Comparative Analysis of Multi-Threaded and Non-Threaded

Implementations of the Merge Sort Algorithm

(Final Report)

By

Bernice Templeman

Florida Institute of Technology

CIS 5898 – Projects in Computer Information Systems

For

Dr. Stephen Johnson

November 15, 2016

# Table of Contents

Introduction 3

Sorting Decisions 3

Java Threads 3

BTMS Proposition 3

Algorithms / Project Solution 4

Merge-Sort ALgorithm 4

Threads 4

Data Set Size 4

Timing 4

Implementation 5

Platform 5

Development Tools 5

Programming Language and Code Libraries 5

Implementation Procedure 5

Schedule 6

Results 9

Future Enhancements 11

References 12

Appendices 13

# Introduction

Bernice’s Threaded Merge Sort (BTMS) will sort large datasets with Java using the merge sort algorithm on Virtual Machines.

### Sorting Decisions

There are several sorting algorithms to choose from. With an increase in the amount of data being created, implementing the best sort for the data will affect performance. George Heineman says in *Algorithms in a Nutshell* that “Because today’s computers are so much more powerful than the ones of 50 years ago, the size of the data sets being processed is now on the order of terabytes of information. Although you may not be called on to sort such huge data sets, you will likely need to sort large numbers of items [1].” Merge Sort is considered as a stable sort. A stable sort is defined as “One that maintains relative order of "equal" elements. It is important for secondary sorting, e.g. sort by name, then sort again by age, then by salary… [7].” It is also considered as well-suited for sorting data in secondary storage [1].

### Java Threads

To further improve the merge sort performance, multi-threaded sorts are another option. In his report, *An Implementation of Sorting Algorithm Based on Java Multithread Technology*, Wang says

“With the spread of multicore computer, ordinary desktop computers have strongparallel processing ability. But using traditional serial sort algorithm cannot take full advantage of powerful parallel computing power of the computer. Therefore, how to reduce the recursion level and how to improve the traditional algorithms, made him able to adapt to the development of computer parallel technology, to increase the efficiency of traditional algorithms to a new level, is a worthy subject of study [2].”

**Virtual Machines**

Virtual Machines are used in production, development and testing environments. With the increasing Cloud Services, Virtual Machines are also increasing. Per VMware, one of the most popular virtual machine software,

“Virtualization is the process of creating a software-based (or virtual) representation of something rather than a physical one. Virtualization can apply to applications, servers, storage, and networks and is the single most effective way to reduce IT expenses while boosting efficiency and agility for all size businesses [3].”

Another popular Virtual Machine software, Oracle VirtualBox, is shareware. It can be used on different platforms including Windows and Linux. Both VMware and VirtualBox have been reviewed by Inworld and both were found capable [4]. Because VirtualBox is both capable and free and works on the platforms I am testing, I decided to use VirtualBox to satisfy the Virtual Machine project requirement.

## BTMS Proposition

BTMS will test the performance of a non-threaded merge sort and a multi-threaded merge sort java program on Virtual Machines using different data set types, data set sizes and thread counts.

# Algorithms / Project Solution

This section describes the unique problem, the proposed solution that will be known as BTMS and the resources that will be used to implement BTMS.

**The Merge-Sort Algorithm**

Merge-Sort is an algorithm that divides its input list into two lists, sorts them, and merges the two sub-lists into a sorted version of the input list. There are many examples available. The following are notes from Washington University [7]:

**The Basic Algorithm**

MERGESORT(A)

ifLENGTH(A) =< 1

then returnA

B <- MERGESORT(first half of A)

C <- MERGESORT(second half of A)

A <- MERGE(B, C)

returnA

Merge Sort: “Repeatedly divides the data in half, sorts each half, and combines the sorted halves into a sorted whole.

**The algorithm:**

* Divide the list into two roughly equal halves.
* Sort the left half.
* Sort the right half.
* Merge the two sorted halves into one sorted list.
* Often implemented recursively.
* An example of a "divide and conquer" algorithm.
* Invented by John von Neumann in 1945
* Runtime: O(N log N). Somewhat faster for asc/descending input [7].”

As hardware improvements and processors are increases, new algorithms are being tested to optimize the performance of Merge-Sort Algorithms [5]. This project will focus on the basic merge sort algorithm and a threaded version of the merge sort algorithm.

**Threads**

One of the main reasons to use threads is to improve performance. Brian Goetz says in his book, Java Concurrency in Practice, that:

“While the goal may be to improve performance overall, using multiple threads always introduces some performance costs compared to the single threaded approach. These include the overhead associated with coordinating between threads (locking, signaling, and memory synchronization), increased context switching, thread creation and teardown, and scheduling overhead. When threading is employed effectively, these costs are more than made up for by greater throughput, responsiveness, or capacity. On the other hand, a poorly designed concurrent application can perform even worse than a comparable sequential one [6].”

For this reason, BTMS will test a non-threaded merge sort (NT) and test a threaded merge sort. Even though the Non-Threaded Merge Sort is not using the threaded solution, it still uses a Java thread. The one threaded (1T) Threaded Merge Sort uses the threaded solution with only 1 thread. The multi-threaded program will allow the following number of threads to be used for the merge-sort:

* 1 (1T)
* 2 (2T)
* 4 (4T)
* 8 (8T)
* 16 (16T)

**Data Set Size**

Different data set sizes will be created and tested. An array of integers will be created using the dataset type. The following data set sizes will be tested:

* 1000
* 2000
* 4000
* 8,000
* 16,000
* 32,000
* 64,000
* 128,000
* 256,000
* 512,000
* 1,024,000
* 2,048,000
* 4,096,000
* 8,192,000
* 16,384,000
* 32,768,000
* 65,536,000

The Java code doubles the size of the array each loop [7]. Larger data sets were not tested due to hardware limitations.

**Data Set Type**

In my research on existing data sets, I found several that have been tested in other research projects [8]. In this project the following data sets were tested:

* Random: Returns the next pseudorandom, uniformly distributed int value from this random number generator's sequence [9].
* Discrete: *random integer from the specified discrete distribution* [10].
* Uniform: *Returns a random integer uniformly in [0, n)* [10].
* Bernoulli: *random boolean from a Bernoulli distribution with a success*

*probability* [10].

* Gaussian: This data set consists of *random real numbers from a standard Gaussian distribution* [10].
* Zero: This data set consists of only zeros [8].

**Re-Using Exiting Code and Libraries**

The project is a hybrid of the following reusable components:

* Merge Sort [7]
* Threaded Merge Sort [7]
* Random [9]
* StdRandom [10]

**Timing**

The time will be taken before and after the sort is performed using: System.currentTimeMillis();

# Implementation

This section describes the specific resources that will be used to implement BTMS.

## Platform

BTMS will be a Java 8 application. It will be running on an Intel Host platform with a Windows 10 Home Operating System.

The hosted virtual machines will be:

* Oracle Virtual Box VM: Windows 10 VM
* Oracle Virtual Box VM: Ubuntu 16.04.1 LTS

Performance will not be tested on an Ubuntu bootable USB drive because my research shows there are performance costs running Ubuntu on an USB [15].

Information and downloads for Java can be found at: <http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html>

## Development Tools

Development will be done on an Intel machine running Windows 10, a Windows 10 Virtual Machine and an Ubuntu 16.04.1 LTS Virtual Machine. The NetBeans IDE for Java will be installed on all three systems. Information and downloads can be found at <http://www.oracle.com/technetwork/articles/javase/jdk-netbeans-jsp-142931.html>.

Additional NetBeans information can be found at: <https://netbeans.org/>

Oracle VM VirtualBox will be installed on the Windows 10 host.

VirtualBox information and downloads can be found at: <http://www.oracle.com/technetwork/server-storage/virtualbox/downloads/index.html>

**Operating Systems for the Virtual Machines**

* **Free Windows VM:** <https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/>

This is an image you can import.

* **Free Ubuntu iso:** <https://www.ubuntu.com/download/desktop>

This is an ISO, you will need to follow VirtualBox Directions to create a VM and then install the iso.

**Configuration Management**

* Github: <https://github.com/>
* Google drive: <https://www.google.com/drive/>

**Video**

* CamStudio: <http://camstudio.org>

**Anti-Virus on host:**

* McAfee: <http://www.mcafee.com/us/index.html>

**Reports**

* Word: <https://www.microsoftstore.com>
* Excel: <https://www.microsoftstore.com>
* Gantt: <http://www.ganttproject.biz>

**Hardware**

* Host:

|  |
| --- |
| **Model** Samsung - Notebook 7 Spin 2-in-1 15.6" |
| **Platform A** Windows 10 Home |
| **CPU** 2.66GHz x 4, 8 MB Cache, |
| **Processor** Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz, 2592 Mhz, 2 Core(s), 4 Logical Processor(s)  **Model** |
| **Hard Disk** 1TB Hard Drive |
| **RAM** 12 GB |
| **Graphics** NVIDIA GeForce 940MX |

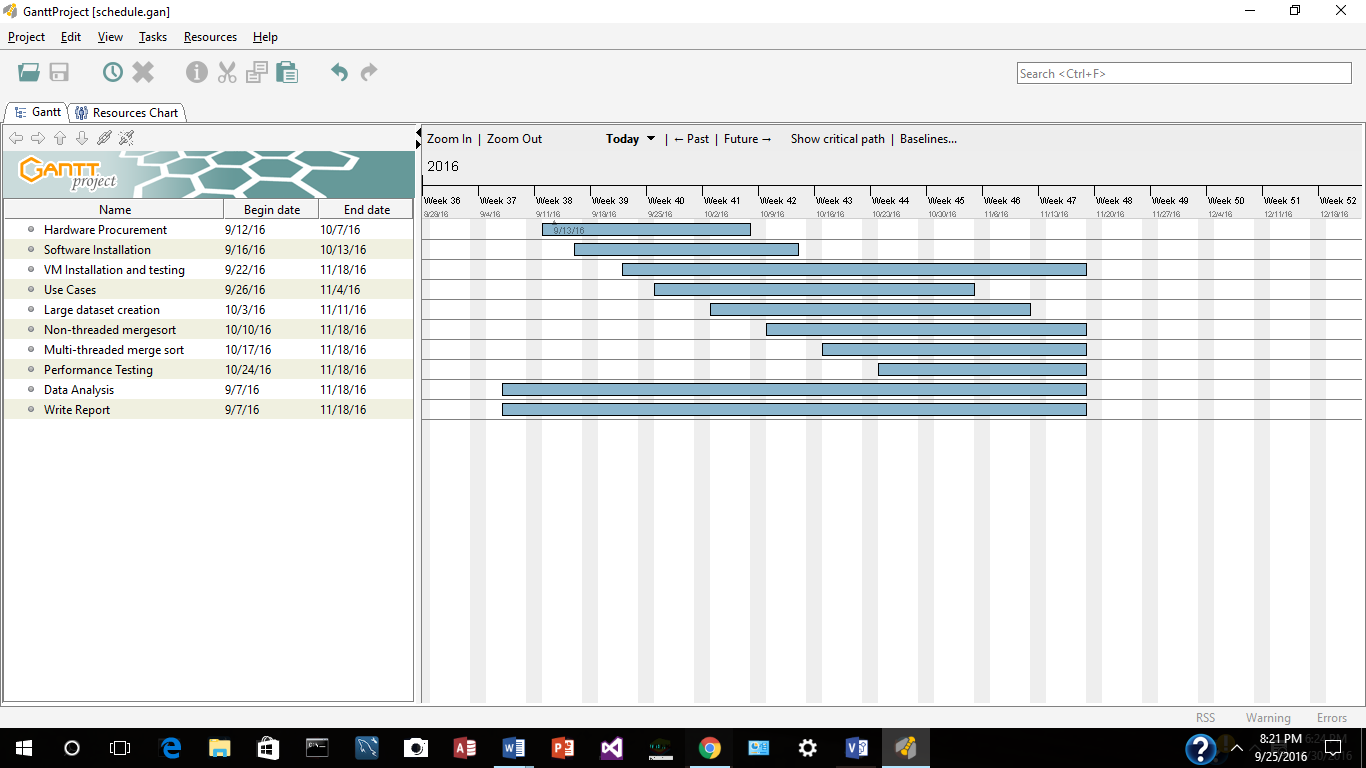
* Virtual Machines:

|  |
| --- |
| **Model** Oracle VirtualBox Virtual Machine |
| **Platform A** Windows Windows 10 |
| **Platform B** Linux Ubuntu 16.04.1 LTS |
| **Processor(s) 1 and 2**  **(tested by changing Virtual Machine setting in “Setting”, “System”, “Processor”)** |
| **Hard Disk** 40GB Hard Drive |
| **RAM** 4 GB |

## Implementation Procedure

I plan to start by procuring a Windows 10 laptop, installing OS updates and installing software. I will also install Oracle VirtualBox, download a Windows 10 VM and create an Ubuntu Linux VM and test them during this phase. Next, I will install Java and the NetBeans IDE on both Windows 10 laptop and on the Windows and Ubuntu VMs. I will research existing datasets, code to produce a dataset or writing a Java program to create a large dataset of integers to be sorted by the merge-sort programs. I will then research non-threaded and threaded merge-sort. I will run performance tests to compare the non-threaded merge-sort with the threaded merge-sort with different number of threads, data set types and data set sizes.

# Schedule



# Results

This section should describe my experience in implementing the solution.

**Hardware and Operating Systems**

After researching project topics and deciding to implement a non-threaded and threaded merge sort algorithm, I first planned to keep my cost low and purchased an inexpensive Windows 10 laptop. When I discovered I needed the project needed to be on virtual machines, I tried testing VirtualBox and a Windows VM on the laptop. I did not have enough memory on the system to run the host and a Virtual Machine. I successfully installed VirtualBox on the host and imported a Windows VM, but the performance was very slow. Because of these performance results, I purchased another laptop with more memory. I installed VirtualBox on the 2nd new laptop and can successfully run both the Windows and Ubuntu VMs. Based on the memory requirements of the Host OS and Virtual Machines, 8 GB of memory may have worked, but I chose 12 GB of RAM. The results for this project are based on this hardware, Virtual Machines and Operating Systems.

**Software**

I researched existing data sets and merge sorts, and combined them to run performance tests on the Virtual machines. I decided to keep the Java application simple and I did not create a Graphical User Interfaces (GUI) because they use threads and may impact performance.

**Data Set Results**

Comparing the averages of the different data sets, the Standard Deviation is low. For this reason, the remainder of the report will use the Random data set results. I have included the charts with the averages of the data sets below.

