**Project 1- Program Assignment (CSC 505)**

**mraveen, nsompal**

**Objective:**

A comparative study on the asymptotic analysis of various sorting algorithms, tested with sorted input, descending order input and randomly generated input of varying size.

**Testing Environment Specifications:**

OS: MacOS High Sierra

Processor: 2.6 GHz Intel Core i7

Memory: 16 GB 2133 MHz LPDDR3

Graphics: Intel HD Graphics 530 1536 MB

Language: Java 9.0.1

Compiler Version: javac 9.0.1

Cache: 6MB shared L3 Cache, L3cachesize: 6291456

**Experimental Approach:**

**Sorting algorithms considered:**

* Insertion sort
* Merge sort
* Heap sort
* Java utility sort

**Inputs chosen:**

**Insertion and Merge sort:** Inputs of size ranging from **5000 to 640000**(taken in multiples of 2)

**Merge Sort, Heap Sort, Utility Sort:** The input size ranging from **0.1 million to 12.4 million** (taken in multiples of 2)

**The types inputs are**:

1. Ascending order input (sorted)
2. Descending Order input (Sorted in reverse order)
3. Random inputs (distribution of data not known)

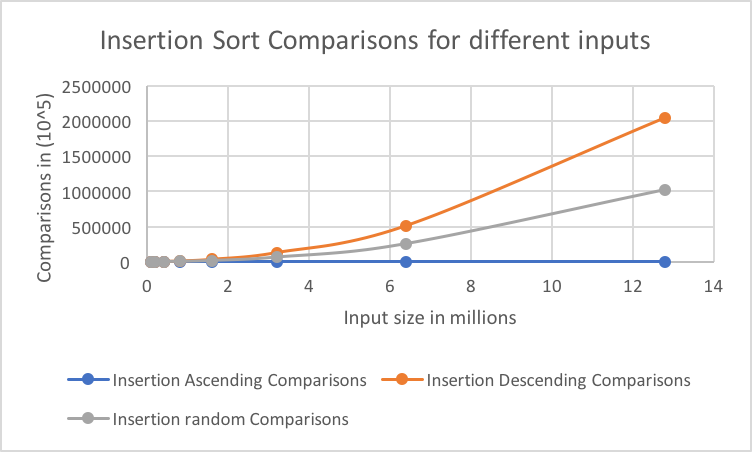
**Comparison and Analysis of Merge Sort and Insertion Sort:**

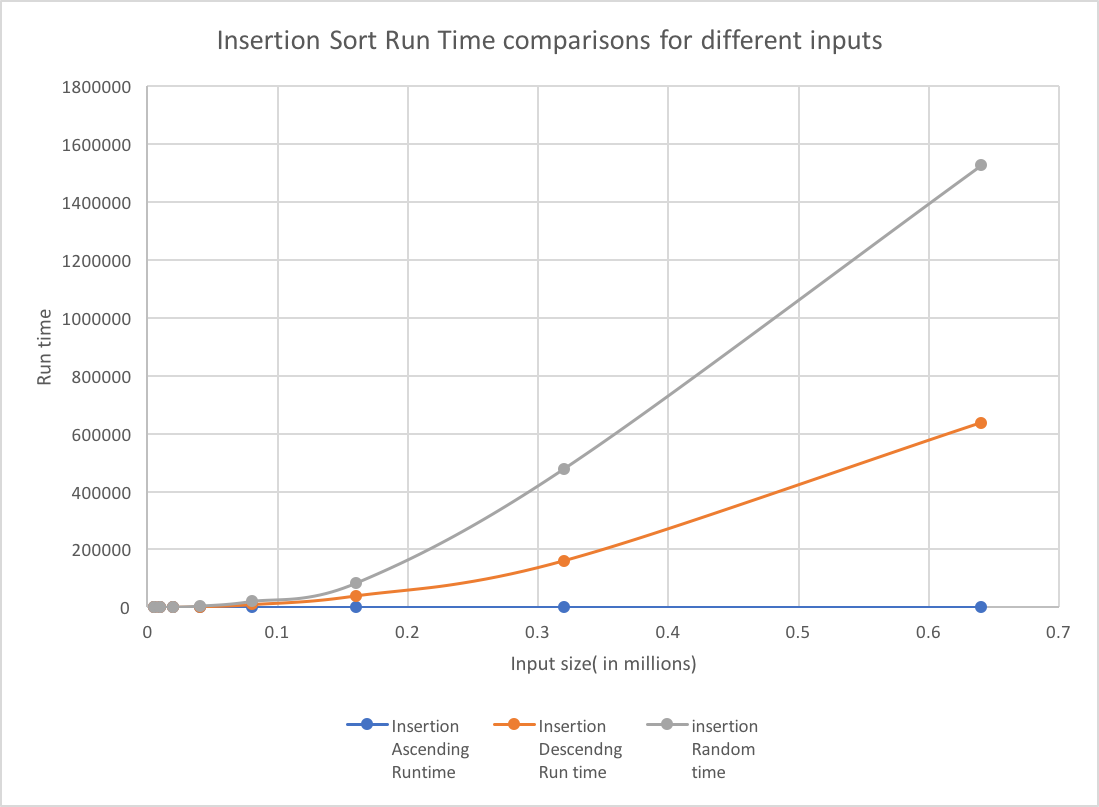
Upon analyzing the insertion and merge sort, we could observe the following:

1. Insertion sort is very effective for the Ascending order sorted input compared to merge.
2. Insertion Sort is comparable to Merge sort only when the input size is very less. As the input size increases, the growth rate of insertion sort greatly exceeds the growth rate of merge sort.
3. Insertion sort for random inputs performs between the ranges of worst case and best case.
4. Insertion sort could be preferred over merge sort for small input sizes if distribution of data is known.

****

****





**The comparison between Merge, Heap and Utility Sort is as follows:**

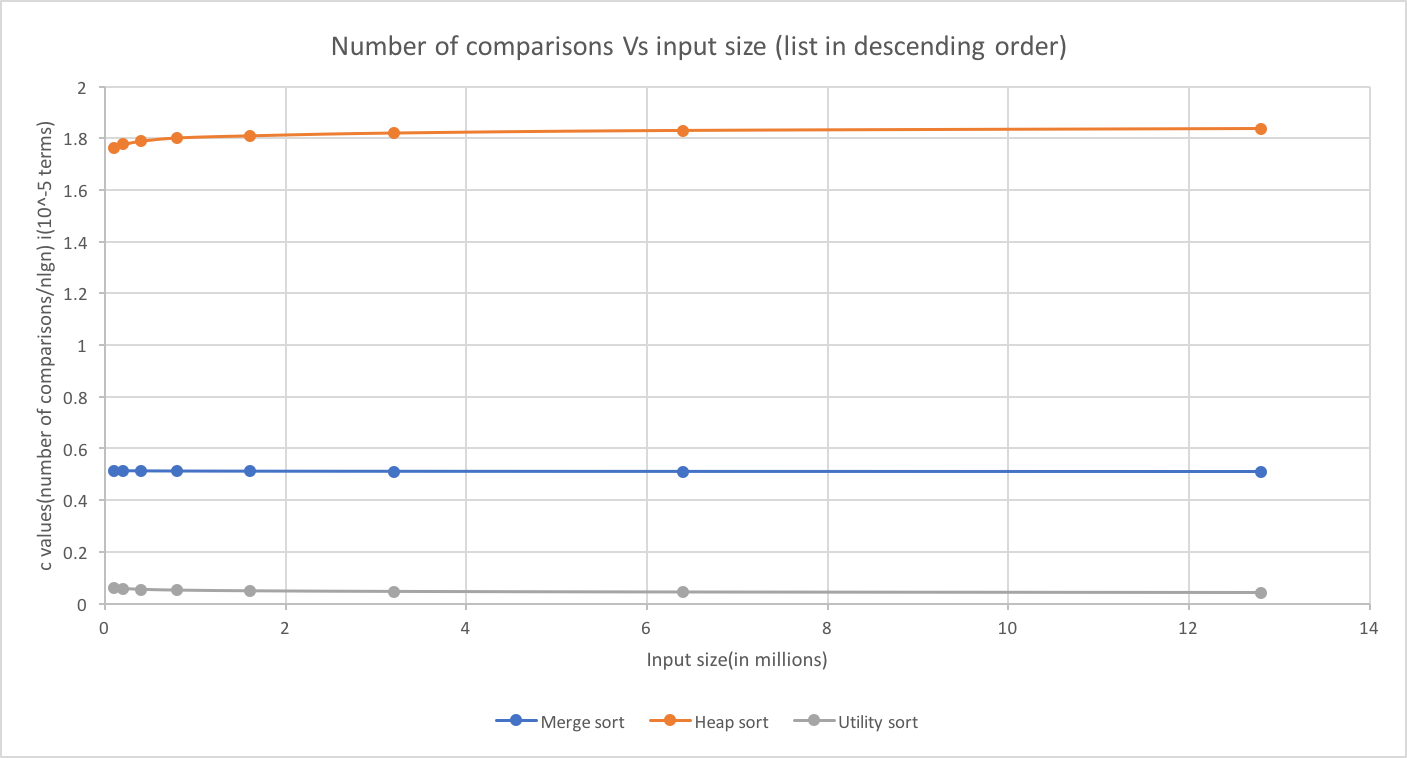
Comparing Merge Sort, Heap Sort and Utility Sort for the values ranging from 0.1 million to 12.4 million. The Results for the Comparisons are as follows:





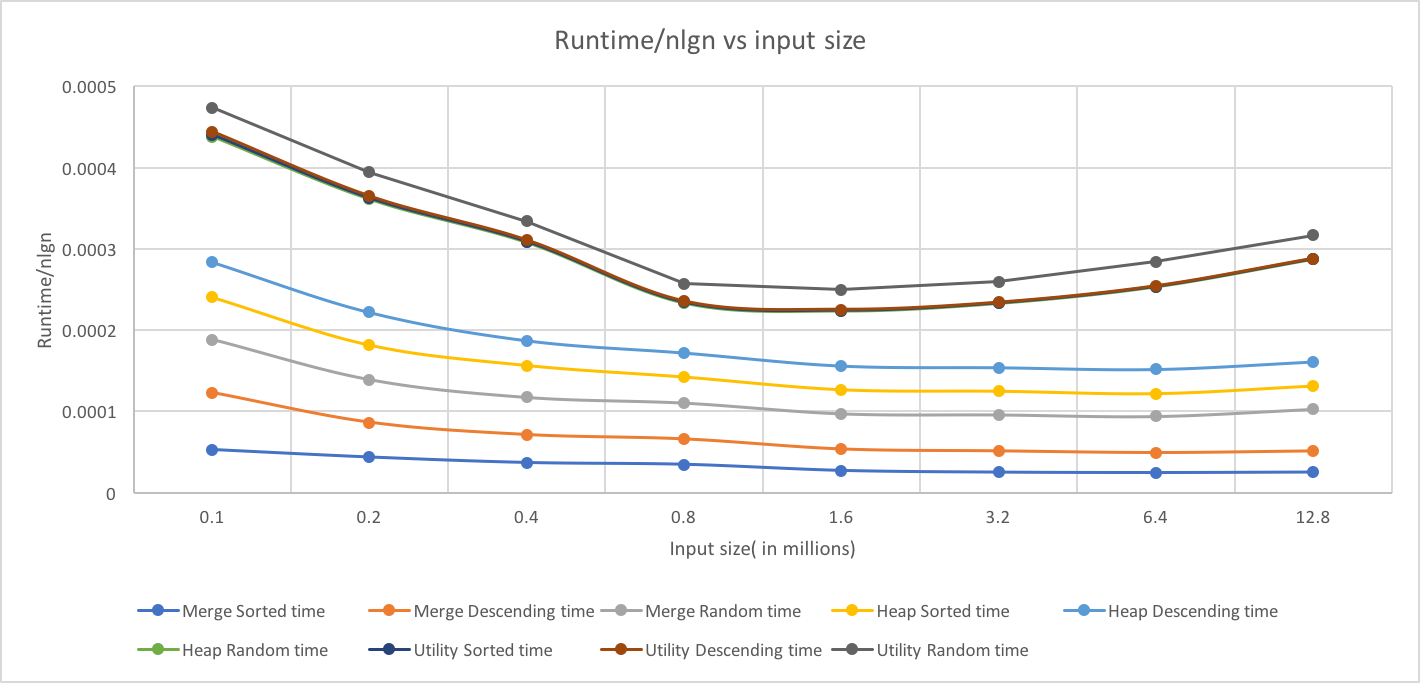
1. **Ratio between Actual and Predicted Comparisons**

The predicted number of Comparisons for Merge Sort(nlgn), Heap Sort(nlgn), Utility Sort.

