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.” Merge Sort is considered as a stable sort. It is also considered as well-suited for sorting data in secondary storage. (Heineman).

### Java Threads

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

“With the spread of muticore 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.”

**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 machines, …

<http://www.vmware.com/solutions/virtualization.html>

Another popular Virtual machine \_\_\_\_ is Oracle VirtualBox is shareware. It can be used on different platforms including Windows and Linux. For these reasons, I decided to use VirtualBox to satisfy the Virtual Machine project requirement.

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

## 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.

MERGESORT(A)

ifLENGTH(A) =< 1

then returnA

B <- MERGESORT(first half of A)

C <- MERGESORT(second half of A)

A <- MERGE(B, C)

returnA

**Threads**

The multi-threaded program will allow the following number of threads to be used for the merge-sort:

* 1
* 2
* 4
* 8
* 16

Threads were tested up to 16 threads. Due to hardware limitations (processors), increasing the number of threads to larger sizes did not improve performance.

**Data Set Size**

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

Larger data sets were not tested due to hardware limitations.

**Data Set Type**

The following data sets were tested:

* Random : http://docs.oracle.com/javase/8/docs/api/java/util/Random.html
* Discrete
* Uniform
* Bernoulli
* Gaussian
* Zero

**Using Exiting Code and Libraries**

* **Merge Sort**
* **Threaded Merge Sort**
* **Random**
* **STDRandom**

**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 running. It will be installed on an Intel platform running:

* Windows 10 Operating System (OS)
* Oracle Virtual Box Windows 10 VM
* Oracle VM Virtual Box 5..1.8 ( with extension pack)Ubuntu VM

**Operating Systems**

Windows

Ubuntu

Windows host

Decided not to boot to a Ubuntu USB for performance reasons.

Performance will not be tested on an Ubuntu bootable USB drive because research found shows there are performance costs running on an USB. <http://askubuntu.com/questions/21741/performance-cost-of-running-ubuntu-from-external-hard-drive>

Information and downloads for Java can be found at <https://docs.oracle.com/javase/8/>.

## Development Tools

Development will be done on an Intel machine running Windows 10 and a Linux OS Bootable drive. I will be using NetBeans IDE for Java EE Developers. Information and downloads can be found at <http://www.oracle.com/technetwork/articles/javase/jdk-netbeans-jsp-142931.html>.

## Programming Language and Code Libraries

BTMS will be programmed in the Java programming language using the NetBeans IDE.

Washington University has the merge sort and threaded merge sort java code available online. I decided to use this code in my performance tests.

**Data Sets**

There are different APIs to create Data Sets. A common API is the Random API which is included in Java. Another one is StdRandom which I found on the Princeton website. It includes methods to create different data sets. I chose the Random and some of the methods in the StdRandom API to test the merge sorts.

**User Interface**

## Graphical user interfaces use threads and impact performance. For these reasons I decided not to use a GUI and run the application from NetBeans. A JAR file will be created so the application can be ran with out NetBeans.