These are the results of the data set averages:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| No threads:  elements | Random | Discrete | Uniform | Bernoulli | Gaussian | Zero | Standard  Deviation |
| 1000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4000 | 1.5 | 0 | 0 | 0 | 0 | 0 | 0.559017 |
| 8000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16000 | 3 | 0 | 0 | 1.6 | 1.5 | 3.3 | 1.289272 |
| 32000 | 1.5 | 3 | 0 | 3.3 | 0 | 0 | 1.414214 |
| 64000 | 0 | 1.6 | 0 | 3.2 | 1.6 | 4.6 | 1.650926 |
| 128000 | 3.1 | 3.2 | 3.2 | 4.7 | 8.1 | 1.5 | 2.066935 |
| 256000 | 7.9 | 4.9 | 10.9 | 7.8 | 9.4 | 10.9 | 2.083 |
| 512000 | 22 | 18.8 | 20.6 | 19.1 | 17.4 | 19 | 1.458786 |
| 1024000 | 42.3 | 41.8 | 40.2 | 41.1 | 36.1 | 34.2 | 3.042705 |
| 2048000 | 85.6 | 82.8 | 79.7 | 82.9 | 79.4 | 76.6 | 2.928405 |
| 4096000 | 168.8 | 165.8 | 163.8 | 171.9 | 164.3 | 162.5 | 3.226668 |
| 8192000 | 336.1 | 337.4 | 337.4 | 332.8 | 332.7 | 329.6 | 2.868023 |
| 16384000 | 688.9 | 690.9 | 690.6 | 681 | 684.5 | 698.4 | 5.458556 |
| 32768000 | 1361 | 1351.2 | 1351.8 | 1353 | 1362.6 | 1359.7 | 4.656805 |
| 65536000 | 2739 | 2711 | 2715.9 | 2698.4 | 2717.7 | 2734.5 | 13.77721 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 Thread  elements | Random | Discrete | Uniform | Bernoulli | Gaussian | Zero | Standard  Deviation |
| 1000 | 40.3 | 34.1 | 34 | 35.7 | 34.2 | 32.2 | 2.543892 |
| 2000 | 32.6 | 31.3 | 32.9 | 26.5 | 31.1 | 29.7 | 2.145085 |
| 4000 | 28.2 | 28 | 28 | 26.5 | 31.4 | 28.2 | 1.472432 |
| 8000 | 31.3 | 23.4 | 31.4 | 31.4 | 31.1 | 26.4 | 3.140418 |
| 16000 | 27.8 | 34.3 | 31.1 | 26.5 | 33 | 28.1 | 2.866279 |
| 32000 | 26.8 | 27.9 | 29.7 | 31.3 | 31.1 | 26.8 | 1.873203 |
| 64000 | 26.6 | 28.1 | 32.7 | 29.8 | 32.7 | 28 | 2.347161 |
| 128000 | 35.7 | 32.6 | 36 | 31.1 | 32.7 | 37.5 | 2.264705 |
| 256000 | 36 | 38.8 | 40.6 | 37.4 | 42.2 | 36 | 2.30579 |
| 512000 | 48.3 | 51.7 | 53.3 | 54.4 | 57.5 | 51.6 | 2.822528 |
| 1024000 | 77.9 | 76.5 | 71.9 | 78 | 88.8 | 72.2 | 5.605578 |
| 2048000 | 120.4 | 120.3 | 123.2 | 121.6 | 128 | 116.9 | 3.382143 |
| 4096000 | 215.6 | 217.2 | 215.6 | 212.4 | 221.9 | 204.9 | 5.179768 |
| 8192000 | 395.8 | 387.5 | 392.4 | 385.8 | 393.7 | 392.2 | 3.480741 |
| 16384000 | 767.4 | 764.2 | 756.1 | 751.3 | 758.2 | 768.6 | 6.256907 |
| 32768000 | 1429.6 | 1431.3 | 1429.6 | 1436.1 | 1434.5 | 1448.4 | 6.493437 |
| 65536000 | 2815.1 | 2779.7 | 2759.3 | 2760.8 | 2720.2 | 2787.3 | 29.17076 |
| 2 threads:  elements | Random | Discrete | Uniform | Bernoulli | Gaussian | Zero | Standard  Deviation |
| 1000 | 37.6 | 41.8 | 42.2 | 38.5 | 40.2 | 39 | 1.685642 |
| 2000 | 40.4 | 34.4 | 40.4 | 42.2 | 36.1 | 42.1 | 2.969661 |
| 4000 | 40.4 | 42.2 | 39.3 | 37.6 | 35.9 | 37.4 | 2.088859 |
| 8000 | 37.6 | 35.8 | 36 | 32.5 | 36.1 | 34.4 | 1.594783 |
| 16000 | 39.1 | 40.6 | 35.9 | 40.7 | 33.1 | 36.2 | 2.767671 |
| 32000 | 35.8 | 32.8 | 38.9 | 35.8 | 37.5 | 39.2 | 2.181488 |
| 64000 | 37.5 | 40.5 | 34.4 | 39.2 | 40.4 | 35.9 | 2.275534 |
| 128000 | 39 | 37 | 39.1 | 34.2 | 40.4 | 40.7 | 2.226357 |
| 256000 | 46.8 | 42.4 | 38.6 | 39.1 | 43.5 | 40.5 | 2.812719 |
| 512000 | 56.2 | 51.4 | 47 | 51.3 | 51 | 51.7 | 2.66625 |
| 1024000 | 66.9 | 65.8 | 70.1 | 67.2 | 69.9 | 66.9 | 1.616581 |
| 2048000 | 98.4 | 99.7 | 95.2 | 93.7 | 101.5 | 96.8 | 2.64118 |
| 4096000 | 163.6 | 159.8 | 165.4 | 162.3 | 162.6 | 153.3 | 3.893014 |
| 8192000 | 295.2 | 282.6 | 292.1 | 287.5 | 284.4 | 281.5 | 4.993134 |
| 16384000 | 535.7 | 534.5 | 532.7 | 546.9 | 531.3 | 535.9 | 5.065789 |
| 32768000 | 1006.2 | 1027.9 | 1001.7 | 1004.8 | 1009.4 | 999.8 | 9.290138 |
| 65536000 | 1954.7 | 1892.2 | 1859.2 | 1851.8 | 1829.6 | 1934.7 | 45.07818 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 Thread  elements | Random | Discrete | Uniform | Bernoulli | Gaussian | Zero | Standard  Deviation |
| 1000 | 64.1 | 65.5 | 63.9 | 62.5 | 60.7 | 64.2 | 1.519229 |
| 2000 | 67.1 | 62.3 | 64.2 | 60.8 | 56.2 | 62.5 | 3.318341 |
| 4000 | 64.2 | 64.2 | 59.3 | 54.5 | 58.1 | 59.3 | 3.417927 |
| 8000 | 67.2 | 60.8 | 64.2 | 64.4 | 64.2 | 60.8 | 2.236068 |
| 16000 | 63.9 | 62.7 | 56.2 | 59.5 | 60.7 | 61.1 | 2.452493 |
| 32000 | 61.3 | 66.9 | 64.2 | 60.7 | 56.2 | 64.3 | 3.404246 |
| 64000 | 60.8 | 65.8 | 59.3 | 56.2 | 62.4 | 60.8 | 2.913427 |
| 128000 | 65.4 | 67.5 | 61 | 60.9 | 59.1 | 65.7 | 3.068478 |
| 256000 | 68.8 | 70 | 65.4 | 68.7 | 67.3 | 65.4 | 1.74069 |
| 512000 | 73.3 | 73.5 | 74.6 | 68.1 | 74.6 | 72.1 | 2.226357 |
| 1024000 | 86 | 90.6 | 89.1 | 84.4 | 80.8 | 84.2 | 3.252563 |
| 2048000 | 107.5 | 109.2 | 109.1 | 106.3 | 107.2 | 109.5 | 1.195361 |
| 4096000 | 162.5 | 159.3 | 159.1 | 159.4 | 157.4 | 164 | 2.243076 |
| 8192000 | 265.6 | 207.75 | 256.5 | 257.4 | 259.3 | 262.4 | 19.80039 |
| 16384000 | 493.8 | 484.5 | 470.3 | 470 | 468.9 | 482.8 | 9.310821 |
| 32768000 | 867.3 | 867.3 | 864.3 | 850.1 | 856 | 882.4 | 10.1301 |
| 65536000 | 1657.6 | 1645.3 | 1640.3 | 1638.9 | 1589 | 1664.1 | 24.21129 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 threads:  elements | Random | Discrete | Uniform | Bernoulli | Gaussian | Zero | Standard  Deviation |
| 1000 | 121.9 | 120.2 | 119.8 | 121.7 | 118.5 | 123.2 | 1.548566 |
| 2000 | 124.7 | 118.7 | 122.2 | 123.3 | 118.5 | 118.8 | 2.476332 |
| 4000 | 122.2 | 122 | 126.5 | 118.6 | 112.3 | 120.2 | 4.317407 |
| 8000 | 132.5 | 124.9 | 120.4 | 126.9 | 119 | 123.5 | 4.43421 |
| 16000 | 120.3 | 120.4 | 126.4 | 117.3 | 121.8 | 118.7 | 2.870201 |
| 32000 | 125.2 | 123.4 | 118.7 | 120.2 | 117 | 125.2 | 3.180365 |
| 64000 | 117.1 | 121.9 | 115.7 | 117.3 | 118.6 | 120.5 | 2.113778 |
| 128000 | 126.6 | 128.1 | 120.3 | 123.2 | 121.5 | 126.1 | 2.829016 |
| 256000 | 129.6 | 128.1 | 122 | 123.4 | 117.4 | 126.8 | 4.125429 |
| 512000 | 131.2 | 132.6 | 125 | 131 | 124.9 | 126.5 | 3.150485 |
| 1024000 | 140.4 | 138.8 | 138.7 | 144.9 | 145.2 | 139.1 | 2.791306 |
| 2048000 | 159.2 | 164 | 160.9 | 160.7 | 164 | 170.2 | 3.601234 |
| 4096000 | 210.9 | 214 | 209.2 | 206.2 | 204.4 | 202.9 | 3.830434 |
| 8192000 | 314.1 | 309.2 | 312.6 | 306.5 | 304.4 | 307.7 | 3.368193 |
| 16384000 | 512.4 | 513.8 | 548.4 | 515.5 | 508 | 521.6 | 13.35137 |
| 32768000 | 909.3 | 902.9 | 904.5 | 899.9 | 901.1 | 926.5 | 9.064706 |
| 65536000 | 1715.6 | 1695.1 | 1735.7 | 1690.1 | 1670.5 | 1717.2 | 21.24599 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 16 Thread  elements | Random | Discrete | Uniform | Bernoulli | Gaussian | Zero | Standard  Deviation |
| 1000 | 248.2 | 241.9 | 239.1 | 242 | 240.6 | 239 | 3.097849 |
| 2000 | 250 | 248.7 | 239 | 249.9 | 235.9 | 246.9 | 5.554178 |
| 4000 | 256.5 | 249.9 | 248.5 | 247 | 243.6 | 254.8 | 4.425212 |
| 8000 | 248.1 | 253 | 268.7 | 256 | 242.2 | 251.4 | 8.148347 |
| 16000 | 251.6 | 250.2 | 248.5 | 259.5 | 245.1 | 253.4 | 4.456986 |
| 32000 | 254.6 | 251.2 | 251.6 | 254.8 | 239 | 252.9 | 5.397659 |
| 64000 | 262.6 | 248.5 | 251.6 | 250 | 237.6 | 247 | 7.361103 |
| 128000 | 252.8 | 242.4 | 265.6 | 238.9 | 241.9 | 248.4 | 8.978431 |
| 256000 | 245.5 | 245.4 | 249.9 | 246.9 | 245.2 | 257.7 | 4.446597 |
| 512000 | 245.1 | 248.1 | 256.3 | 243.5 | 247 | 259.5 | 5.898705 |
| 1024000 | 262.4 | 259.1 | 250 | 251.5 | 254.2 | 273.4 | 7.929831 |
| 2048000 | 276.1 | 276.5 | 282.8 | 276.6 | 270 | 287.3 | 5.493456 |
| 4096000 | 318.8 | 323.2 | 337.1 | 329.8 | 318.5 | 332.6 | 7.005633 |
| 8192000 | 415.5 | 409.1 | 414.2 | 418.5 | 421.4 | 407.8 | 4.808124 |
| 16384000 | 628.1 | 611 | 618.8 | 621.9 | 640.5 | 616.8 | 9.428812 |
| 32768000 | 1017.3 | 996.9 | 1015.5 | 1029.8 | 1040.8 | 1012.2 | 13.79188 |
| 65536000 | 1800 | 1792.4 | 1798.1 | 1845.6 | 1740.4 | 1820.5 | 31.91562 |

**Operating System Results**

I am a long-time Unix person. I first wrote C and C++ programs on Unix Systems and then also did Unix System Administration as a Software Engineer. Later I also did RedHat Linux System Administration. There were always questions and debates on which OS developers should use and what Operating System should be used in production. From the results of this project, when using threads, developers will need to know when and how to use threads. Adding threads to an application adds another layer of performance tests. Brian Goetz says, “*To have the best chance of finding latent bugs before they occur in production, combine traditional testing techniques with code reviews and automated analysis tools. Each of these techniques finds problems that the others are likely to miss [6 ].”*

When I first ran the merge sort on the different Operating Systems, I was surprised by the results. To see more specific results, I put the averages of the 10 runs into a chart. The results show that on the Virtual Machines with 1 processor, the Windows 10 Non-Threaded Merge Sort was the fastest. The following are the results of the first runs on a Windows 10 and Ubuntu Virtual Machine with 1 Processor (Random Data Set Averages):