****

**Analysis:**

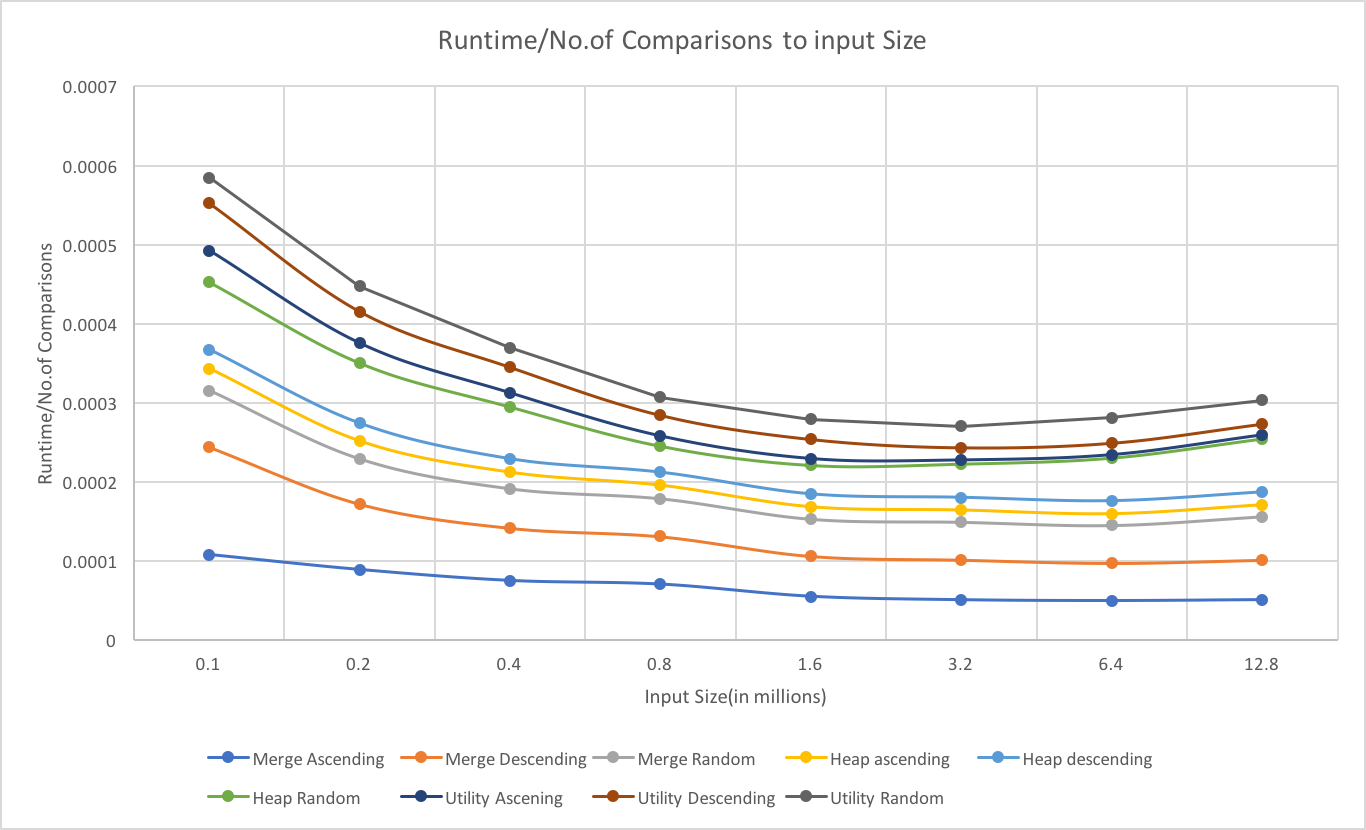
The ratio of predicted and actual comparisons is 1.8 for Heap Sort, 0.5 for Merge Sort and 0.1 for Utility Sort. These values show the value for the constant ‘C’ in the T(n) function for the worst-case analysis for the number of comparisons.