**Camtasia**

**Anti-Virus**

* **McAfee**

**Configuration Management**

* **Github**
* **Google drive**

**Reports**

* **Word**
* **Excel**
* **Camtasia**
* **Gantt**

**Hardware**

**Operating Systems**

* **Windows 10**
* **Ubuntu**

**Virtual Machines**

* **Oracle VirtualBox**

# Deliverables:

# Packaging and Distributing Java Desktop Applications

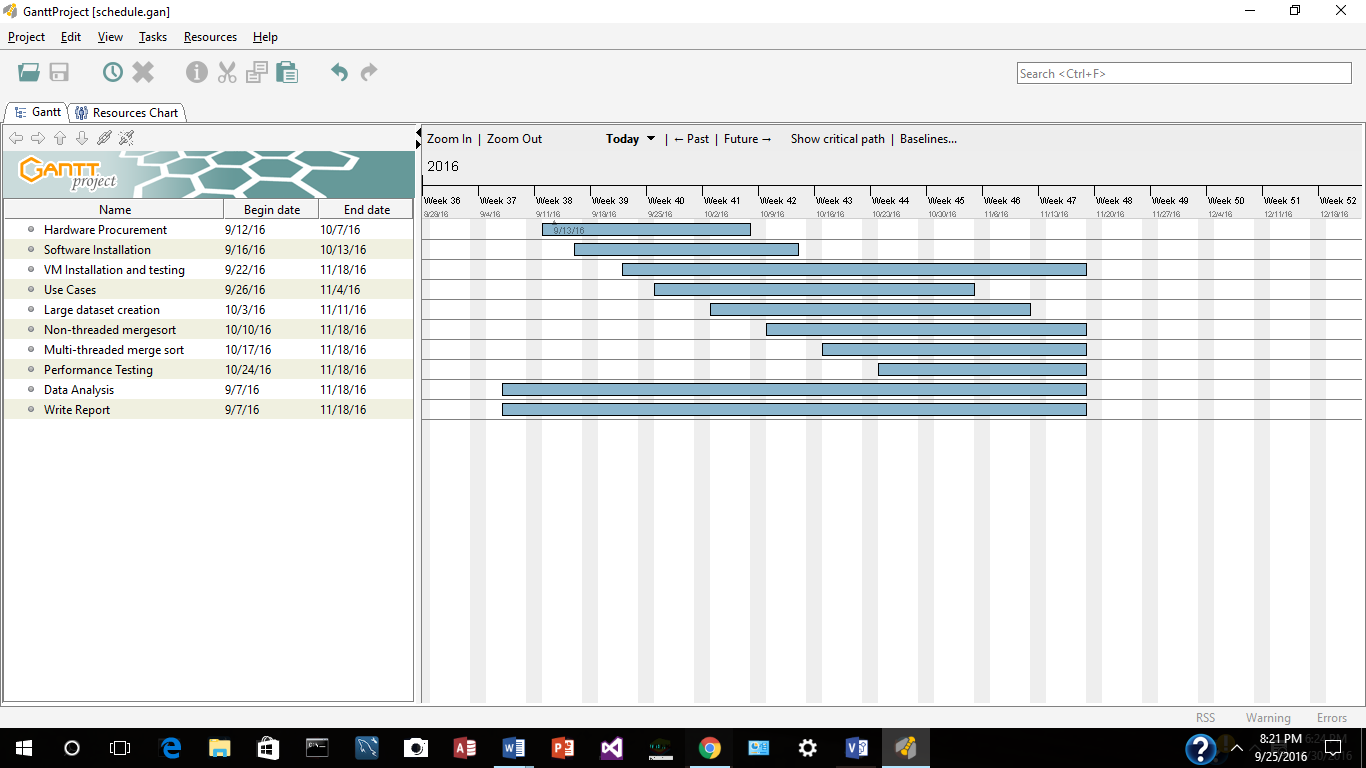
* Export Virtual Machines

## 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 test it during this phase. Next I will develop high-level use case diagrams for the threaded application. Also during this phase I’ll design the interface. Next, I will install Java and the NetBeans IDE on both Windows 10 hard and an Ubuntu Linux 14.04. 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 write a non-threaded merge-sort. After the non-threaded merge-sort is complete I will write the multi-threaded merge-sort. Then I will run performance tests to compare the non-threaded merge-sort with the threaded merge-sort with different number of threads.

|  |
| --- |
| Model Samsung - Notebook 7 Spin 2-in-1 15.6" |
| **Platform A** Windows 7 Enterprise  Edition |
| **Platform B** Windows 8.1 Pro |
| **Platform C** Ubuntu 14.04 LTS |
| **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 |
| Fig. 3 Hardware Platform Used in  Experiments |

# Schedule



# Results

**Results of Merge Sort**

**Comparing Merge Sort Times on different Operating Systems:**

I am a long time Unix person. I first wrote C and C++ programs on Unix Systems and then also did Unix System Administration. Later I also did RedHat Linux System Administration. There were always questions on which OS developers should use. When I first ran the merge sort on the different Operating Systems, I was surprised by the results.