**Comparing Merge Sort Average Times on different Operating Systems (ms):**

|  |  |
| --- | --- |
| Windows 10 VM (1 Processor) | Ubuntu VM (1 Processor) |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Array size  izeize | NT | 1T | 2T | 4T | 8T | 16T | NT | 1T | 2T | 4T | 8T | 16T |
| **1,000** | 0 | 14.1 | 26.5 | 50.1 | 104.6 | 214.5 | 0.4 | 9.6 | 8.9 | 17.4 | 29.2 | 68.3 |
| **2,000** | 3.2 | 14.2 | 23.5 | 55 | 98.6 | 213.8 | 2.1 | 9.6 | 8.5 | 18.2 | 31.8 | 68.1 |
| **4,000** | 0 | 17.3 | 31.1 | 52.9 | 112.6 | 219 | 0 | 8.4 | 10.8 | 18.1 | 30.3 | 61.5 |
| **8,000** | 0 | 10.6 | 23.9 | 54.8 | 104.7 | 218.6 | 0.2 | 7.7 | 9.4 | 17.2 | 40.6 | 86.3 |
| **16,000** | 0 | 14.3 | 25.1 | 53.1 | 104.6 | 221.7 | 0.5 | 9.6 | 10.2 | 16 | 40.2 | 78.7 |
| **32,000** | 0 | 14.1 | 26.3 | 57.9 | 112.6 | 217.3 | 1.2 | 9.5 | 9.3 | 17 | 36.1 | 74.3 |
| **64,000** | 1.5 | 15.5 | 34.7 | 53.3 | 113.8 | 210.9 | 2.6 | 11.1 | 10.3 | 18.4 | 62.1 | 156.8 |
| **128,000** | 4.7 | 18.5 | 36 | 62 | 107.9 | 224.9 | 7.1 | 12.1 | 13.6 | 21.2 | 57.7 | 151.9 |
| **25,6000** | 9.5 | 23.4 | 40.6 | 72.3 | 118.9 | 233 | 11 | 27.5 | 19.7 | 27.6 | 62.3 | 160.4 |
| **512,000** | 19.9 | 35 | 50 | 82.4 | 136.2 | 231.2 | 33 | 30.3 | 36.3 | 77.3 | 82.7 | 202.3 |
| **1,024,000** | 43.9 | 56.1 | 70.1 | 103 | 155.9 | 264.2 | 76.9 | 75.3 | 62.2 | 96.1 | 143.5 | 237 |
| **2,048,000** | 81.7 | 98.5 | 123.5 | 149.6 | 201.4 | 295.1 | 187.7 | 142.6 | 187.6 | 166.9 | 292.1 | 373.2 |
| **4096000** | 168.8 | 185.9 | 201.3 | 237.5 | 287.2 | 423.1 | 381.6 | 290.3 | 430.5 | 475 | 501.4 | 561.9 |
| **8192000** | 351.7 | 357.7 | 393.8 | 427.9 | 476.6 | 614 | 902.3 | 869.6 | 946.7 | 890 | 894.2 | 967.6 |
| **16384000** | 693.5 | 696.8 | 736.2 | 765.3 | 815.7 | 1000.2 | 1452.8 | 1592.8 | 1600.5 | 1650.1 | 1614.5 | 1640.8 |
| **32768000** | 1366.8 | 1389.1 | 1398.4 | 1465.8 | 1531.2 | 1660.9 | 2990 | 2992.8 | 3002.3 | 3228.9 | 3010.8 | 3144.2 |
| **65,536,000** | 2673.3 | 2901.4 | 2785.8 | 2777.8 | 3015.4 | 3070.7 | 5346.3 | 5341.1 | 5295.5 | 5489.2 | 5325.7 | 5376.5 |

From these results, the Windows 10 VM threaded merge sort was not faster than the non-threaded merge sort. On the Ubuntu VM, the threaded merge sort performed better on some of the larger data sets.

I then changed the Virtual Machine Configurations to 2 Processors and these are the results from the Random Data Sets (Averages) in ms:

|  |  |
| --- | --- |
| Windows 10 VM (2 Processors) | Ubuntu VM (2 Processors) |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Array size  izeize | NT | 1T | 2T | 4T | 8T | 16T | NT | 1T | 2T | 4T | 8T | 16T |
| **1,000** | 0 | 40.3 | 37.6 | 64.1 | 121.9 | 248.2 | .1 | 23 | 26.8 | 58.9 | 106.9 | 220.2 |
| **2,000** | 0 | 32.6 | 40.4 | 67.1 | 124.7 | 250 | 1.3 | 17.8 | 22.8 | 52.8 | 107.9 | 225.6 |
| **4,000** | 1.5 | 28.2 | 40.4 | 64.2 | 122.2 | 256.5 | 0.1 | 17.9 | 25.8 | 49.5 | 119.8 | 232 |
| **8,000** | 0 | 31.3 | 37.6 | 67.2 | 132.5 | 248.1 | 0.1 | 23.3 | 25.9 | 72.1 | 124.1 | 225.1 |
| **16,000** | 3 | 27.8 | 39.1 | 63.9 | 120.3 | 251.6 | 1.5 | 17.8 | 25.5 | 60 | 118.3 | 206.2 |
| **32,000** | 1.5 | 26.8 | 35.8 | 61.3 | 125.2 | 254.6 | 1.3 | 17.5 | 29.9 | 57.3 | 121 | 228.7 |
| **64,000** | 0 | 26.6 | 37.5 | 60.8 | 117.1 | 262.6 | 2.9 | 19.3 | 33.3 | 72.5 | 127.2 | 236.9 |
| **128,000** | 3.1 | 35.7 | 39 | 65.4 | 126.6 | 252.8 | 5.1 | 29.9 | 37.2 | 76.3 | 121.1 | 287.1 |
| **25,6000** | 7.9 | 36 | 46.8 | 68.8 | 129.6 | 245.5 | 10.9 | 35.5 | 52.5 | 85.1 | 125.5 | 245.2 |
| **512,000** | 22 | 48.3 | 56.2 | 73.3 | 131.2 | 245.1 | 25.1 | 55.7 | 59.6 | 120 | 154.6 | 267.1 |
| **1,024,000** | 42.3 | 77.9 | 66.9 | 86 | 140.4 | 262.4 | 50.9 | 84.5 | 108 | 146.7 | 245.6 | 327.5 |
| **2,048,000** | 85.6 | 120.4 | 98.4 | 107.5 | 159.2 | 276.1 | 112.1 | 159.2 | 185.5 | 212.5 | 280.7 | 391.6 |
| **4096000** | 168.8 | 215.6 | 163.6 | 162.5 | 210.9 | 318.8 | 235.1 | 264.4 | 278.2 | 325.4 | 403.7 | 531.7 |
| **8192000** | 336.1 | 395.8 | 295.2 | 265.6 | 314.1 | 415.5 | 390.8 | 478.6 | 447.1 | 476.8 | 571 | 764.1 |
| **16384000** | 688.9 | 767.4 | 535.7 | 493.8 | 512.4 | 628.1 | 747.5 | 915.8 | 727.8 | 736.9 | 875.9 | 1071.1 |
| **32768000** | 1361 | 1429.6 | 1006.2 | 867.3 | 909.3 | 1017.3 | 1479.6 | 1674.4 | 1324.4 | 1325 | 1403.4 | 1544.6 |
| **65,536,000** | 2739 | 2815.1 | 1954.7 | 1657.6 | 1715.6 | 1800 | 2952.8 | 3255.5 | 2500.2 | 2538.2 | 2575.2 | 2725.4 |

These results show some small differences in the non-threaded merge sort on Windows 1o and Ubuntu. The results also show the Windows 10 threaded merge sort was faster on some of the large data sets. Overall, the Windows 10 performance times were faster than the Ubuntu performance times.

**Comparing Windows 10 VM 1 Processor to 2 Processor**

Next, I compared the result of the Windows 10 VM with 1 processor to the Windows 10 VM with 2 processors.

**Random Data Set Averages (ms)**

|  |  |
| --- | --- |
| Windows 10 VM (1 Processor) | Windows 10 VM (2 Processors) |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Array size  izeize | NT | 1T | 2T | 4T | 8T | 16T | NT | 1T | 2T | 4T | 8T | 16T |
| **1,000** | 0 | 14.1 | 26.5 | 50.1 | 104.6 | 214.5 | 0 | 40.3 | 37.6 | 64.1 | 121.9 | 248.2 |
| **2,000** | 3.2 | 14.2 | 23.5 | 55 | 98.6 | 213.8 | 0 | 32.6 | 40.4 | 67.1 | 124.7 | 250 |
| **4,000** | 0 | 17.3 | 31.1 | 52.9 | 112.6 | 219 | 1.5 | 28.2 | 40.4 | 64.2 | 122.2 | 256.5 |
| **8,000** | 0 | 10.6 | 23.9 | 54.8 | 104.7 | 218.6 | 0 | 31.3 | 37.6 | 67.2 | 132.5 | 248.1 |
| **16,000** | 0 | 14.3 | 25.1 | 53.1 | 104.6 | 221.7 | 3 | 27.8 | 39.1 | 63.9 | 120.3 | 251.6 |
| **32,000** | 0 | 14.1 | 26.3 | 57.9 | 112.6 | 217.3 | 1.5 | 26.8 | 35.8 | 61.3 | 125.2 | 254.6 |
| **64,000** | 1.5 | 15.5 | 34.7 | 53.3 | 113.8 | 210.9 | 0 | 26.6 | 37.5 | 60.8 | 117.1 | 262.6 |
| **128,000** | 4.7 | 18.5 | 36 | 62 | 107.9 | 224.9 | 3.1 | 35.7 | 39 | 65.4 | 126.6 | 252.8 |
| **25,6000** | 9.5 | 23.4 | 40.6 | 72.3 | 118.9 | 233 | 7.9 | 36 | 46.8 | 68.8 | 129.6 | 245.5 |
| **512,000** | 19.9 | 35 | 50 | 82.4 | 136.2 | 231.2 | 22 | 48.3 | 56.2 | 73.3 | 131.2 | 245.1 |
| **1,024,000** | 43.9 | 56.1 | 70.1 | 103 | 155.9 | 264.2 | 42.3 | 77.9 | 66.9 | 86 | 140.4 | 262.4 |
| **2,048,000** | 81.7 | 98.5 | 123.5 | 149.6 | 201.4 | 295.1 | 85.6 | 120.4 | 98.4 | 107.5 | 159.2 | 276.1 |
| **4096000** | 168.8 | 185.9 | 201.3 | 237.5 | 287.2 | 423.1 | 168.8 | 215.6 | 163.6 | 162.5 | 210.9 | 318.8 |
| **8192000** | 351.7 | 357.7 | 393.8 | 427.9 | 476.6 | 614 | 336.1 | 395.8 | 295.2 | 265.6 | 314.1 | 415.5 |
| **16384000** | 693.5 | 696.8 | 736.2 | 765.3 | 815.7 | 1000.2 | 688.9 | 767.4 | 535.7 | 493.8 | 512.4 | 628.1 |
| **32768000** | 1366.8 | 1389.1 | 1398.4 | 1465.8 | 1531.2 | 1660.9 | 1361 | 1429.6 | 1006.2 | 867.3 | 909.3 | 1017.3 |
| **65,536,000** | 2673.3 | 2901.4 | 2785.8 | 2777.8 | 3015.4 | 3070.7 | 2739 | 2815.1 | 1954.7 | 1657.6 | 1715.6 | 1800 |

The results show the non-threaded times are similar on both systems. On the larger data sets, the 2 processor VM has faster performance with the threaded merge sort.

I also compared the Ubuntu VM with 1 processor to the Ubuntu VM with 2 processors.

|  |  |
| --- | --- |
| Ubuntu VM (1 Processor) | Ubuntu VM(2 Processors) |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Array size  izeize | NT | 1T | 2T | 4T | 8T | 16T | NT | 1T | 2T | 4T | 8T | 16T |
| **1,000** | .02 | 8.6 | 10.1 | 19.3 | 39.8 | 71.6 | .1 | 23 | 26.8 | 58.9 | 106.9 | 220.2 |
| **2,000** | 1.9 | 9.3 | 10.2 | 17 | 41.1 | 71.3 | 1.3 | 17.8 | 22.8 | 52.8 | 107.9 | 225.6 |
| **4,000** | 0.1 | 8.5 | 8.6 | 17.7 | 76.8 | 89.9 | 0.1 | 17.9 | 25.8 | 49.5 | 119.8 | 232 |
| **8,000** | 0.2 | 8 | 12.9 | 17.9 | 42.6 | 87.4 | 0.1 | 23.3 | 25.9 | 72.1 | 124.1 | 225.1 |
| **16,000** | 1.8 | 7.3 | 12.2 | 18.3 | 38.8 | 69.9 | 1.5 | 17.8 | 25.5 | 60 | 118.3 | 206.2 |
| **32,000** | 1.1 | 9.7 | 13 | 20.3 | 53.1 | 114 | 1.3 | 17.5 | 29.9 | 57.3 | 121 | 228.7 |
| **64,000** | 2.8 | 11.1 | 14.3 | 20.6 | 59.5 | 146,8 | 2.9 | 19.3 | 33.3 | 72.5 | 127.2 | 236.9 |
| **128,000** | 7.4 | 19.4 | 17.9 | 23 | 49.4 | 169.1 | 5.1 | 29.9 | 37.2 | 76.3 | 121.1 | 287.1 |
| **25,6000** | 17 | 19.8 | 22.3 | 29.3 | 64.2 | 178.7 | 10.9 | 35.5 | 52.5 | 85.1 | 125.5 | 245.2 |
| **512,000** | 46.9 | 34.9 | 36.6 | 73.9 | 114 | 202.7 | 25.1 | 55.7 | 59.6 | 120 | 154.6 | 267.1 |
| **1,024,000** | 88.5 | 74.1 | 73.3 | 72.7 | 195.2 | 290.9 | 50.9 | 84.5 | 108 | 146.7 | 245.6 | 327.5 |
| **2,048,000** | 169.9 | 153.6 | 136.7 | 239.6 | 285.2 | 389.3 | 112.1 | 159.2 | 185.5 | 212.5 | 280.7 | 391.6 |
| **4096000** | 339.9 | 418.9 | 397.6 | 457.5 | 498.1 | 561.3 | 235.1 | 264.4 | 278.2 | 325.4 | 403.7 | 531.7 |
| **8192000** | 817.1 | 793.9 | 847.3 | 851.8 | 894.2 | 561.3 | 390.8 | 478.6 | 447.1 | 476.8 | 571 | 764.1 |
| **16384000** | 1517.2 | 1509.6 | 1544.7 | 1623.4 | 1655.3 | 1708.2 | 747.5 | 915.8 | 727.8 | 736.9 | 875.9 | 1071.1 |
| **32768000** | 2434 | 2945 | 2973.2 | 3107.7 | 2985.4 | 3300.6 | 1479.6 | 1674.4 | 1324.4 | 1325 | 1403.4 | 1544.6 |
| **65,536,000** | 4850.8 | 5356.6 | 5282.8 | 5323.3 | 5225.3 | 5360.9 | 2952.8 | 3255.5 | 2500.2 | 2538.2 | 2575.2 | 2725.4 |

These results also show that on the larger data sets, the multi-threaded merge sort performed faster.

