2XB3 – Assignment 1 Analysis

Stephanus Jonatan

400072396

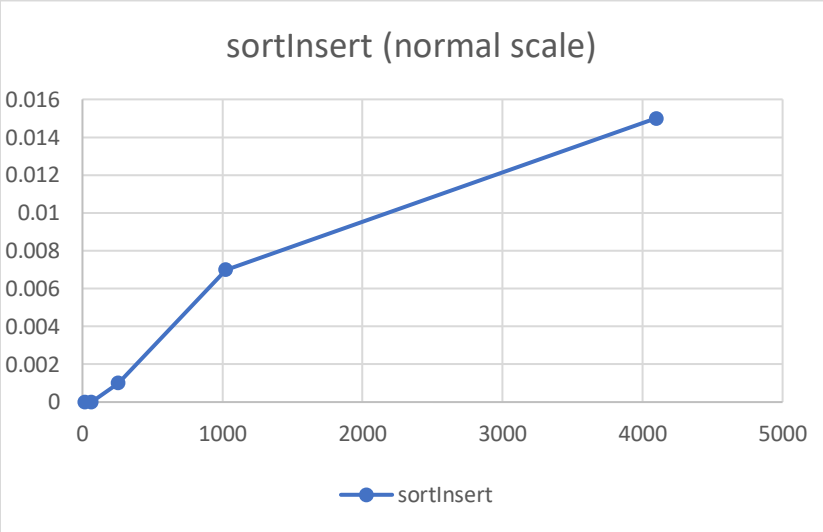
Jonatans

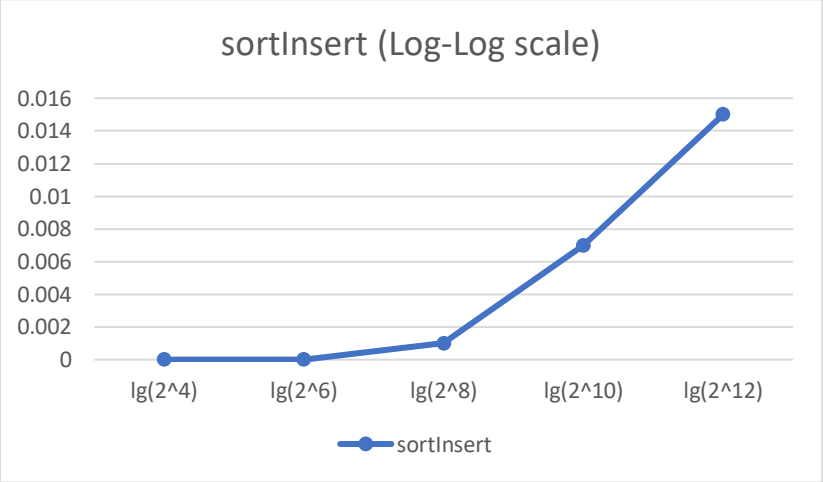
L01

Test Results						
		Sort Algorithm (Seconds)				
		sortInsert	*sortComparable*	*sortBinary*	*sortMerge*	*sortHeap*
Size of List	2^4	0.00000	0.00000	0.00000	0.00000	0.00000
	2^6	0.00000	0.00000	0.00000	0.00000	0.00000
	2^8	0.00100	0.00000	0.00100	0.00000	0.00000
	2^{10}	0.00700	0.00500	0.00300	0.00000	0.00100
	2^{12}	0.01500	0.03100	0.04500	0.00200	0.00200

(These values were taken from running SortTest.Java)

3.1.1 :