The following are the results of the first runs on a Windows 10 and Ubuntu Virtual Machine with 1 Processor (Random Data Set Averages):

|  |  |
| --- | --- |
| 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 |

The results show that on the Virtual Machines with 1 processor, Windows 10 Non-Threaded was the fastest.

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

|  |  |
| --- | --- |
| 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 |

**Observations Of adding a processor to Virtual Machine:**

**Random Data Set Averages**

|  |  |
| --- | --- |
| 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 |

|  |  |
| --- | --- |
| 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 |

**Comparing Non-Threaded and Threaded Merge Sorts:**

Random Data Set Averages

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

**Random Data Set Averages**

|  |  |
| --- | --- |
| 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 |

Comparing Windows 10 Laptop Run via Command line vs NetBeans

|  |  |
| --- | --- |
| 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 |

**Comparing Different Data Sets:**

1. Non threaded vs Threaded merge sort performance

Chart of windows random set

Nonthreaded and each thread

Chart of linux random set

Nonthreaded and each thread

1. Windows vs Linux
2. Any difference in different data sets?
3. Data for windows laptop (different number of processors)?

Windows non threaded?

Multi threaded very large data sets?

Strange windows time cost at the beginning of threaded runs (running in loop)

Test windows threaded 1 run at a time to see time cost each run?

Latest Windows 10 and Ubuntu

Netbeans error message importing a project from a Zip file, but project is imported when you go to “open existing project”3

# Future Enhancements

Performance optimizations

External data set

*“Many applications will find the method*[*Math.random()*](http://docs.oracle.com/javase/8/docs/api/java/lang/Math.html#random--)*simpler to use.*

*Instances of java.util.Random are threadsafe.*

*However, the concurrent use of the same java.util.Random instance across threads may encounter contention and consequent poor performance. Consider instead using*[*ThreadLocalRandom*](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadLocalRandom.html)*in multithreaded designs.”*

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

# References

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

Wang, Deming, et al*. An Implementation of Sorting Algorithm Based on Java Multithread*

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

Concurrency Tutorial by Oracle,

<http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>, Accessed in September 2016.

Java Documentation by Oracle,

<https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html>, Accessed in September 2016.

Java Documentation by Oracle,

<https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html>, Accessed in September 2016.

merge sort code is based on code from: https://courses.cs.washington.edu/courses/cse373/13wi/lectures/03-13/ArraySum.java

http://docs.oracle.com/javase/8/docs/api/java/util/Random.html

stdRandom is based on code from: http://introcs.cs.princeton.edu/java/stdlib/javadoc/StdRandom.html

<http://www.vmware.com/solutions/virtualization.html>

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

<https://netbeans.org/kb/articles/javase-deploy.html>

https://netbeans.org/files/documents/40/860/DeploymentTutorial.zip

# 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 Machines:**

* 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
* Download and Import project from zip file (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.

1. Unzip the "WebApplication1.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., "WebApplication1" in othis case)

========================

BUILD OUTPUT DESCRIPTION

========================

When you build an Java application project that has a main class, the IDE

automatically copies all of the JAR

files on the projects classpath to your projects dist/lib folder. The IDE

also adds each of the JAR files to the Class-Path element in the application

JAR files manifest file (MANIFEST.MF).

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

type the following:

java -jar "BTMS.jar"

To distribute this project, zip up the dist folder (including the lib folder)

and distribute the ZIP file.

Notes:

\* If two JAR files on the project classpath have the same name, only the first

JAR file is copied to the lib folder.

\* Only JAR files are copied to the lib folder.

If the classpath contains other types of files or folders, these files (folders)

are not copied.

\* If a library on the projects classpath also has a Class-Path element

specified in the manifest,the content of the Class-Path element has to be on

the projects runtime path.