**Comparing Windows 10 Virtual Machine to Windows 10 Host:**

Next I compared the result of the Windows 10 VM with 2 processors to the result of the Windows 10 Host.

|  |  |
| --- | --- |
| Windows 10 Host (NetBeans) | Windows 10 VM (2 Processors) |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Array size  izeize | NT | 1T | 2T | 4T | 8T | 16T | NT | 1T | 2T | 4T | 8T | 16T |
| **1,000** | 0.2 | 15.2 | 20.3 | 30.2 | 54.8 | 108.5 | 0 | 40.3 | 37.6 | 64.1 | 121.9 | 248.2 |
| **2,000** | 0.1 | 10.7 | 19.6 | 32.9 | 56.4 | 111.1 | 0 | 32.6 | 40.4 | 67.1 | 124.7 | 250 |
| **4,000** | 0.5 | 11.9 | 17.9 | 29.9 | 55.3 | 138.2 | 1.5 | 28.2 | 40.4 | 64.2 | 122.2 | 256.5 |
| **8,000** | 0.5 | 11.9 | 18.9 | 29.6 | 58 | 113 | 0 | 31.3 | 37.6 | 67.2 | 132.5 | 248.1 |
| **16,000** | 0.6 | 12.8 | 17.9 | 27.8 | 55.4 | 112.1 | 3 | 27.8 | 39.1 | 63.9 | 120.3 | 251.6 |
| **32,000** | 0.9 | 12.4 | 18 | 29.3 | 54.6 | 121.9 | 1.5 | 26.8 | 35.8 | 61.3 | 125.2 | 254.6 |
| **64,000** | 2.2 | 13.3 | 18.8 | 30.1 | 57.6 | 111.5 | 0 | 26.6 | 37.5 | 60.8 | 117.1 | 262.6 |
| **128,000** | 4.1 | 15.1 | 20 | 31.8 | 58 | 112.7 | 3.1 | 35.7 | 39 | 65.4 | 126.6 | 252.8 |
| **25,6000** | 10 | 25.1 | 24.4 | 31.5 | 58.7 | 113.1 | 7.9 | 36 | 46.8 | 68.8 | 129.6 | 245.5 |
| **512,000** | 17.7 | 44.3 | 34.2 | 33.9 | 63.6 | 118.5 | 22 | 48.3 | 56.2 | 73.3 | 131.2 | 245.1 |
| **1,024,000** | 39.4 | 61.8 | 42.9 | 44.4 | 77.4 | 118.2 | 42.3 | 77.9 | 66.9 | 86 | 140.4 | 262.4 |
| **2,048,000** | 78.4 | 114.7 | 62.3 | 63.7 | 84.6 | 133.6 | 85.6 | 120.4 | 98.4 | 107.5 | 159.2 | 276.1 |
| **4096000** | 155 | 202.2 | 111.4 | 104 | 119.9 | 161.6 | 168.8 | 215.6 | 163.6 | 162.5 | 210.9 | 318.8 |
| **8192000** | 313 | 354.4 | 210.7 | 195.7 | 200.5 | 265.9 | 336.1 | 395.8 | 295.2 | 265.6 | 314.1 | 415.5 |
| **16384000** | 618.9 | 671.9 | 382.1 | 363.2 | 355.1 | 393.6 | 688.9 | 767.4 | 535.7 | 493.8 | 512.4 | 628.1 |
| **32768000** | 1227.8 | 1297.3 | 750.6 | 674.3 | 665.7 | 695.7 | 1361 | 1429.6 | 1006.2 | 867.3 | 909.3 | 1017.3 |
| **65,536,000** | 2481.2 | 2528.2 | 1475.6 | 1286.2 | 1327.4 | 1325.8 | 2739 | 2815.1 | 1954.7 | 1657.6 | 1715.6 | 1800 |

The Host has 2 physical processors but 4 virtual processors. The host also has more memory and disk space. The results show that the multi-threaded merge sort had faster results on more of the larger data sets. Interestingly, a couple of the 8-threaded sorts had faster performance than the 4 threaded sorts.

Comparing Windows 10 Laptop Run via Command line vs NetBeans

Lastly, I compared the Windows 10 Host results from the NetBeans run to the Windows 10 Host results from the Command line run.

|  |  |
| --- | --- |
| Windows 10 Host (Netbeans) | Windows 10 Host (Command line) |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Array size  izeize | NT | 1T | 2T | 4T | 8T | 16T | NT | 1T | 2T | 4T | 8T | 16T |
| **1,000** | 0.2 | 15.2 | 20.3 | 30.2 | 54.8 | 108.5 | 0 | 15.3 | 26 | 32.9 | 65.2 | 121.7 |
| **2,000** | 0.1 | 10.7 | 19.6 | 32.9 | 56.4 | 111.1 | 0 | 9.7 | 20.5 | 33.6 | 71.2 | 120 |
| **4,000** | 0.5 | 11.9 | 17.9 | 29.9 | 55.3 | 138.2 | 0 | 12.7 | 19.3 | 34.2 | 63.4 | 153.9 |
| **8,000** | 0.5 | 11.9 | 18.9 | 29.6 | 58 | 113 | 1.5 | 14.8 | 19.8 | 33.6 | 67.7 | 117 |
| **16,000** | 0.6 | 12.8 | 17.9 | 27.8 | 55.4 | 112.1 | 1.6 | 15.9 | 20.8 | 32.6 | 58.7 | 127.3 |
| **32,000** | 0.9 | 12.4 | 18 | 29.3 | 54.6 | 121.9 | 1.5 | 13.2 | 19.5 | 33.3 | 61.5 | 141.6 |
| **64,000** | 2.2 | 13.3 | 18.8 | 30.1 | 57.6 | 111.5 | 0.3 | 17.5 | 20.6 | 33.6 | 67.8 | 117.4 |
| **128,000** | 4.1 | 15.1 | 20 | 31.8 | 58 | 112.7 | 1.8 | 20.4 | 21.1 | 31.5 | 64.1 | 117.7 |
| **25,6000** | 10 | 25.1 | 24.4 | 31.5 | 58.7 | 113.1 | 13 | 27.5 | 25 | 32.9 | 64.9 | 125.7 |
| **512,000** | 17.7 | 44.3 | 34.2 | 33.9 | 63.6 | 118.5 | 19.5 | 44.4 | 29.2 | 38.5 | 70.5 | 128.1 |
| **1,024,000** | 39.4 | 61.8 | 42.9 | 44.4 | 77.4 | 118.2 | 37.5 | 62.4 | 43.6 | 51.5 | 87.8 | 130.8 |
| **2,048,000** | 78.4 | 114.7 | 62.3 | 63.7 | 84.6 | 133.6 | 76 | 111.1 | **61.9** | 65.6 | 94.1 | 153.2 |
| **4096000** | 155 | 202.2 | 111.4 | 104 | 119.9 | 161.6 | 148.9 | 186.3 | 113.3 | 100.1 | 128.2 | 175.5 |
| **8192000** | 313 | 354.4 | 210.7 | 195.7 | 200.5 | 265.9 | 305.6 | 339.3 | 197.3 | 176.6 | 222.6 | 263 |
| **16384000** | 618.9 | 671.9 | 382.1 | 363.2 | 355.1 | 393.6 | 609.1 | 628 | 384 | 341.3 | 392.6 | 404.8 |
| **32768000** | 1227.8 | 1297.3 | 750.6 | 674.3 | 665.7 | 695.7 | 1219.4 | 1230.6 | 740.9 | 680.8 | 698.7 | 723.2 |
| **65,536,000** | 2481.2 | 2528.2 | 1475.6 | 1286.2 | 1327.4 | 1325.8 | 2453 | 2446 | 1492.3 | 1390.5 | 1324.3 | 1396.5 |

Interestingly, some of the result times in NetBeans are faster than the Command Line version. The Command Line version uses a Java .jar file and in Dr. Rhoda Baggs research report, she states:*“This means that an extra layer of translation was done at runtime [13].”*

From these results, I realize it is important for software developers to know how to use thread correctly and to also test threads. Software developers will need to know the hardware, operating systems and data types and sizes. Dr. Stephen Johnson said:

*“Also keep in mind that performance improvements for multithreading applications are readily seen when a process is required to interact with a slow device.  The idea being that instead of waiting, a thread can perform some work activity.  If there is no interaction with a slow device, a multithreaded application can actually take longer to run (i.e., single thread may be more efficient).”*

Faster hardware and newer Operating Systems may improve performance, but there may also be bugs or performance tuning that needs to be done.

# Future Enhancements

This project looked at the performance of a non-threaded merge-sort and a multi-threaded merge sort on Windows 10 and Ubuntu Virtual Machines. Different data set types and sizes were tested and different thread numbers were tested. Future Enhancements could include:

* External Data Sets [1]
* Merge Sort Enhancements [2]
* Operating System Performance comparison with older OS versions [13]
* Java API Enhancements ( for example: [*ThreadLocalRandom*](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadLocalRandom.html) *[16] )*
* Java thread testing [6]

One area of research is optimizing data-intensive applications in Java 8. In a report on this topic, the authors state:

*“Cloud computing has proven to be an essential concept in developing large-scale applications for several domains of research and industry. Examples of these domains include designing real-time applications for embedded systems, reputation-based web services, data aggregation in wireless networks, high-performance computing, and delivering cloud services at several levels through SaaS or PaaS. However, distributed characteristics such as scalability, performance, and replication over a large number of resources can prove to be a very complex effort. When it comes to designing applications, the Java language is one of the mainstream object-oriented programming languages in terms of usability and ease of learning. Java has both a concurrent model and remote method invocation standard, and both require following a certain pattern in programming that gives the user little flexibility in terms of designing a loosely coupled parallel and distributed application [17].”*

For future-work, I want to continue using Java threads and test external data sets with non-threaded and multi-threaded merge-sorts.

# References

[1] Heineman, George T., et al. *Algorithms in a Nutshell*, O’Reilly Media, Sebastopol, CA, 2016.

[2] Wang, Deming, et al*. An Implementation of Sorting Algorithm Based on Java Multithread* Optimized Merge Sort on Modern Commodity Multi-Core CPUs."

*Technology*, vol. 1, IEEE, 2012.doi:10.1109/ICCSEE.2012.152.

[3] VMware. <http://www.vmware.com/solutions/virtualization.html>

[4] Infoworld. <http://www.infoworld.com/article/2615128/virtualization/review--vmware-workstation-9-vs--virtualbox-4-2.html>

[5] Xu, Ming, et al. " *TELKOMNIKA (Telecommunication Computing Electronics and Control),* vol. 14, no. 1, 2016., pp. 309doi:10.12928/telkomnika.v14i1.2741.

[6] Goetz, Brian, et al. *Java Concurrency in Practice.*  Addison-Wesley Professional. 2006.

[7] merge sort code is based on code from: <https://courses.cs.washington.edu/courses/cse373/13wi/>

[8] Li Xiao, Xiaodong Zhang, and Stefan A. Kubricht. 2000. Improving memory performance of sorting algorithms. J. Exp. Algorithmics 5, Article 3 (December 2000). DOI=http://dx.doi.org/10.1145/351827.384245

[9] Random. <http://docs.oracle.com/javase/8/docs/api/java/util/Random.html>

[10] StdRandom.java <http://introcs.cs.princeton.edu/java/stdlib/javadoc/StdRandom.html>

[11] McDowell, Gayle L., *Cracking the Coding Interview*, CareerCup, LLC., Palo Alto, CA, 2016.

[12] Mahafzah, Basel A. "Performance Assessment of Multithreaded Quicksort Algorithm on Simultaneous Multithreaded Architecture." *The Journal of Supercomputing,* vol. 66, no. 1, 2013., pp. 339-363doi:10.1007/s11227-013-0910-2.

[13] Baggs, Rhoda., Delgado, H., Bakkhtiani, Patrick., “Performance Analysis of I/O-Intensive & CPU-Intensive Benchmarks on Windows 7, 8.1 & Ubuntu Linux 14.04LTS”, 2015 Proceedings of the Information Education Conference, Orlando, Florida, May 2015. 2015.

[14] Deitel, P. J., Deitel, H. M., & Safari Books Online (Firm). (2015). *Java* (10th ed.). Upper Saddle River, N.J: Pearson.

[15] <http://askubuntu.com/questions/21741/performance-cost-of-running-ubuntu-from-external-hard-drive>

[16] https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadLocalRandom.html

[17] Serbanescu, V., Azadbakht, K., Boer, F., Nagarajagowda, C., & Nobakht, B. (2016). A design pattern for optimizations in data intensive applications using ABS and JAVA 8.*Concurrency and Computation: Practice and Experience, 28*(2), 374-385. doi:10.1002/cpe.3480

# Appendices

**APPENDIX A**

**Status Reports**

**A.1** Capstone Project Week 5 Status Report for “Comparative Analysis of Multi-Threaded and Non-Threaded Implementations of the Merge Sort Algorithm”

**Date:** 25-September-2016

**Accomplishments**

Activity 1: Hardware and software Installation and testing (22-August-2009 - 25-September-2016)

1. Hardware Procurement – completed for existing hardware. On-going for Virtual Machine requirement.
2. Software Installation – completed for existing hardware. On-going for Virtual Machine requirement.
3. VM installation and testing – completed on existing hardware.

**Current Activities (26-Sept-2016 – 03-Oct-2016)**

Activity 2: Virtual Machine hardware and software procurement and testing and Use Cases

1. Hardware requirements research for Virtual Machines, possible additional hardware procurement and testing.
2. Use Cases

**Challenges**

The challenges are:

1. Additional hardware requirements for Virtual Machines
2. How to obtain or produce a valid data set

The challenges listed above can be resolved and do not place the project at risk.

**Work to be Completed by Oct. 9, 2016**

1. Hardware for Virtual Machine because it is a Capstone Requirement
2. Use Cases
3. Data Set research

**A.2** Capstone Project Week 7 Status Report for “Comparative Analysis of Multi-Threaded and Non-Threaded Implementations of the Merge Sort Algorithm”

**Date:** 03-October-2016

**Accomplishments**

Activity 1: Hardware and software Installation and testing (25-September-2009 - 03-October-2016)

1. Hardware Procurement – completed for existing hardware and for Virtual Machine requirement.
2. Software Installation – completed for existing hardware and for Virtual Machine requirement.
3. VM installation and testing – completed on existing hardware.

 Activity 2: Data Set Research

1. Data Set Research – on-going. Downloaded code from *”Algorithms in a Nutshell”* and researching datasets through the FIT Library.
2. One research paper used several different datasets.

* Random
* Function to return Integers in a range
* Bernoulli function to return 0 or 1
* Geometric function
* Pascal function
* Binomial function
* Zero function
* Unbalanced function

**Current Activities (03-Oct-2016 – 09-Oct-2016)**

Activity 2: Data Set Research and Use Cases

1. Data Set Research and testing code for MergeSort from “*Algorithms in a Nutshell”*
2. Use Cases

**Challenges**

The challenges are:

1. How to obtain or produce a valid data set

The challenges listed above can be resolved and do not place the project at risk.