1. **Run Time/ nlgn:** From the graphs, we could infer that Heap sort has higher run time than Merge Sort at higher nlgn values****

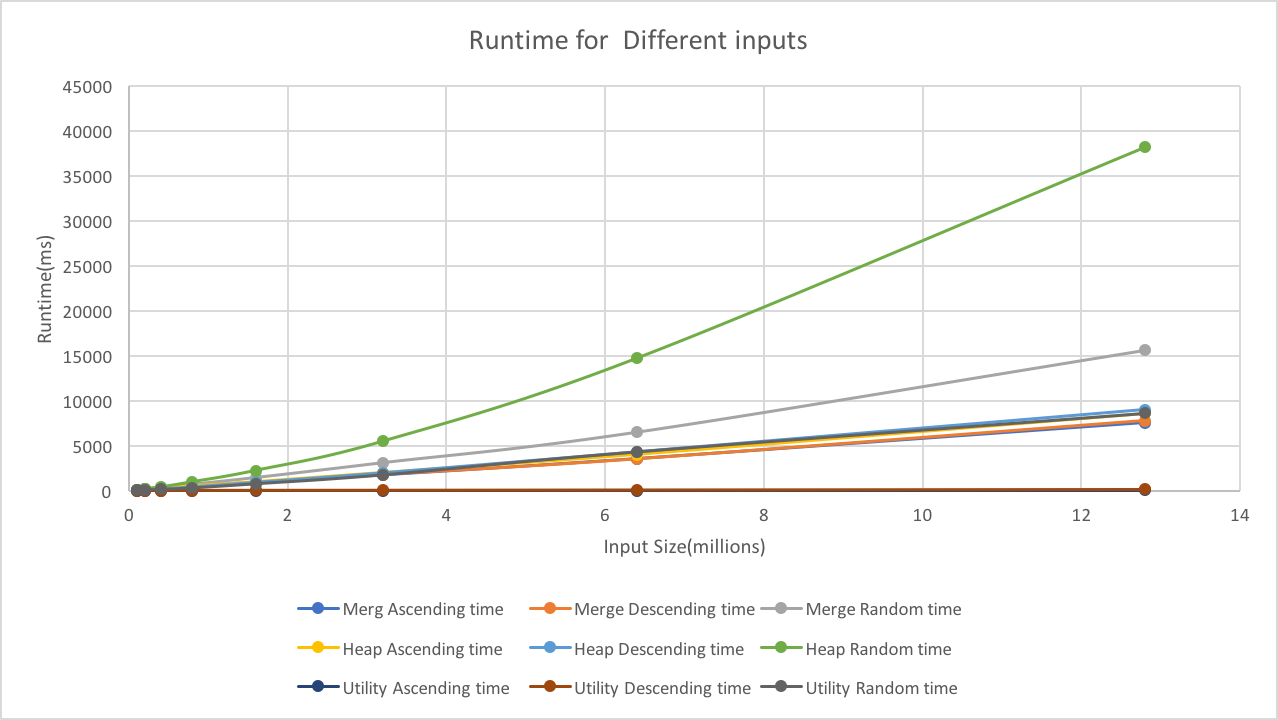
**Analysis:**

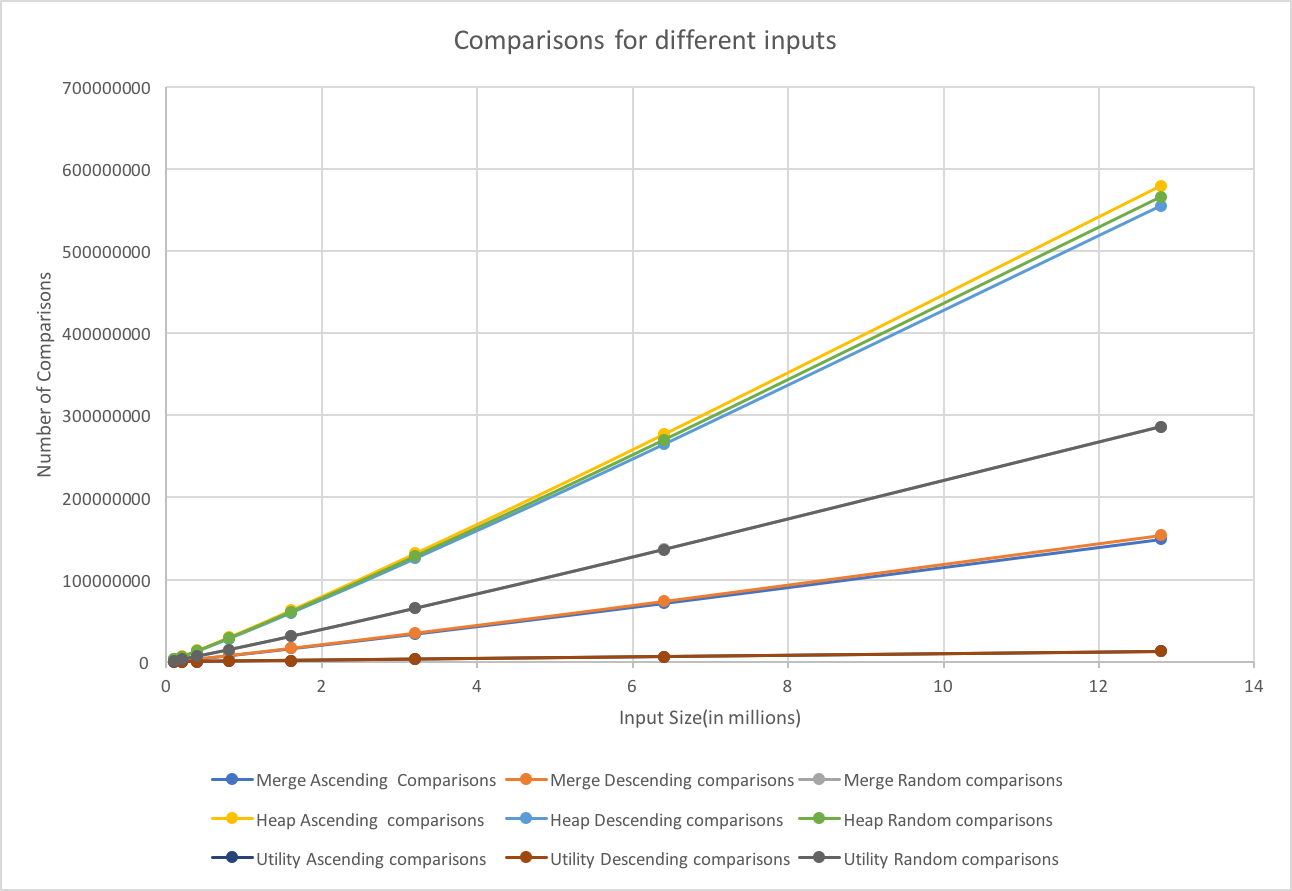
1. The Runtime obtained and the predicted runtime are almost equal
2. There is an anomaly observed for the Utility Sort at the values of 0.8 million, 1.6 million as there is a dip in the value.
3. Initially the obtained runtime is more than the predicted runtime.
4. **The relationship between runtime and number of comparisons**





1. **Comparison of the run time and comparisons with different inputs**





**Summary of the Project:**

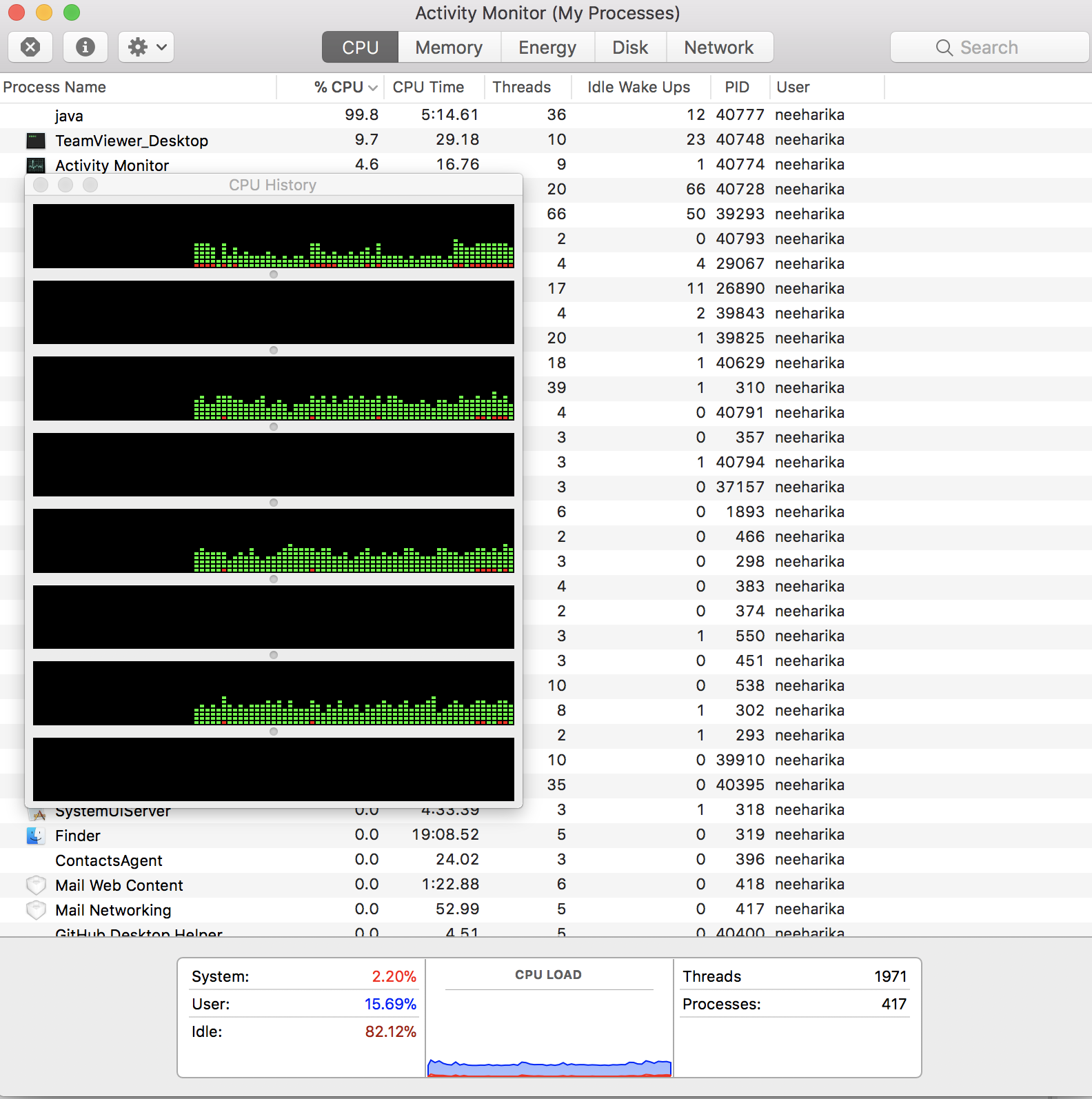
**Surprises:**

1. The Utility sort outperforms the other 3 algorithms.
2. There is an anomaly observed for the Utility Sort at the values of 0.8 million, 1.6 million as there is a dip in the value in the plot for Run Time/ nlgn to input size.
3. Run time for Heap sort when random values are given is very high than expected.
4. We expected insertion sort could be compared with Merge Sort, but the comparison is very difficult as Insertion sort grows drastically nullifying the Merge Sort increment.

**Theory Reflect the Reality:**

Theoretically Insertion sort outperforms for smaller values. But we observed that’s not completely true when random inputs are given as input.

The behavior of Merge and Heap Sort are in line with theory. Run time is directly proportional to the number of inputs. But, the random input takes higher run time than Descending order input to sort for all the sorts, but the comparisons are lower in line with the theory, which could probably be because of CPU running at 100% for Java process.



**How good is the sorting utility of the programming language?**

From the analysis of runtimes, we could see that Java’s internal sorting utility is highly efficient. It is extremely faster than the Merge, Insertion, Heap sorts. Using The utility will render very high efficiency and performance for any kind of inputs.

**Learnings:**

1. Using Java Utility sort for larger numbers is beneficial for good performance. On smaller sets the performance is almost the same.
2. Insertion sort should be used only for smaller inputs, especially for the Ascending sorted inputs, as it has huge runtime and a larger number of comparisons. Better not use Insertion sort!
3. Number of Comparisons (input size) and the Runtime are directly proportional).
4. Actual and predicted comparisons ratio gives the constant value ‘C’, which we usually get by theoretical analysis which is 1.8 for Heap Sort(2NlgN), 0.5 for Merge Sort(NlgN) and 0.1 for Utility Sort