.0, 6009.0, 9444.0,
8265.0, 15372.0, 9228.0, 9194.0, 8945.0, 9825.0, 10942.0, 8778.0, 6256.0, 6161.0, 6619.0,
8255.0, 7823.0, 8472.0, 7345.0, 8666.0, 10513.0, 7889.0, 8261.0, 11543.0, 6854.0, 7877.0,
9669.0, 17395.0, 5747.0, 8743.0, 6467.0, 6748.0, 14227.0, 6454.0, 10927.0, 9357.0, 12156.0,
9299.0, 7579.0, 7616.0, 9526.0, 5540.0, 8565.0, 6361.0, 7011.0, 9078.0, 7844.0, 7473.0, 5908.0,
5985.0, 11258.0, 7486.0, 7151.0, 17325.0, 8553.0, 6339.0, 9706.0, 6951.0, 8950.0, 6823.0,
9371.0, 7107.0, 9090.0, 9848.0, 8280.0, 6394.0, 9998.0, 10529.0, 9505.0, 7279.0, 6079.0,
10309.0, 6721.0, 6681.0, 8369.0, 7658.0, 8367.0, 6743.0, 7617.0, 10154.0, 5827.0, 9632.0,
8142.0, 7391.0, 8556.0, 7037.0, 6588.0, 6224.0, 7006.0, 8715.0, 13446.0, 6438.0, 9116.0,
8319.0, 11341.0, 5906.0, 5625.0, 7327.0, 9293.0, 7186.0, 6067.0, 9014.0, 7389.0, 7241.0,
6994.0, 15963.0, 16663.0, 8383.0, 11362.0, 9379.0, 6393.0, 6603.0, 6294.0, 6129.0, 9102.0,
14437.0, 8711.0, 9385.0, 8090.0, 8011.0, 6543.0, 16364.0, 6404.0, 12564.0, 6362.0, 14869.0,
9659.0, 7210.0, 7570.0, 6676.0, 7130.0, 7193.0, 10075.0, 8326.0, 6779.0, 9273.0, 12711.0,
8556.0, 8576.0, 9173.0, 9850.0, 10639.0, 8810.0, 9013.0, 5980.0, 17636.0, 9312.0, 8004.0,
8424.0, 8773.0, 11773.0, 7939.0, 7715.0, 6366.0, 13946.0, 13916.0, 11942.0, 6874.0, 8621.0,
16178.0, 7677.0, 8777.0, 7521.0, 5641.0, 14714.0, 8590.0, 9660.0, 6578.0, 6572.0, 9816.0,
16033.0, 6854.0, 8650.0, 7388.0, 9343.0, 8439.0, 6624.0, 9754.0, 7766.0, 5961.0, 15965.0,
16180.0, 9462.0, 9708.0, 6828.0, 8251.0, 7504.0, 7543.0, 7393.0, 7640.0, 5848.0, 5616.0,
6047.0, 8735.0, 8761.0, 9246.0, 17134.0, 6769.0, 7530.0, 9524.0, 8277.0, 8691.0, 12967.0,
6068.0, 6512.0, 7981.0, 6787.0, 8052.0, 8624.0, 9970.0, 6560.0, 17452.0, 7420.0, 8830.0,
6676.0, 13444.0, 14330.0, 8354.0, 9150.0, 8312.0, 16863.0, 7026.0, 15445.0, 14229.0, 6786.0,
9450.0, 9185.0, 12564.0, 6838.0, 13016.0, 9438.0, 8456.0, 8452.0, 6386.0, 16692.0, 15545.0,
15516.0, 7720.0, 10368.0, 6848.0, 13698.0, 8032.0, 10402.0, 15314.0, 8724.0, 6138.0, 6871.0,
5631.0, 9119.0, 6766.0, 9193.0, 5838.0, 9889.0, 6673.0, 8865.0, 9504.0, 7829.0, 6347.0, 7162.0,
9352.0, 8220.0, 15814.0, 9394.0, 7010.0, 7738.0, 13421.0, 9539.0, 10504.0, 6692.0, 9479.0,
9283.0, 6960.0, 7602.0, 6354.0, 9946.0, 8742.0, 9073.0, 13593.0, 8583.0, 9272.0, 5899.0,
13509.0, 6561.0, 15481.0, 6310.0, 9620.0, 15681.0, 6253.0, 7971.0, 8892.0, 6276.0, 6616.0,
9603.0, 9521.0, 11910.0, 7360.0, 6994.0, 15176.0, 6577.0, 5785.0, 16945.0, 7224.0, 6555.0,
9914.0, 13040.0, 9068.0, 14216.0, 6415.0, 11266.0, 9433.0, 6220.0, 8620.0, 8464.0, 13043.0,
7549.0, 6184.0, 9526.0, 7776.0, 8268.0, 9563.0, 9579.0, 6988.0, 6679.0, 5892.0, 9622.0, 7659.0,
11076.0, 8659.0, 8765.0, 7076.0, 5610.0, 7052.0, 6090.0, 10147.0, 9833.0, 6738.0, 9993.0,
6226.0, 9289.0, 8246.0, 7996.0, 9812.0, 12555.0, 9619.0, 8947.0, 9314.0, 17146.0, 11992.0,
9953.0, 9017.0, 8462.0, 6092.0, 5703.0, 15057.0, 17640.0, 9299.0, 6259.0, 7277.0, 6625.0,
5809.0, 9959.0, 9089.0, 9336.0, 8426.0, 9928.0, 9738.0, 16453.0, 7201.0, 6080.0, 6688.0,
```

DO NOT COPY

```
9468.0, 9263.0, 8412.0, 8965.0, 14572.0, 9439.0, 7030.0, 8565.0, 9450.0, 7820.0, 7366.0,
8253.0, 5686.0, 5789.0, 17011.0, 12353.0, 9336.0, 7505.0, 9545.0, 13242.0, 8031.0, 6532.0,
6970.0, 13455.0, 16781.0, 9189.0, 8291.0, 8139.0, 7891.0, 9319.0, 8778.0, 8146.0, 14104.0,
9285.0, 6477.0, 6474.0, 5836.0, 6499.0, 15908.0, 9367.0, 7358.0, 6127.0, 8613.0, 7201.0,
10351.0]
```

DO NOT COPY