**Work to be Completed by Oct. 9, 2016**

1. Data Set research
2. Use Cases

**A.3 Capstone Project Week 9 Status Report for** **“Comparative Analysis of Multi-Threaded and Non-Threaded Implementations** **of the Merge Sort Algorithm”**

**Date:** 23-October-2016

**Accomplishments**

Activity 1: Hardware Procurement – complete

Activity 2: Software Installation – complete

Activity 3: VM installation and testing – complete

Activity 4: Uses Cases – in progress

Activity 5: Large Data Set creation – in progress

Activity 6: Non-threaded Merge Sort – in progress

Activity 7: Multi-threaded Merge sort – in progress

Activity 8: Performance Testing – in progress

Activity 9: Data Analysis – in progress

Activity 10: Write report – in progress

**Current Activities (23-Oct-2016 – 06-Nov-2016)**

Activity 4: Uses Cases – in progress

* I have a model for a Use case on paper. My laptop with Visio is at another location. Plan to complete on software this week.

Activity 5: Large Data Set creation – in progress

* Testing existing code for large data set creation
* Coding to create additional data sets based on research results

Activity 6: Non-threaded Merge Sort – in progress

* Testing some existing code
* Researching other merge sort code

Activity 7: Multi-threaded Merge sort – in progress

* Testing some existing code
* Researching other merge sort code

Activity 8: Performance Testing – in progress

Activity 9: Data Analysis – in progress

Activity 10: Write report – in progress

**Challenges**

The challenges are:

1. I am currently testing on my Windows OS on my laptop. I need to Finalizing the code for the data set creation and merge sort algorithm then move to virtual machines and test on the Windows VM and Linux VM.

The challenges listed above can be resolved and do not place the project at risk.

**Work to be Completed by Nov. 6, 2016**

Activity 4: Use Cases - in progress

Activity 5: Large Data Set creation – in progress

Activity 6: Non-threaded Merge Sort – in progress

Activity 7: Multi-threaded Merge sort – in progress

Activity 8: Performance Testing – in progress

Activity 9: Data Analysis – in progress

Activity 10: Write report – in progress

A4. Capstone Project Week 11 Status Report for “Comparative Analysis of Multi-Threaded and Non-Threaded Implementations of the Merge Sort Algorithm”

**Date:** 05-November-2016

**Accomplishments**

Activity 5: Large Data Set creation – in progress

Data Sets I am currently using for testing the performance of my project are:

* Random ( from [https://docs.oracle.com/javase/8/docs/api/java/util/Random.html (Links to an external site.)](https://docs.oracle.com/javase/8/docs/api/java/util/Random.html) )
* StdRandom.java ( from [http://introcs.cs.princeton.edu/java/stdlib (Links to an external site.)](http://introcs.cs.princeton.edu/java/stdlib) )
  + Bernoulli,
  + uniform,
  + Gaussian,
  + Discrete
* Not Tested yet:
  + Zero function (all zeros)

Data Set Size

* Testing sizes of data sets to run on my hardware.
* Have not tested external storage since this will need to run on Virtual Machines

Activity 6: Non-threaded Merge Sort – in progress

* Testing some existing code
* Researching other merge sort code

Activity 7: Multi-threaded Merge sort – in progress

* Testing some existing code
* Researching other merge sort code

Activity 8: Performance Testing – in progress

Activity 9: Data Analysis – in progress

Activity 10: Write report – in progress

Activity 11: Configuration Management

* I have configured GitHub on my laptop to sync with an external site

**Current Activities (05-Nov-2016 – 20-Nov-2016)**

Activity 4: Uses Cases – in progress

Activity 5: Large Data Set creation – in progress

Activity 6: Non-threaded Merge Sort – in progress

Activity 7: Multi-threaded Merge sort – in progress

* Need to test different number of threads

Activity 8: Performance Testing – in progress

Activity 9: Data Analysis – in progress

Activity 10: Write report – in progress

Activity 12: Video  - not tested or completed.

**Challenges**

The challenges are:

1. Added configuration management to the plan for security purposes.

The challenges listed above can be resolved and do not place the project at risk.

**Work to be Completed by Nov. 20, 2016**

Activity 5: Large Data Set creation – in progress

Activity 6: Non-threaded Merge Sort – in progress

Activity 7: Multi-threaded Merge sort – in progress

Activity 8: Performance Testing – in progress

Activity 9: Data Analysis – in progress

Activity 10: Write report – in progress

Activity 12: Video

**APPENDIX B**

**Software Configuration**

This section provides all steps necessary to install and configure application to run.

**B.1 Steps to install BTMS on a Windows 10 Oracle VirtualBox host**

\*\* Note: For this project demonstration, BTMS is currently available on the Virtual Machines at: <https://drive.google.com/open?id=0B98YurnDqxH0S19Db1c4OHMxZXM>

For manual installation, please follow the instructions below:

**1. Virtual Machine Host:**

* Install Oracle VM Virtual Box: <http://www.oracle.com/technetwork/server-storage/virtualbox/downloads/index.html>

VMs were exported from: Oracle VM VirtualBox Version 5.1.8 r111374 (Qt5.5.1)

* Install Oracle Virtual Box Extension Pack: <http://www.oracle.com/technetwork/server-storage/virtualbox/downloads/index.html>

**2. Software:**

* Download and Install Java: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

**Oracle VM Host currently at:**

java version "1.8.0\_102"

Java(TM) SE Runtime Environment (build 1.8.0\_102-b14)

Java HotSpot(TM) 64-Bit Server VM (build 25.102-b14, mixed mode)

* Download and Install NetBeans: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

**Oracle VM Host currently at:**

**Product Version:** NetBeans IDE 8.1 (Build 201510222201)

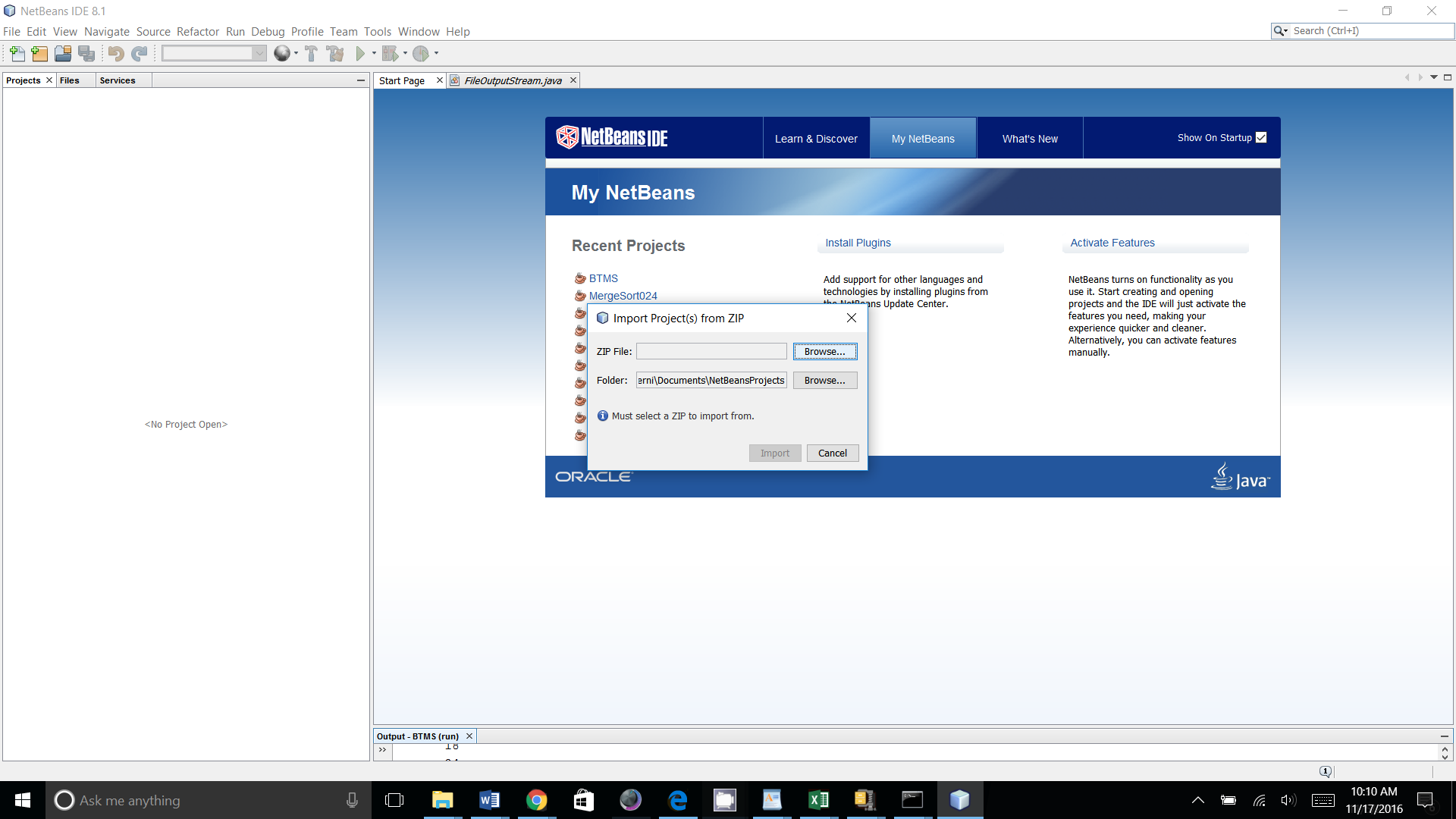
**Updates:** [Updates available](about:blankcheck-for-updates) to version [NetBeans 8.1 Patch 1](http://wiki.netbeans.org/NetBeans8.1PatchesInfo)

**Java:** 1.8.0\_102; Java HotSpot(TM) 64-Bit Server VM 25.102-b14

**Runtime:** Java(TM) SE Runtime Environment 1.8.0\_102-b14

**System:** Windows 10 version 10.0 running on amd64; Cp1252; en\_US (nb)

* Install NetBeans Updates.
* Run NetBeans
* Import or Unzip the BTMS.zip file
  + **To Download and Import project from zip file :**
    - In NetBeans, Under the “File” menu, select “Import Project” then “From ZIP…”
    - Browse to the BTMS zipfile, and select “Open”
      * note: this gives an error that no project was imported, but when you select “Open”, the project is there and can be opened and ran.



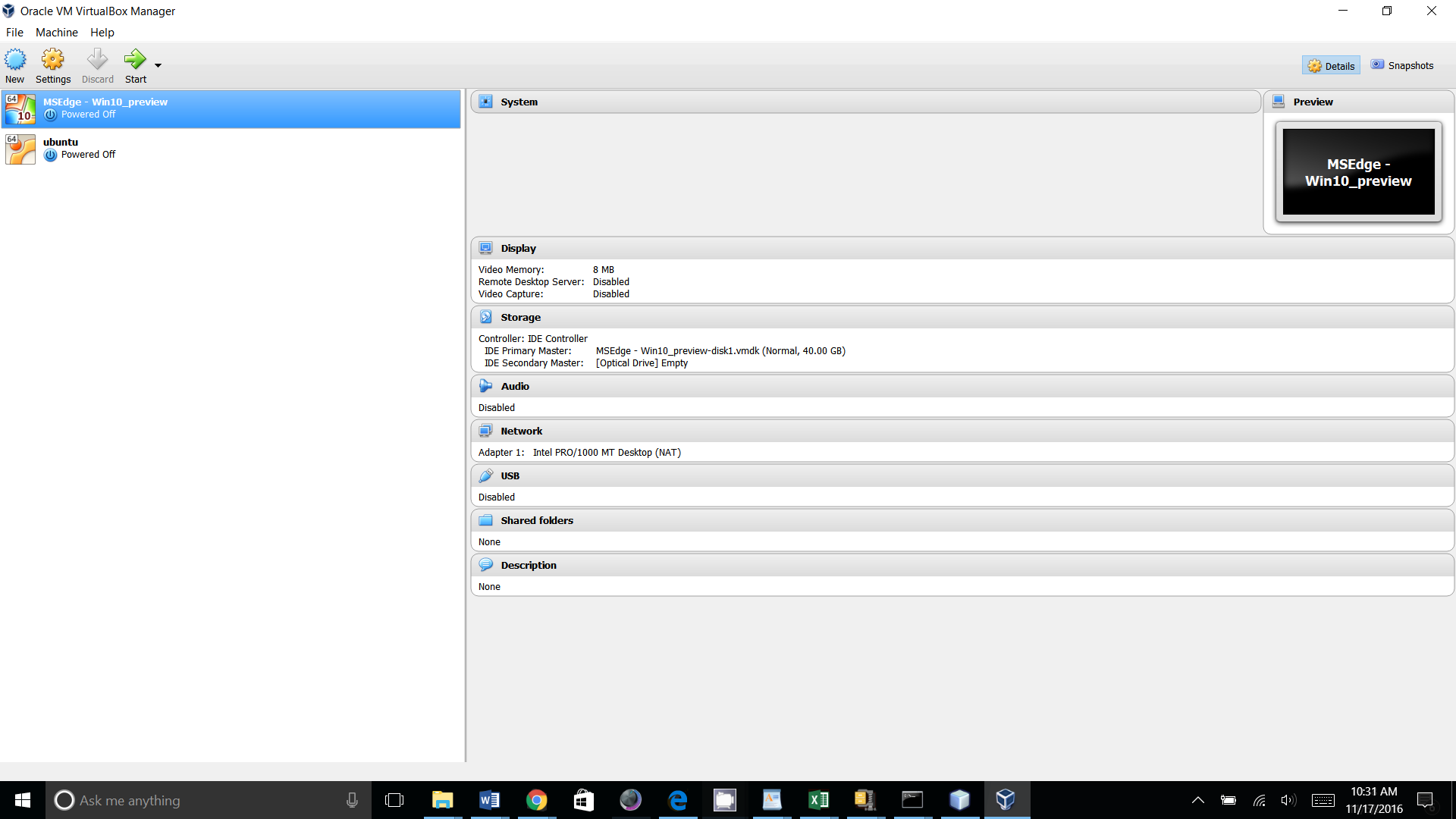
* + **To Unzip the file then open in NetBeans :**

* Unzip the "BTMS.zip" file into the NetBeans Projects directory  
  Note: Be sure to uncompress it into the "NetBeansProjects" folder.  By default the extraction tool will uncompress a file into a folder with the same name as the compressed file (i.e., "BTMS" in this case)
* In NetBeans, go to “File”, “Open Project”,”Look In:” NetBeansProjects directory and select BTMS, then “Open Project”