3.1.2

Hypothesize sortInsert to have a growth rate of $O(n^2)$

$$T(n) = Cn^2$$

Hypothesis:

$$T(2^{12}) = C(2^{12})^2$$

$$0.015 = C(2^{12})^2$$

$$C = 8.94e\text{-}10$$

$$T(n) = 8.94e\text{-}10\ n^2$$

3.1.3

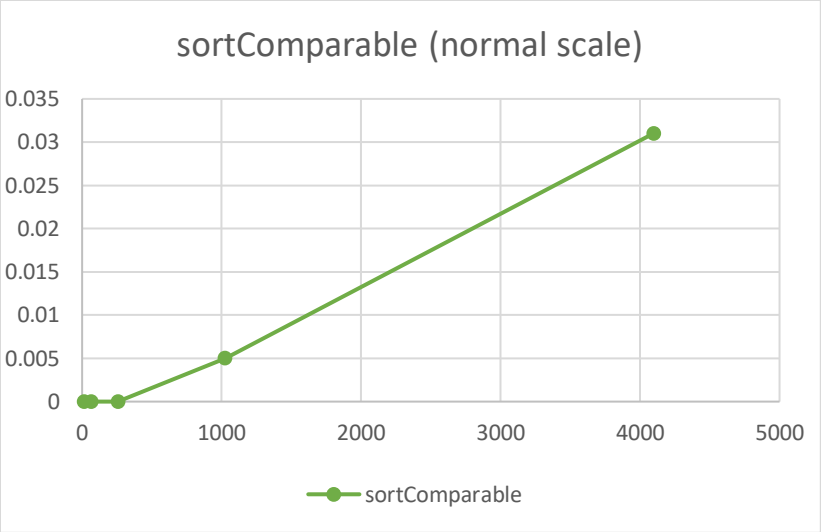
Hypothesis for $n = 2^{14}, 2^{16}$:

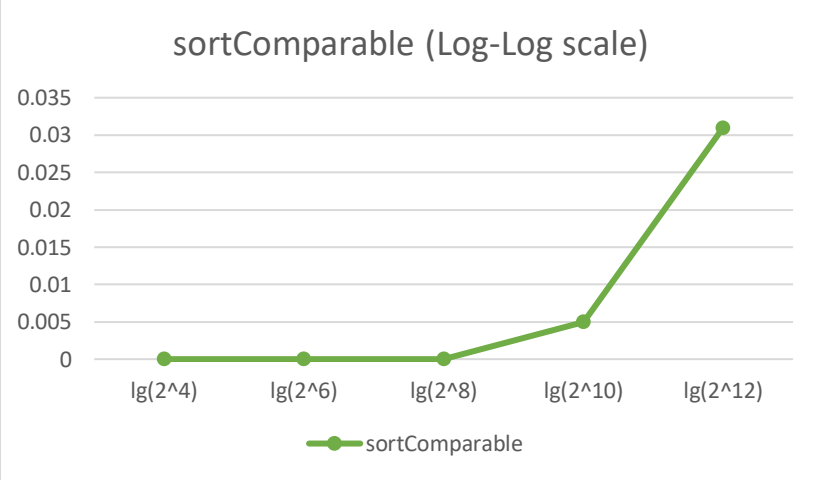
$$T(2^{14}) = 8.94e\text{-}10\ \ (2^{14})^2$$

$$T(2^{14}) = 0.023998s$$

$$T(2^{16}) = 8.94e\text{-}10\ (2^{16})^2$$

$$T(2^{16}) = 3.83970s$$





3.1.2

Hypothesize sortComparable to have a growth rate of $O(n^2)$

$$T(n) = Cn^2$$

Hypothesis:

$$T(2^{12}) = C(2^{12})^2$$

$$0.031 = C(2^{12})^2$$

$$C = 1.848e\text{-}9$$

$$T(n) = 1.848e\text{-}9\ n^2$$

3.1.3

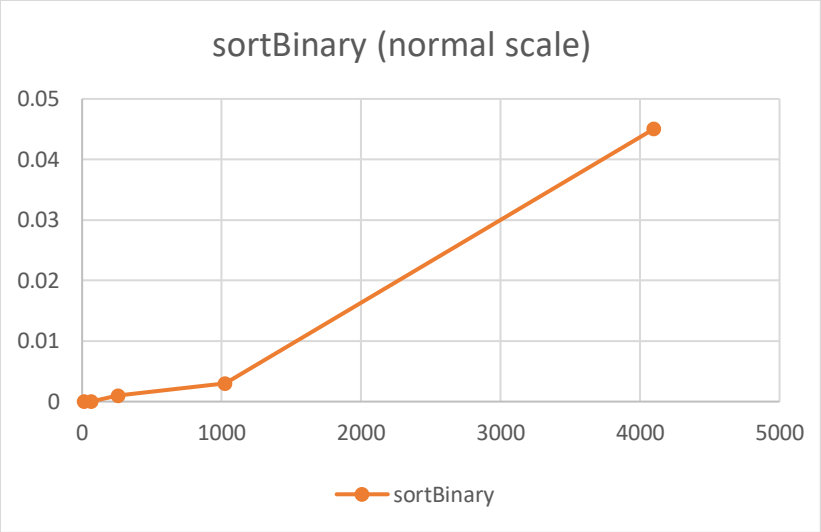
Hypothesis for $n = 2^{14}, 2^{16}$:

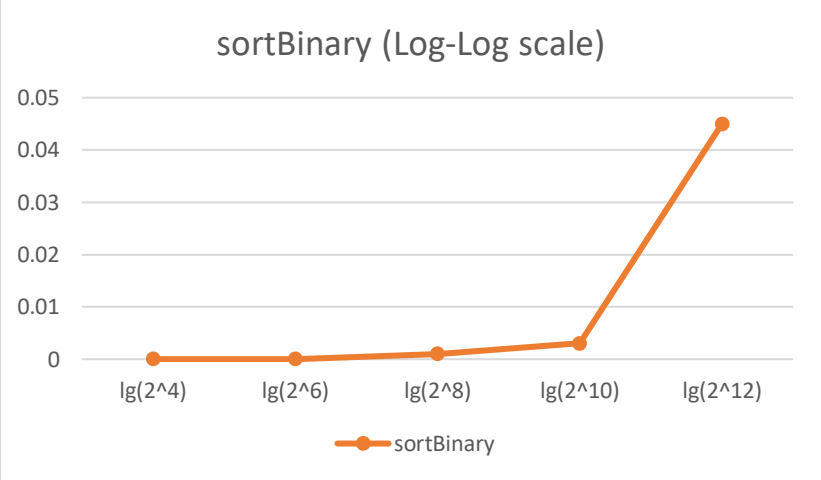
$$T(2^{14}) = 1.848e\text{-}9\ (2^{14})^2$$

$$T(2^{14}) = 0.49607s$$

$$T(2^{16}) = 1.848e\text{-}9\ (2^{16})^2$$

$$T(2^{16}) = 7.93710s$$





3.1.2

Hypothesize sortBinary to have a growth rate of $O(n^2)$

$$T(n) = Cn^2$$

Hypothesis:

$$T(2^{12}) = C(2^{12})^2$$

$$0.045 = C(2^{12})^2$$

$$C = 2.682e\text{-}9$$

$$T(n) = 2.682e\text{-}9\ n^2$$

3.1.3

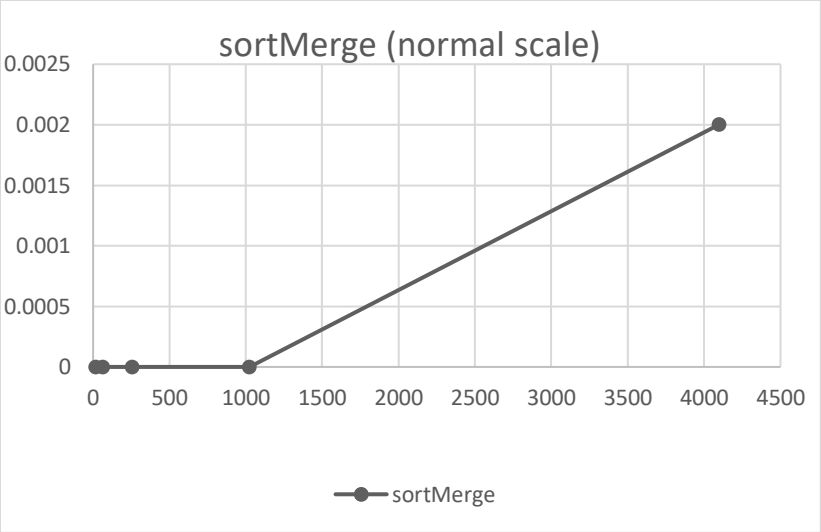
Hypothesis for $n = 2^{14}, 2^{16}$:

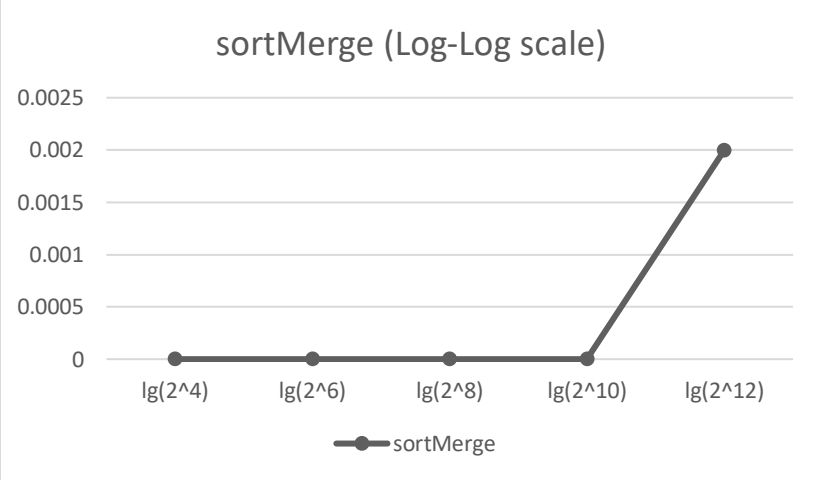
$$T(2^{14}) = 2.682e\text{-}9\ (2^{14})^2$$

$$T(2^{14}) = 0.71994s$$

$$T(2^{16}) = 2.68e\text{-}9\ (2^{16})^2$$

$$T(2^{16}) = 11.51910s$$





3.1.2

Hypothesize sortMerge to have a growth rate of O(nlg(n))

$$T(n) = Cnlg(n)$$

Hypothesis:

$$T(2^{12}) = Cnlg(2^{12})$$

$$0.002 = C2^{12}lg(2^{12})$$

$$C = 4.069e\text{-}5$$

$$T(n) = 4.069e\text{-}5 * nlg(n)$$

3.1.3

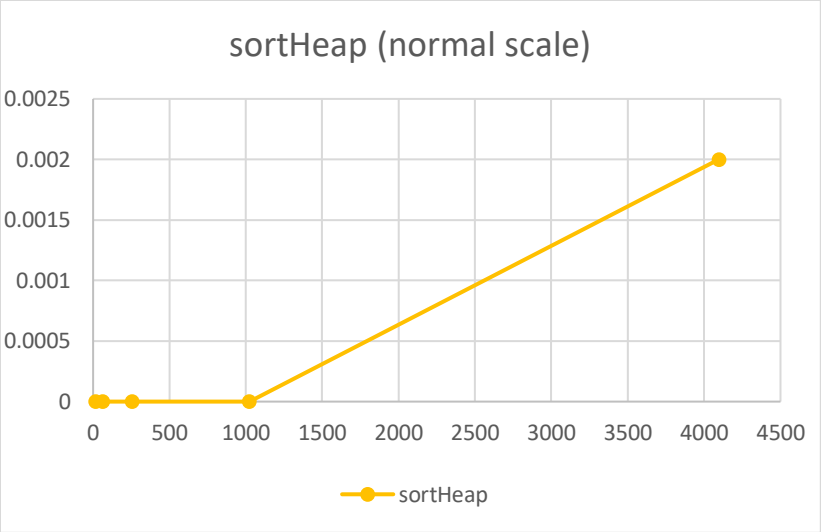
Hypothesis for $n = 2^{14}, 2^{16}$:

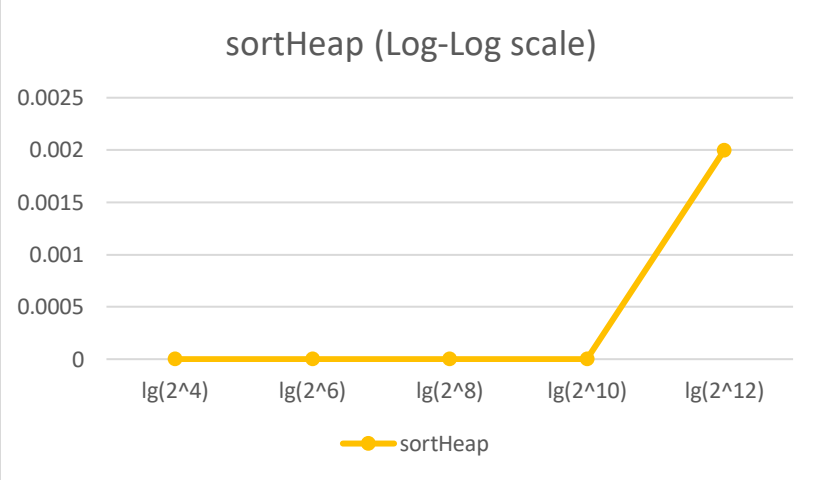
$$T(2^{14}) = 4.069e\text{-}5 * 2^{14}lg(2^{14})$$

$$T(2^{14}) = 0.00933s$$

$$T(2^{16}) = 4.069e\text{-}5 * 2^{16}lg(2^{16})$$

$$T(2^{16}) = 0.04267s$$





3.1.2

Hypothesize sortComparable to have a growth rate of O(nlg(n))

$$T(n) = Cnlg(n)$$

Hypothesis:

$$T(2^{12}) = Cnlg(2^{12})$$

$$0.002 = C2^{12}lg(2^{12})$$

$$C = 4.069e\text{-}5$$

$$T(n) = 4.069e\text{-}5 * nlg(n)$$

3.1.3

Hypothesis for $n = 2^{14}, 2^{16}$:

$$T(2^{14}) = 4.069e\text{-}5 * 2^{14}lg(2^{14})$$

$$T(2^{14}) = 0.00933s$$

$$T(2^{16}) = 4.069e\text{-}5 * 2^{16}lg(2^{16})$$

$$T(2^{16}) = 0.04267s$$

3.1.4

Test Results						
		Sort Algorithm (Seconds)				
		sortInsert	*sortComparable*	*sortBinary*	*sortMerge*	*sortHeap*
Size	2^{14}	0.16000	0.39100	0.20000	0.00700	0.00900
	2^{16}	4.89400	11.66700	5.25200	0.10400	0.02700

(By uncommenting the two for-loops at the bottom of gen.java (lines 66-79) and running gen.java, two new lines will be created in a1_in.txt consisting of a Job list of 2^{14} and 2^{16} elements. Then by running SortTest, all running times will be calculated and printed to the screen for each of the different sorting methods, applied on list sizes 2^4, 2^6, 2^8, 2^{10}, 2^{12}, 2^{14}, and 2^{16}.)

From the test results in the table under 3.1.4, only sortInsert and sortComparable, run at times similar to each respective hypothesis above. However, for sortBinary, sortMerge, and sortHeap, the recorded values of a particular run is not as expected.

sortBinary ran much faster than expected. Even tho the first 5 points of data on the log-log scale of sortBinary appeared to have $O(n^2)$, upon firther inspection with the new data, it would seem that the actual order of growth is O(nlog(n)). However, it was still not as fast as sortInsert. The reason for this could be the compareTo() function used in sortBinary, and that a bigger list size would show that sortBinary is much better than sortInsert.

sortMerge took 3 times longer than the hypothesized run time for a list of size 2^{14} and 2^{16} elements, while sortHeap took half the time. Despite the hypothesizes to have the same order of growth as each other, the results were drastically different from each other; one being slower, and the other being faster than the hypothesized value. This might be because the number of conditionals in Merge.java are more than the conditionals in Heap.java, resulting in a higher consant, C, than the value originally hypothesized.