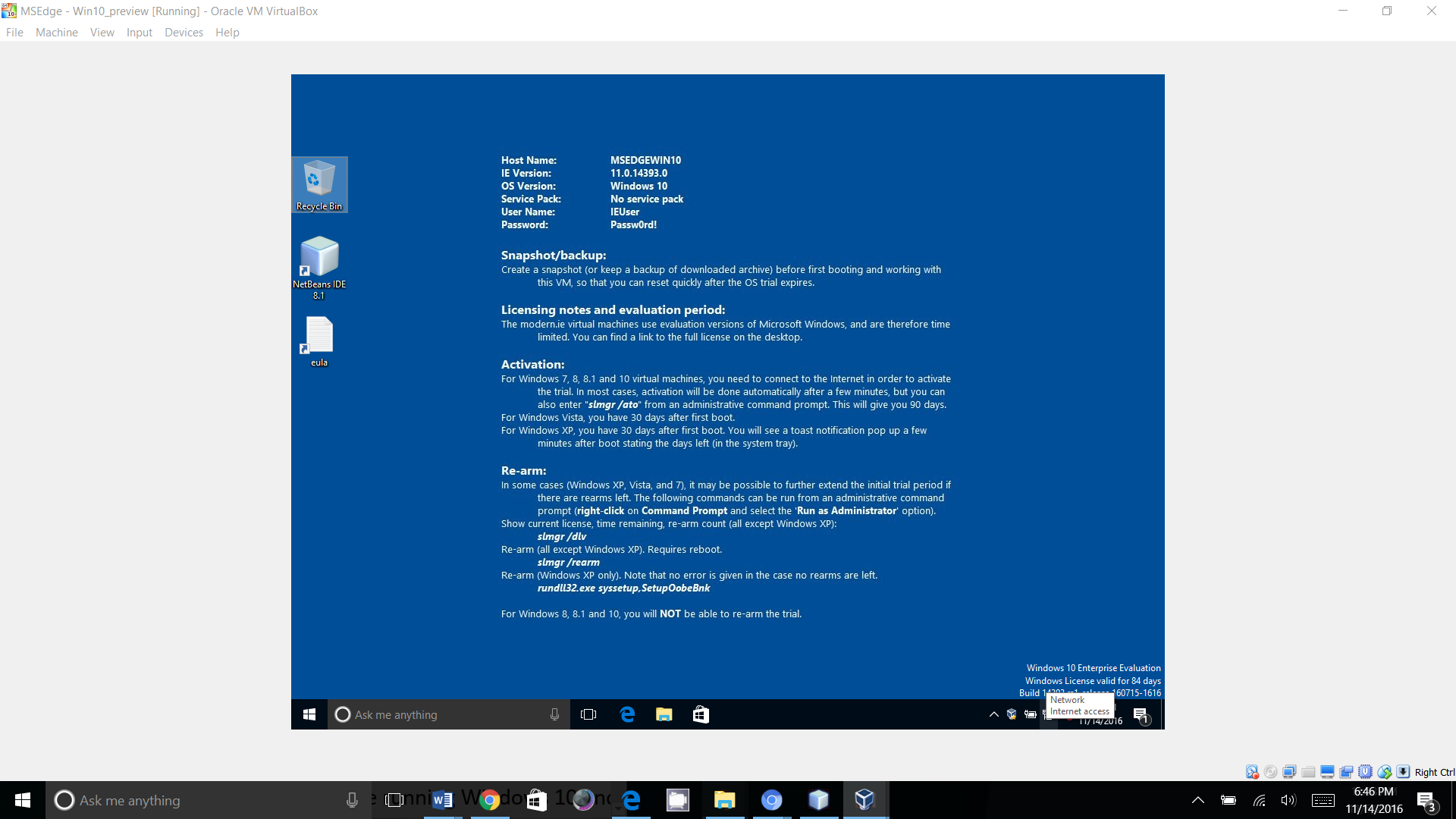
\* To set a main class in a standard Java project, right-click the project node

in the Projects window and choose Properties. Then click Run and enter the

class name in the Main Class field. Alternatively, you can manually type the

class name in the manifest Main-Class element.

**B.2 Steps to install BTMS on Windows 10 Virtual Machine**



**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 it up and running, follow these steps to demonstrate the capabilities:

* Run from NetBeans
  + Run NetBeans
  + Open the "BTMS" project file
  + Right-Click the mouse over the "BTMS"
  + Select Run
* Run from Command Line:

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

type the following:

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