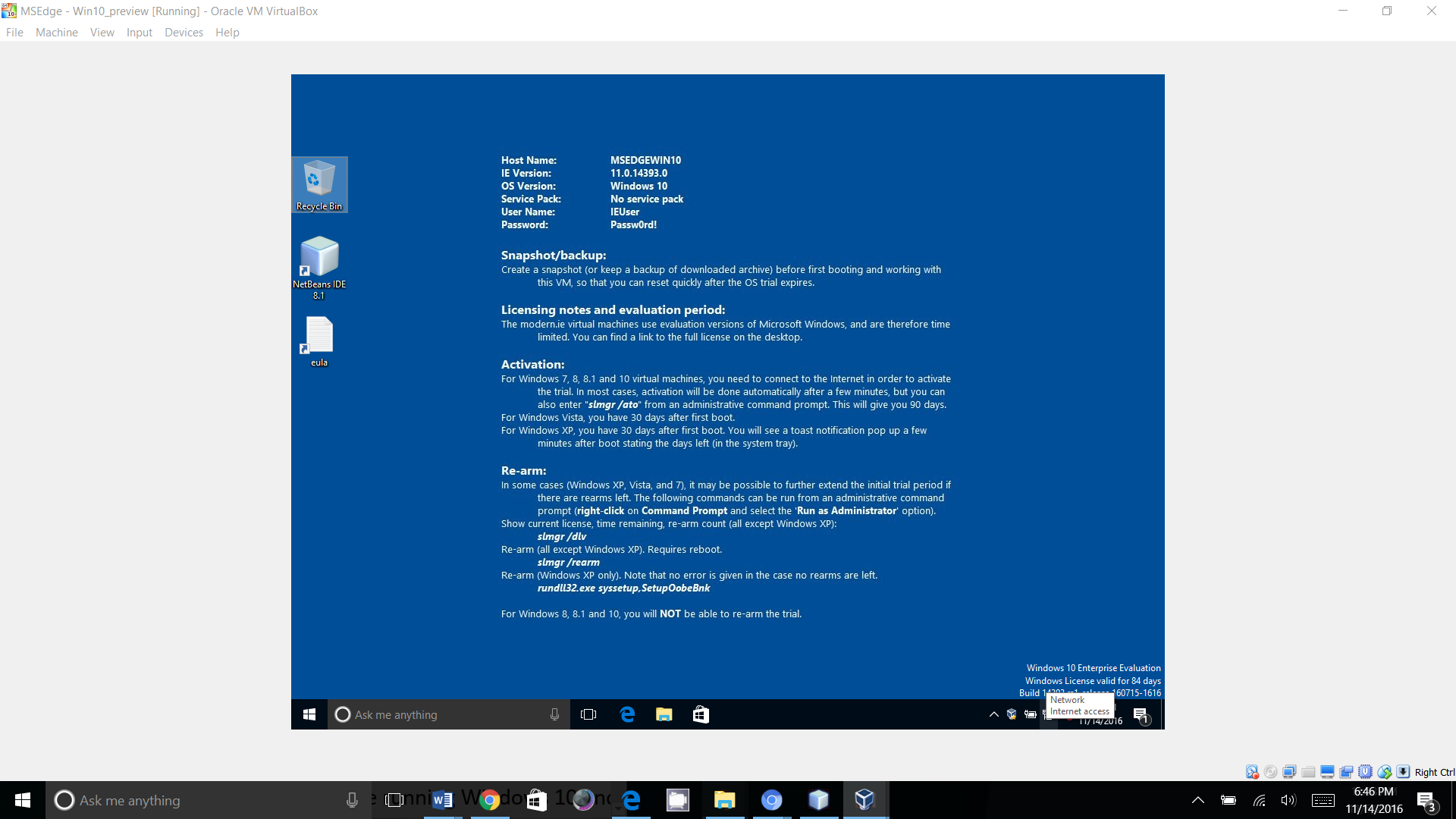
**B.2 Steps to install BTMS on Virtual Machines**

**Virtual Machines:**

* Import the Virtual Machine Images into the host: <http://docs.oracle.com/cd/E26217_01/E26796/html/qs-import-vm.html>



* Select the machine the “Start”
* **Windows:**
  + **User Name: IEUser**
  + **Password: Passw0rd!**

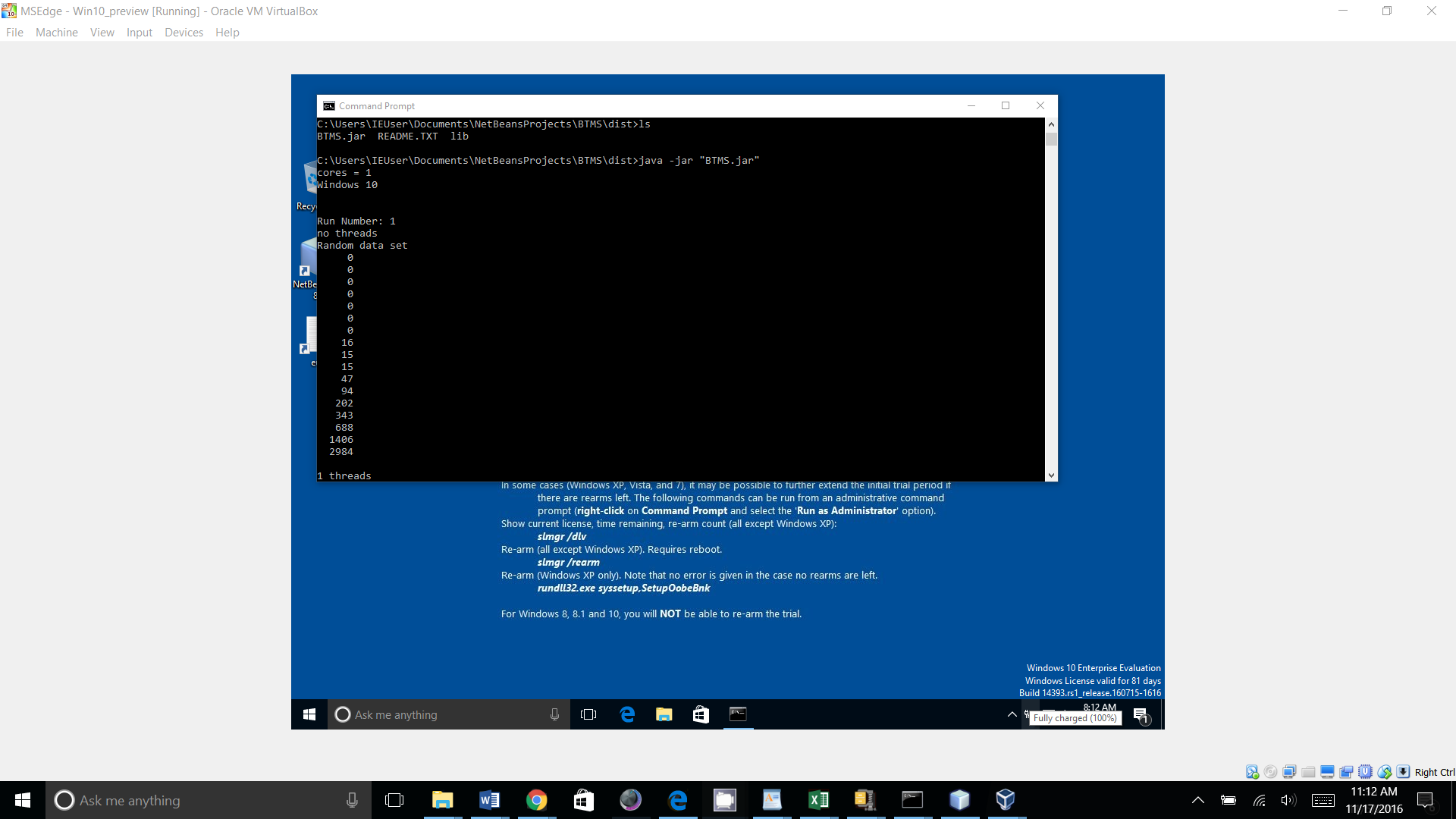


* **Ubuntu:**
  + **User Name: ubuntuVMuser**
  + **Password: Passw0rd!**

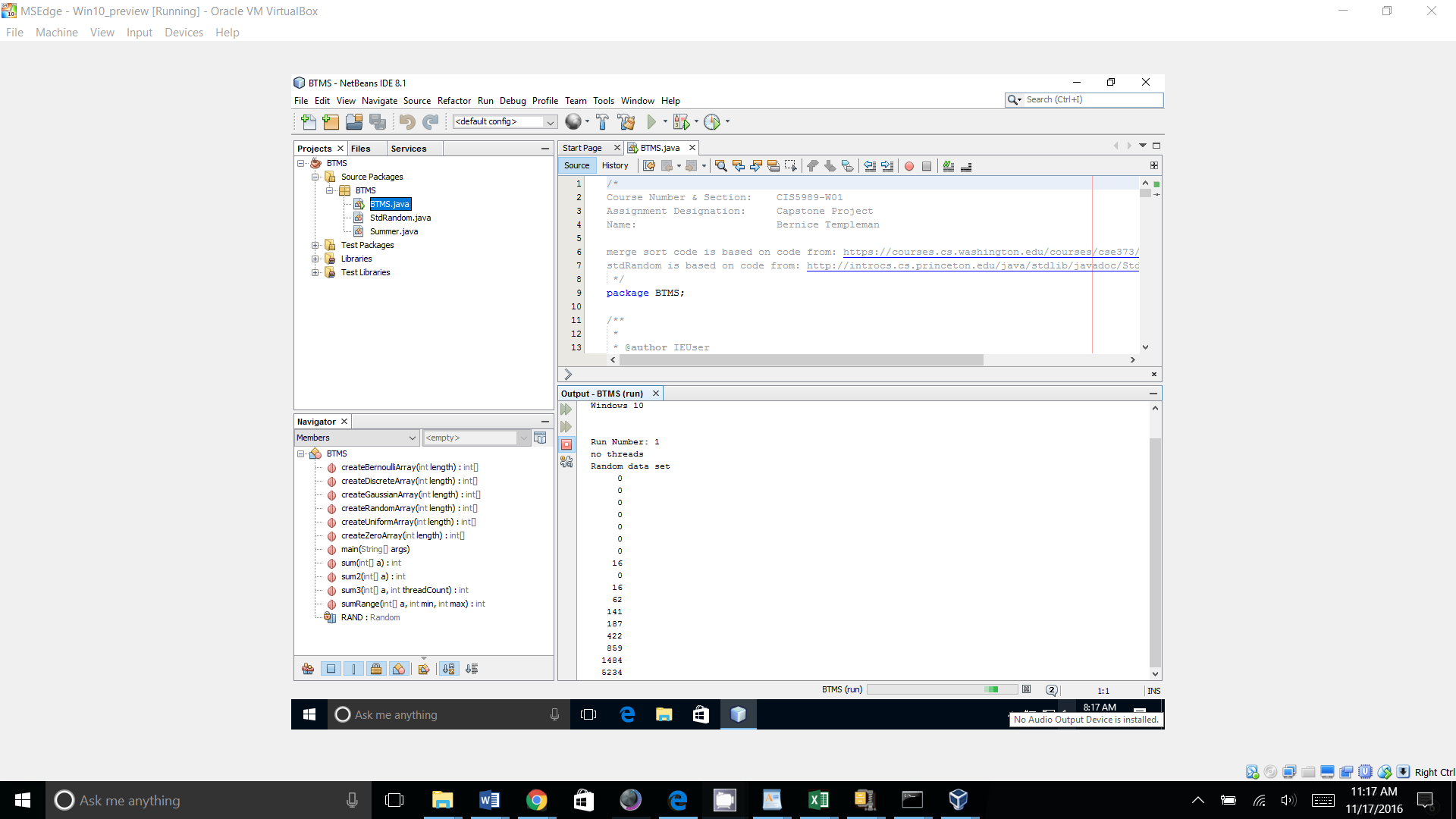


**Windows:**

**1. Command Prompt:**

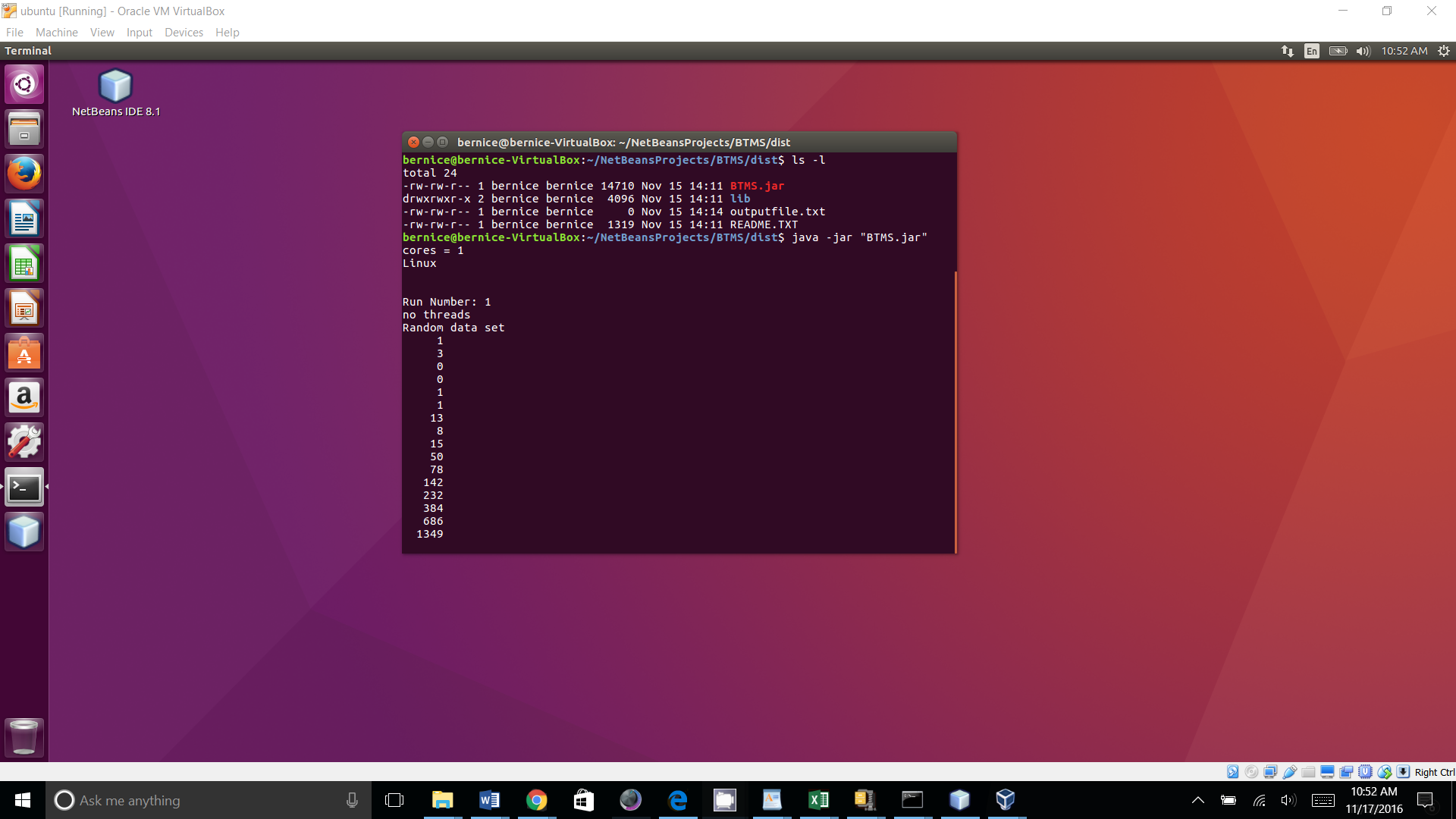


2. NetBeans

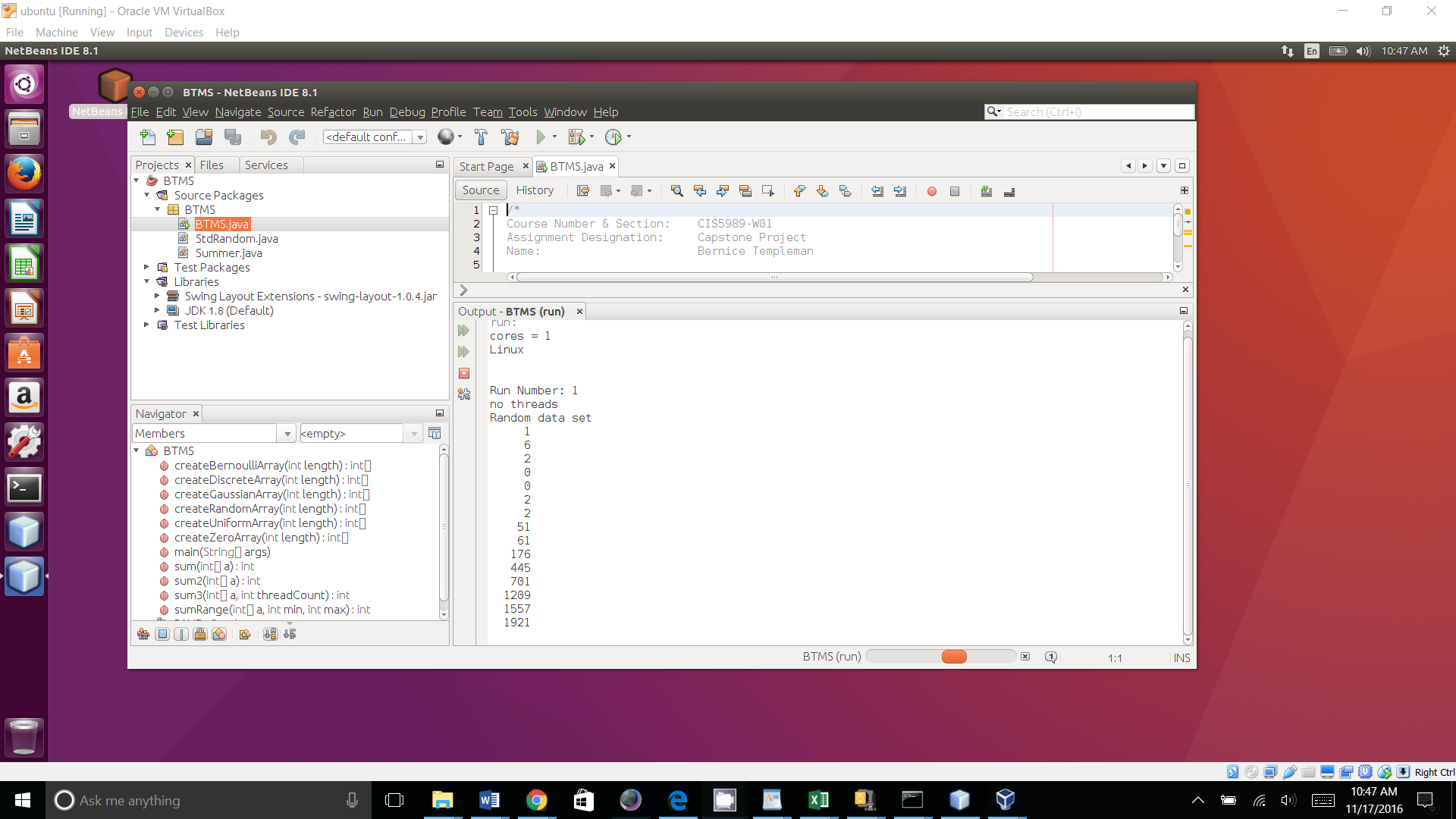


**Ubuntu**

1. Ubuntu “Terminal”



2. Ubuntu “NetBeans”



**B3: Steps to Create new Virtual Machines**

**Virtual Machines:**

* **Follow directions to create virtual machines and install Operating Systems:**

<http://docs.oracle.com/cd/E26217_01/E26796/html/qs-create-vm.html>

* **Free Windows VM:** <https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/>

This is an image you can import.

* **Free Ubuntu iso:** <https://www.ubuntu.com/download/desktop>

This is an ISO, you will need to follow VirtualBox Directions to create a VM and then install the iso.

**Software:**

Login to the Virtual Machines, Download and install Java, NetBeans and BTMS as described in B1 (follow OS appropriate installation Guides for Java and NetBeans.)

• Download and Install Java: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

• Download and Install NetBeans: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Then Import or Unzip and Open the BTMS file.

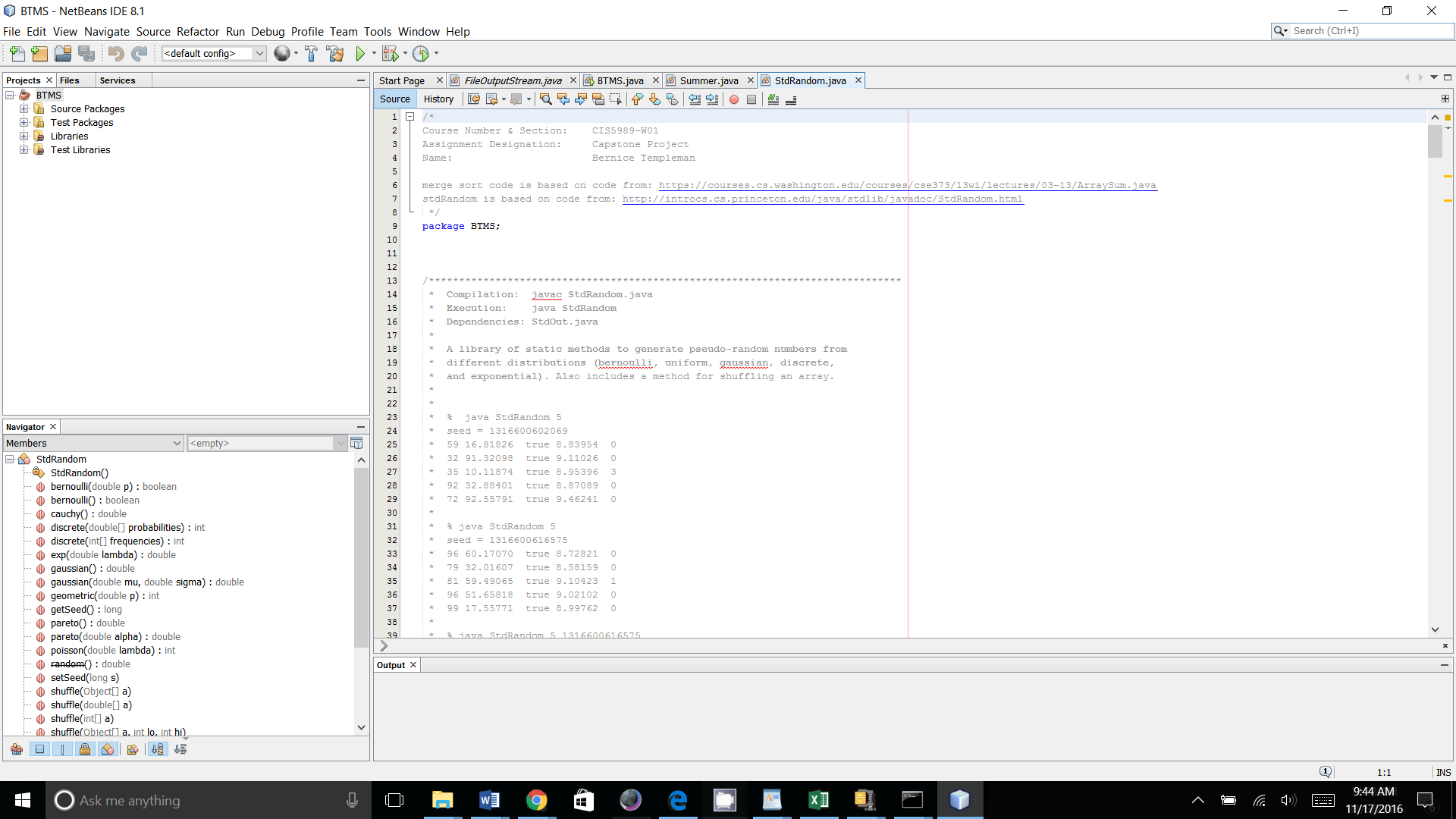
**APPENDIX C**

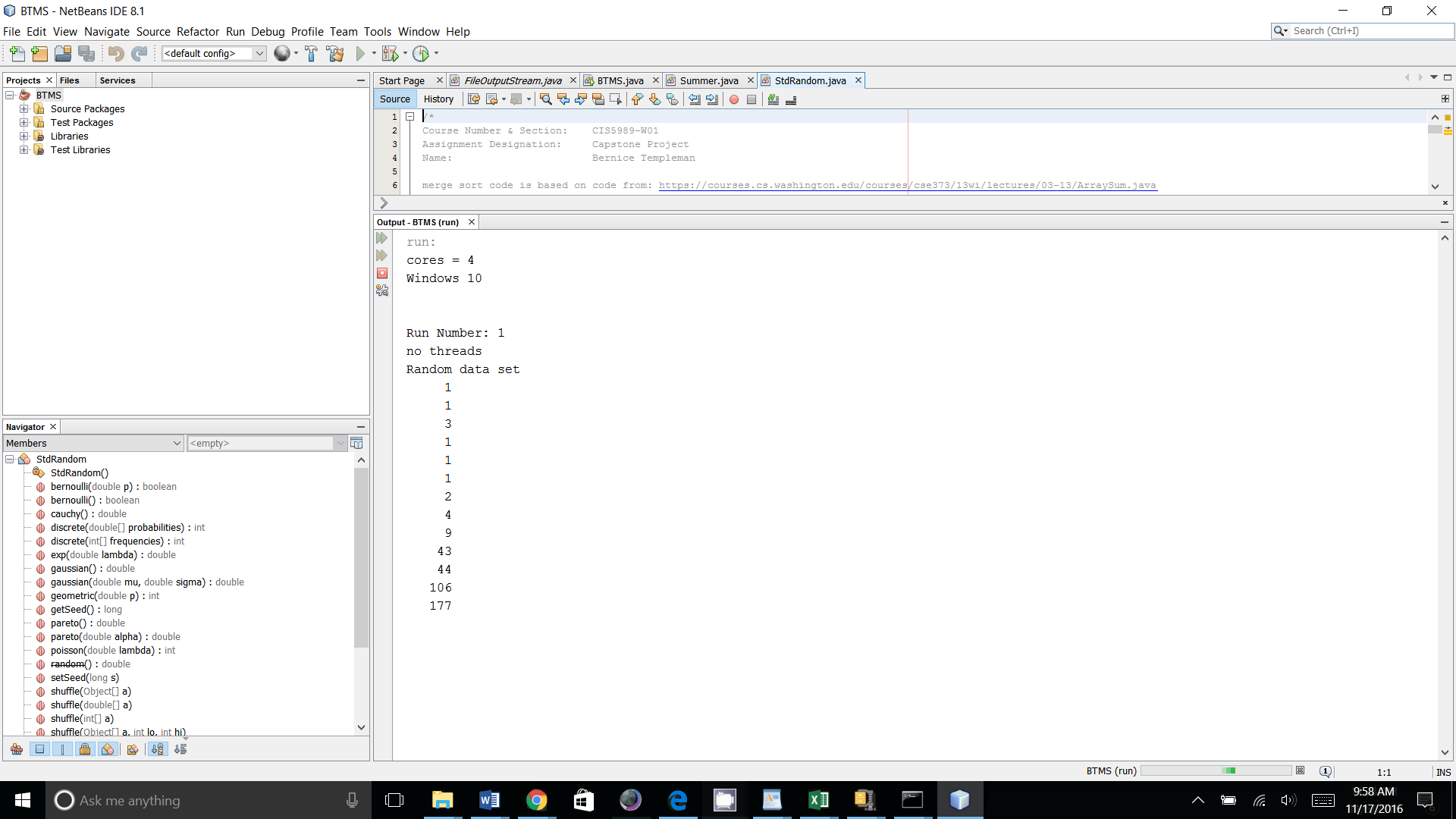
**User’s Manual**

This section provides all instructions for user to run the application and perform all functionality once the application has been installed

1. BTMS is a Java application to test non-threaded and threaded Merge Sort Performance. Once you have Java, NetBeans and the “BTMS” file imported into NetBeans, follow these steps to demonstrate the capabilities:

* Run from NetBeans
  + Run NetBeans
  + Right click on the “File” Menu: select “Open Project” then select "BTMS"
  + Right-Click the mouse over the "BTMS"
  + Select Run





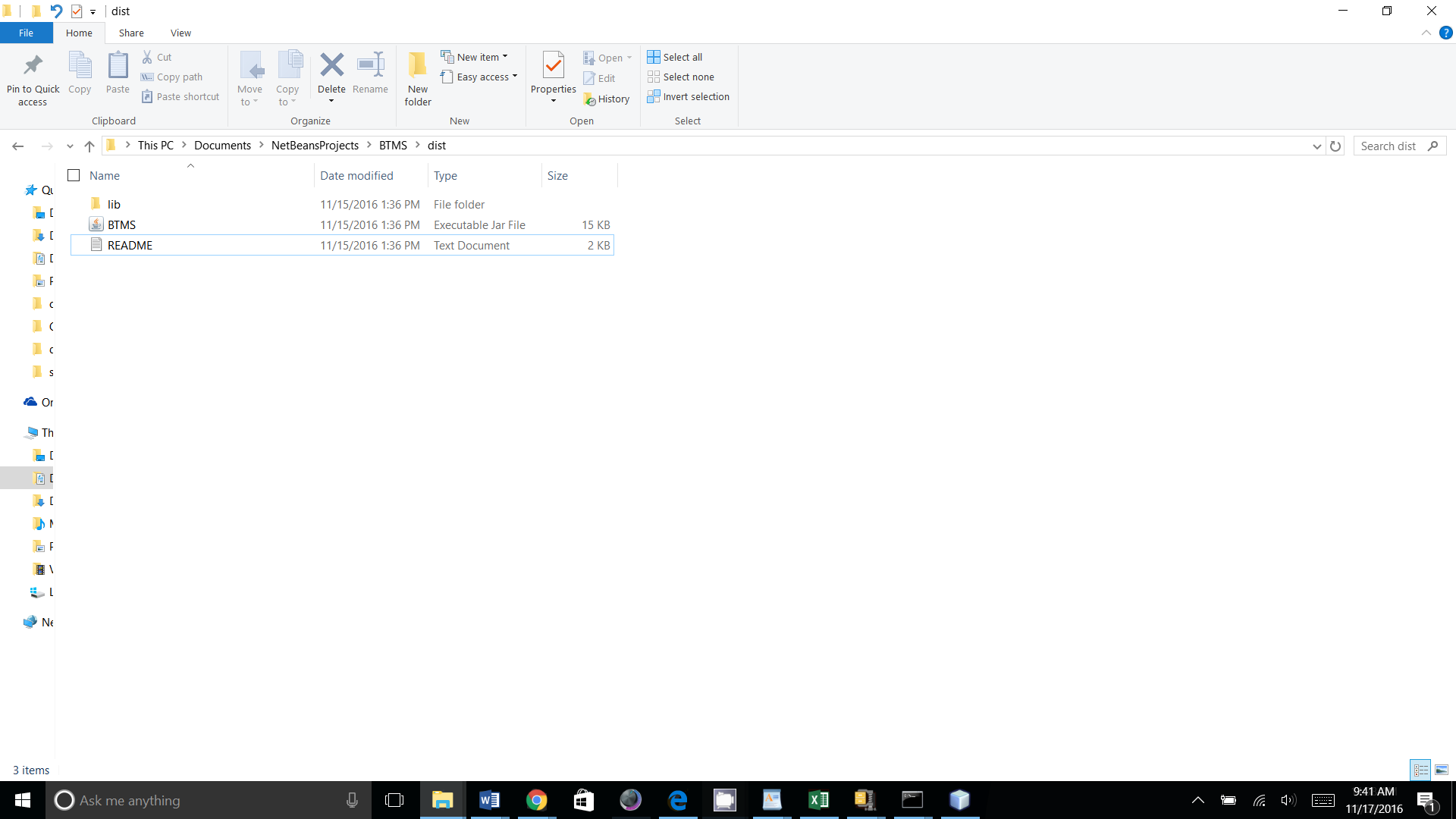
* Run from Command Line:

Windows 10:

Open the Command Prompt

To run the project from the command line, go to the BTMS dist folder

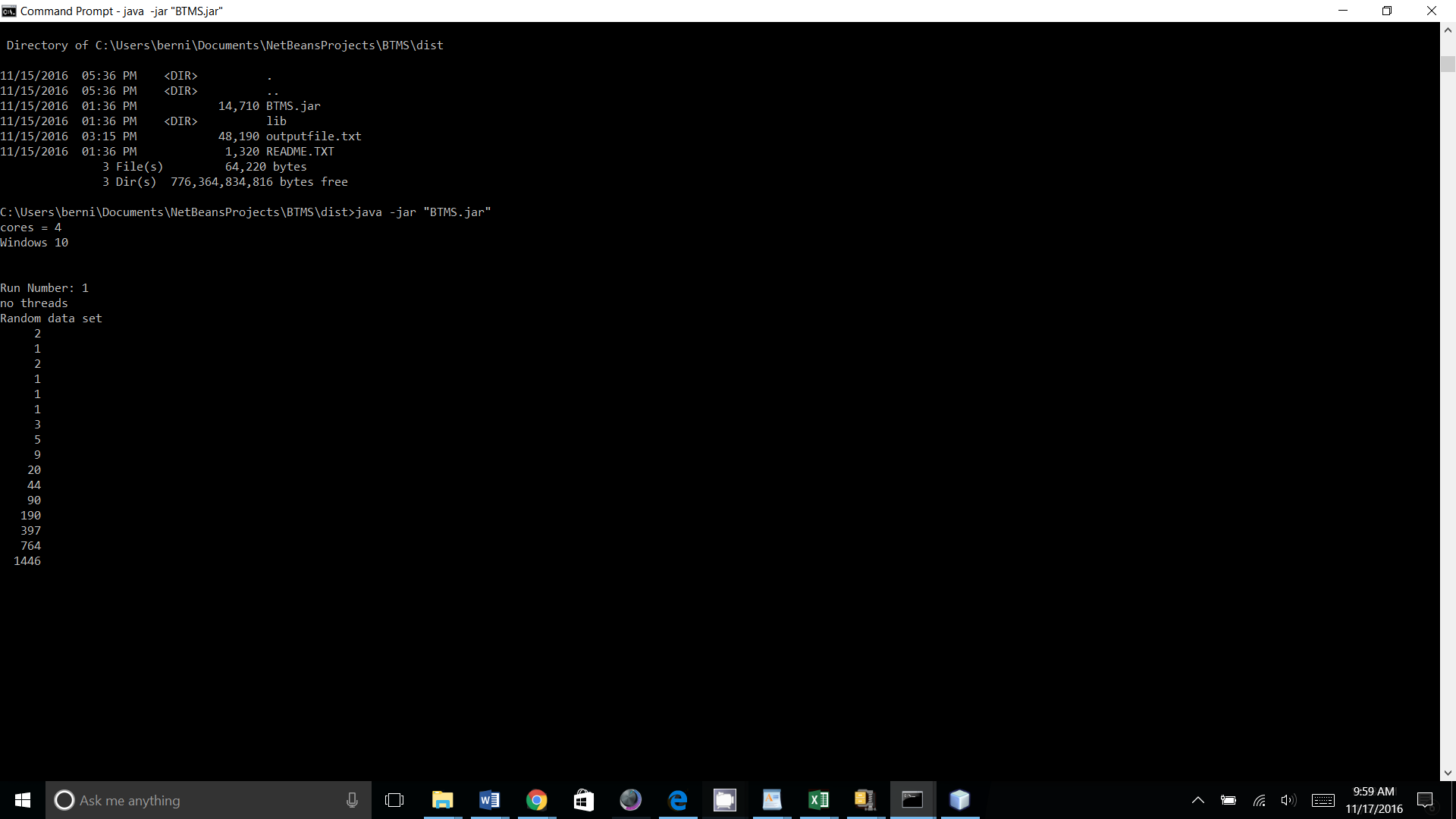
The dist folder In Windows Looks like this:



In the command prompt cd to the dist folder.

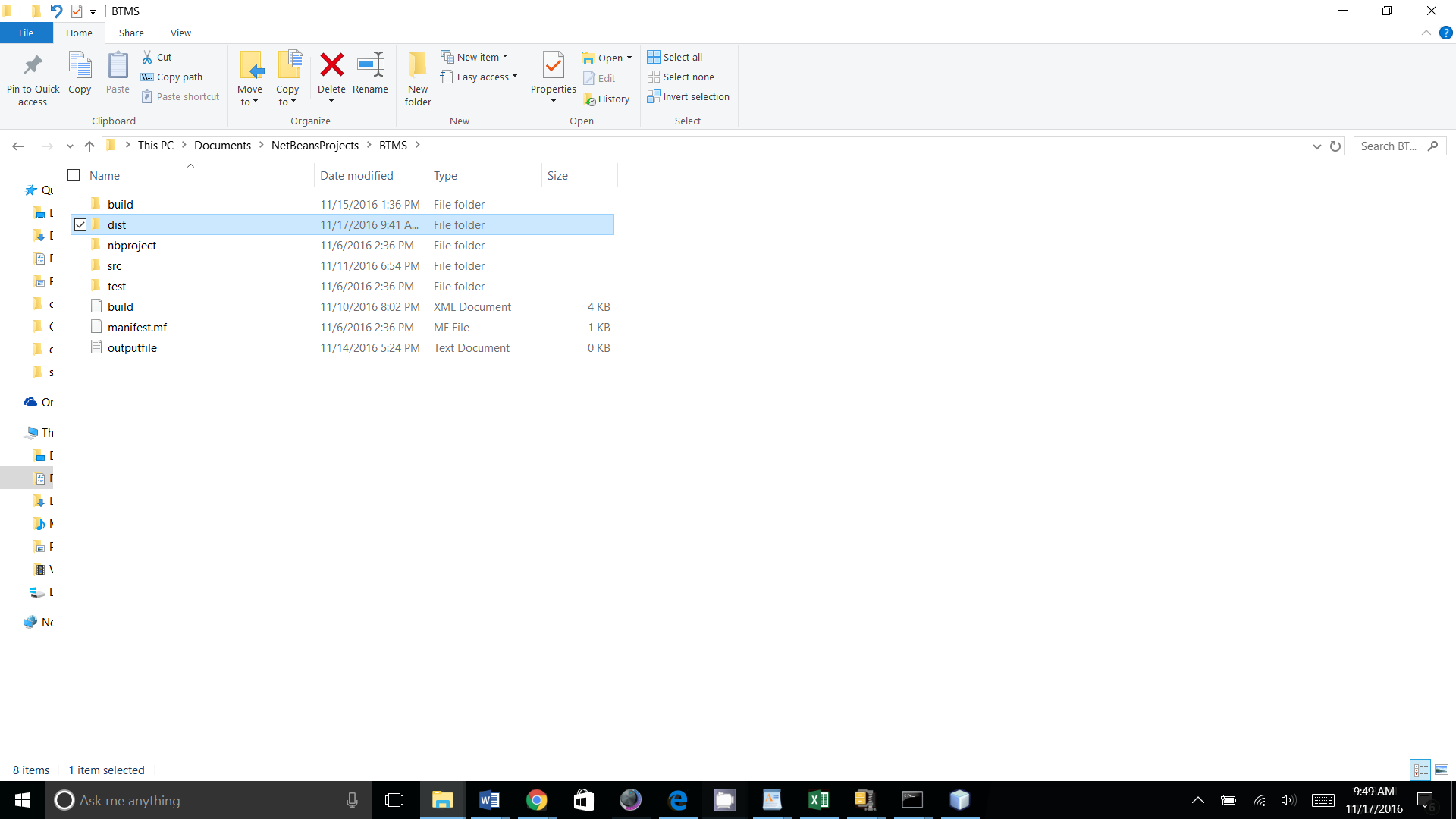
Type the following in the Command Prompt Window:

**java -jar "BTMS.jar"**

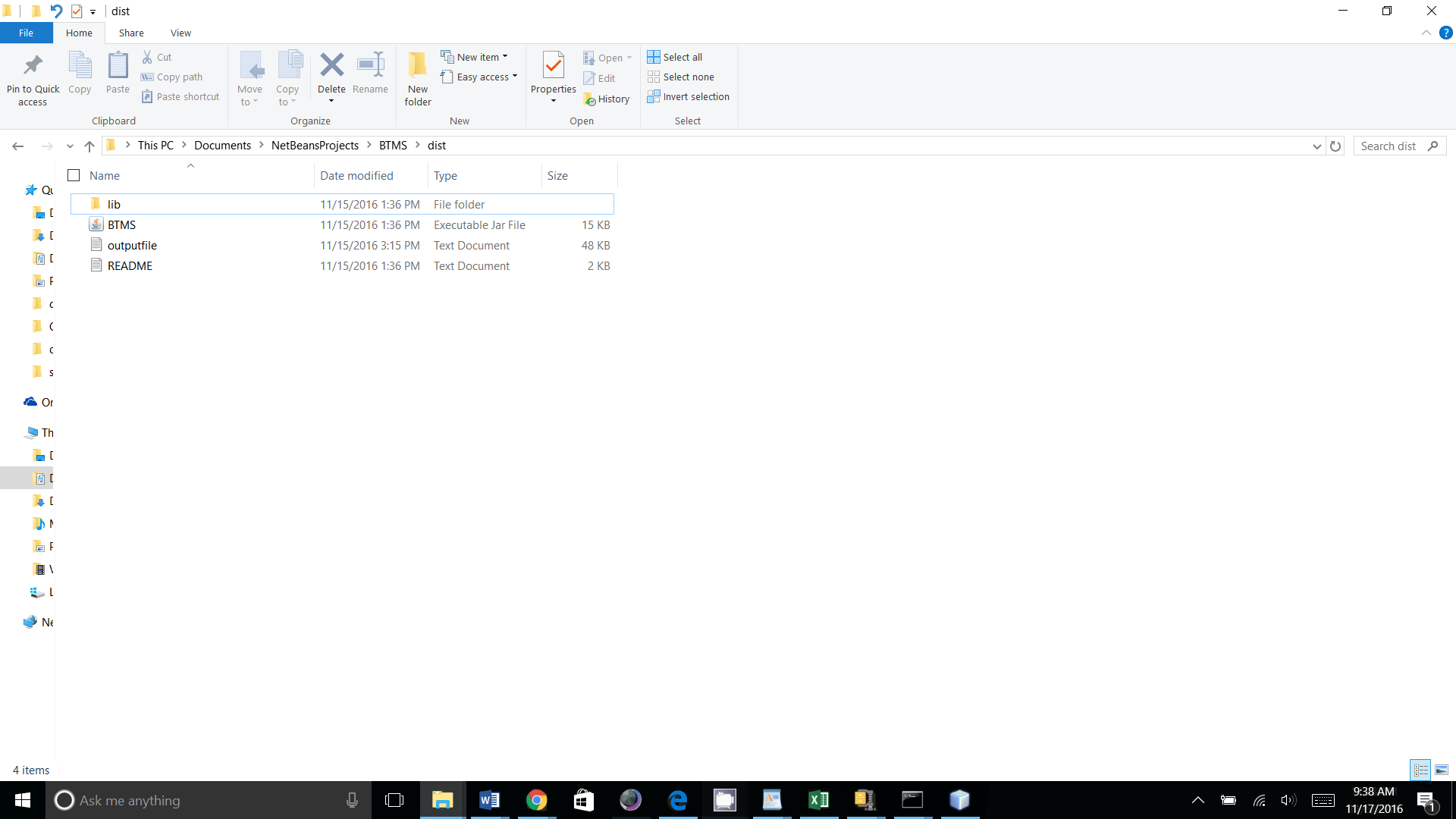


2. Output file will be written to the Java Project Directory

* NetBeans: file will be in the BTMS folder



* Command Line: file will be in the dist folder



3. Output can be copied into Excel Spread Sheet Template and saved to a different name:

