Nanyang Technological University

CZ2001 Example Class 3 Report

Integration of Merge Sort and Insertion Sort

Group Members:
Atbas Ufuk Civan
Cheng Hui Thong
Laurensia Anjani
Takafumi Takemoto
Teo Zhe Wei

1. Introduction

The original Merge Sort algorithm uses the divide-and-conquer approach by splitting the input array into subarrays, and recursively sorting them. However, the use of recursive calls in this sorting method is inefficient when the sizes of the subarrays are small due to the large number of overhead. As such, the modified Merge Sort algorithm has an additional parameter, **S**, that acts as a threshold value for the size of the subarrays. When the size of the subarray is less than or equal to the value of **S**, the algorithm will switch to Insertion Sort. The difference in performance of the modified Merge Sort and the original Merge Sort in sorting the random integers in an ascending order will thus be explored in this lab report.

2. Generation of Data

Our program generates an array of random integers using seed values of 100, 200, 300, 400, and 500. The different array size, **n**, are also stored in an array, so that the program can loop continuously and create a new array of random integers using different **n** values. Our program will then sort the array of integers for 100 times with both the modified Merge Sort and the traditional Merge Sort. Our program will then display the average number of key comparisons and the average CPU time for both the modified Merge Sort and the traditional Merge Sort respectively.

3. Performance Statistics of Modified and Original Merge Sort Algorithms

The performance of both modified and original Merge Sort algorithms will be based on the average CPU time and the average number of key comparisons. In this comparison, we fixed the value of **S** to be 10, and the sort is carried out 100 times. The statistics of the average number of key comparisons and average CPU time can be seen from Table 1, with **n** as the array size.

Table 1. Performance statistics of the two different Merge Sort algorithms

	Modified Merge Sort		**Normal Merge Sort**	
n	**Average CPU Time/ns (100 counts)**	**Average No. of Key Comparisons (100 counts)**	**Average CPU Time/ns (100 counts)**	**Average No. of Key Comparisons (100 counts)**
1000	263335.2	4452.9	946360.2	5039.3
2000	254951.3	9693.0	1886344.0	10871.2
3000	1007418.0	15758.4	2846472.0	16999.8

4000	1188699.0	21470.9	4719533.0	23835.3
5000	1003505.0	26942.5	6205001.0	29895.0
10000	3341832.0	58813.4	19500000.0	64718.0
20000	11500000.0	127456.4	78600000.0	139266.4
30000	27100000.0	204690.2	193000000.0	219635.9
40000	43000000.0	274902.5	332552803.5	298523.2
50000	128000000.0	365331.8	497908574.8	382563.8
70000	189000000.0	514335.6	971593192.0	555292.9
90000	215000000.0	666521.3	1515477987.7	723750.5
100000	302000000.0	780669.1	1838544146.1	815135.7

From Figure 1, it is observed that the average number of key comparisons for both the modified and original Merge Sort algorithms increase at almost the same rate, as shown by the almost parallel lines.
The linear trendline of the modified Merge Sort has the equation: $y = 7.6729x – 14611$.
The linear trendline of the original Merge Sort has the equation: $y = 8.1526x – 13795$.
Despite the similar rate of increase (similar gradient) in the average number of key comparisons between the two algorithms, the average number of key comparison for the modified Merge Sort is generally lower than that of the original Merge Sort, proving its efficiency.

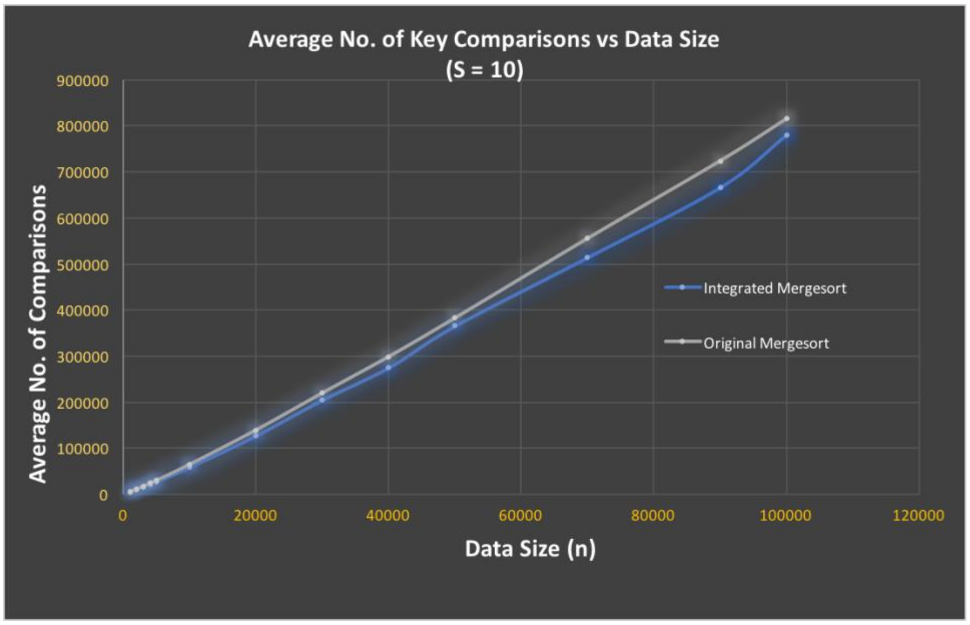

Figure 1. Graph to show the rate of increase of average no. of key comparisons between two Merge Sort algorithms

Moreover, the rate of increase of the average CPU time of the modified Merge Sort is significantly lower than that of the original Merge Sort, as shown in Figure 2. As array size increases, the average CPU time of the original Merge Sort increases in an exponential manner, while that of the modified Merge Sort increases at a more stable rate. It is because the average CPU time taken to do key comparison using Insertion Sort is much shorter than that using Merge Sort when the array size is small. As such, the combination of Merge Sort and Insertion Sort in the modified Merge Sort algorithm results in an overall lower average CPU time.

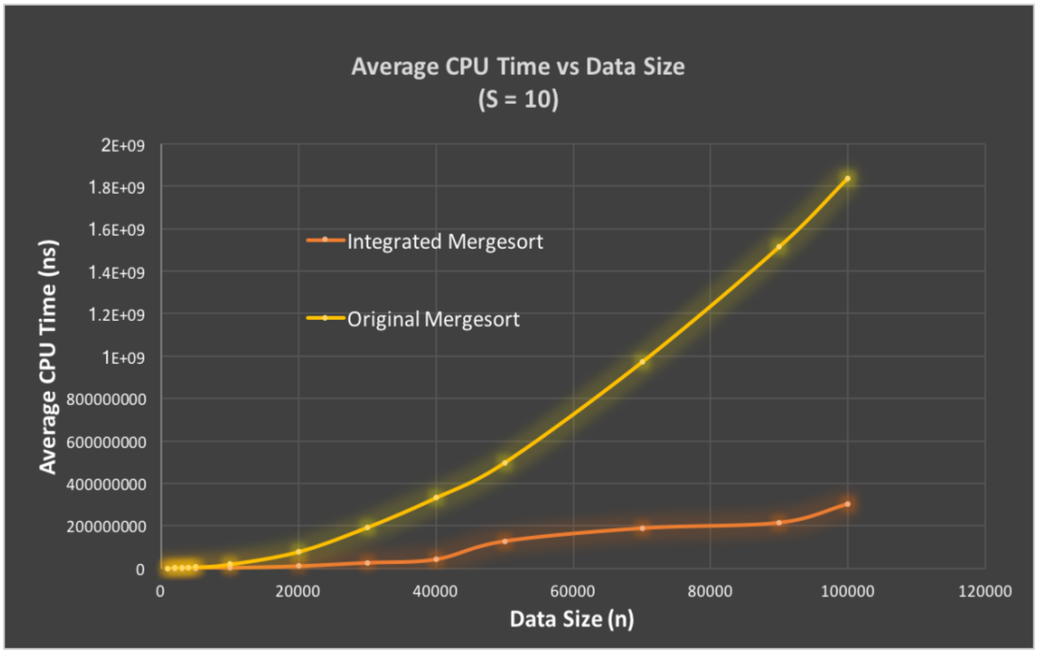

Figure 2. Graph to show the rate of increase in the average CPU time between two Merge Sort algorithms

Additionally, with small data sets, Insertion Sort is more efficient because it only requires a constant amount O(1) of additional memory space, whereas Merge Sort requires additional memory to store the elements in additional array. Therefore, the modified Merge Sort is a more efficient algorithm than the original Merge Sort, especially when the array size is large.

4. Performance Evaluation of Different Values of S

In this section, the performance of the modified Merge Sort is analysed using different values of **S**, so that the optimal threshold value can be determined. Table 2 shows the performance statistics, in terms of the average CPU time, for varying data size, **n**, and S values.

Table 2. Average CPU Time Statistics with Different Values of S

S	n = 1000	n = 2000	n = 3000	n = 10000	n = 20000	n = 30000
10	290717.818	650350.442	452782.142	2111839.14	7800079.278	2.29E+07
20	160053.288	114270.282	215241.58	1107544.166	4122329.076	1.18E+07
30	125820.846	336695.23	116877.076	1109306.764	4063772.518	6102265.502
50	39876.498	233104.088	77903.932	615820.198	2123025.978	6139675.204
100	21770.836	69861.168	55332.444	347530.822	1208394.234	3210872.4
120	15044.432	67686.622	56058.426	370529.628	1157934.25	1820917.824
130	11690.846	32254.456	58029.41	368921.774	1170838.486	1.76E+06
140	12747.806	41122.02	56733.444	369031.616	1212798.198	1.73E+06
150	11380.974	39467.206	52887.002	349441.064	1205899.792	1.78E+06
160	10965.794	42050.192	47776.056	229682.412	703085.006	1.76E+06
170	10365.268	37253.734	47856.6	222100.57	695771.524	1.77E+06
180	13295.342	29398.49	52961.45	236935.682	7.03E+05	1.79E+06
190	20431.972	27208	37966.68	240613.448	6.77E+05	1.70E+06
200	25170.118	25326.776	39006.332	225918.174	7.08E+05	1.79E+06
300	15021.12	28402.34	38740.538	261963.178	7.05E+05	1.06E+06

Based on this statistics, the curves on Figures 3 and 4 are obtained.

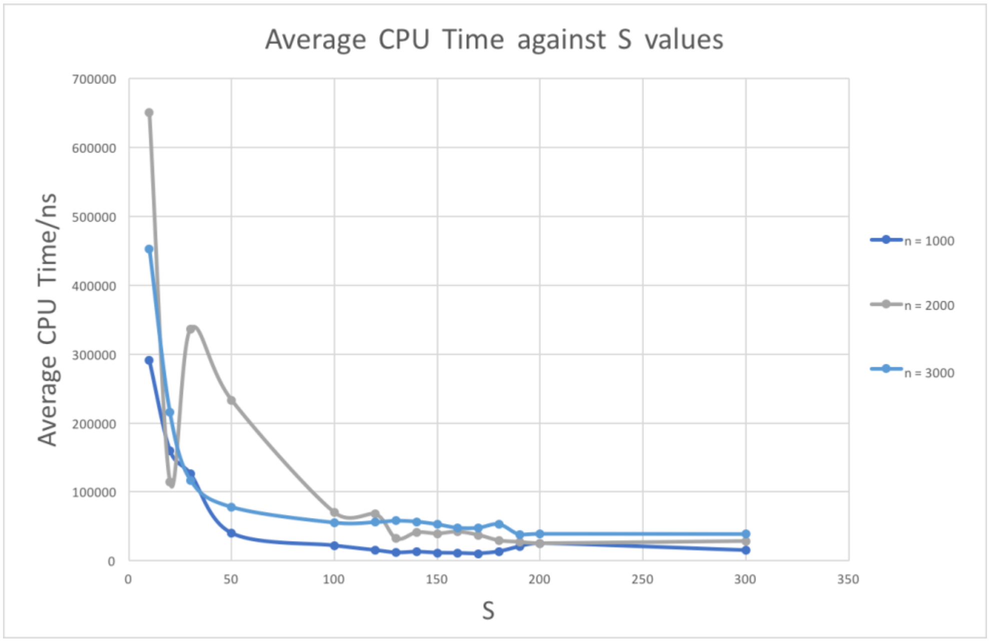

Figure 3. Graph of the average CPU time of the modified Merge Sort with different values of S for n = 1000-3000

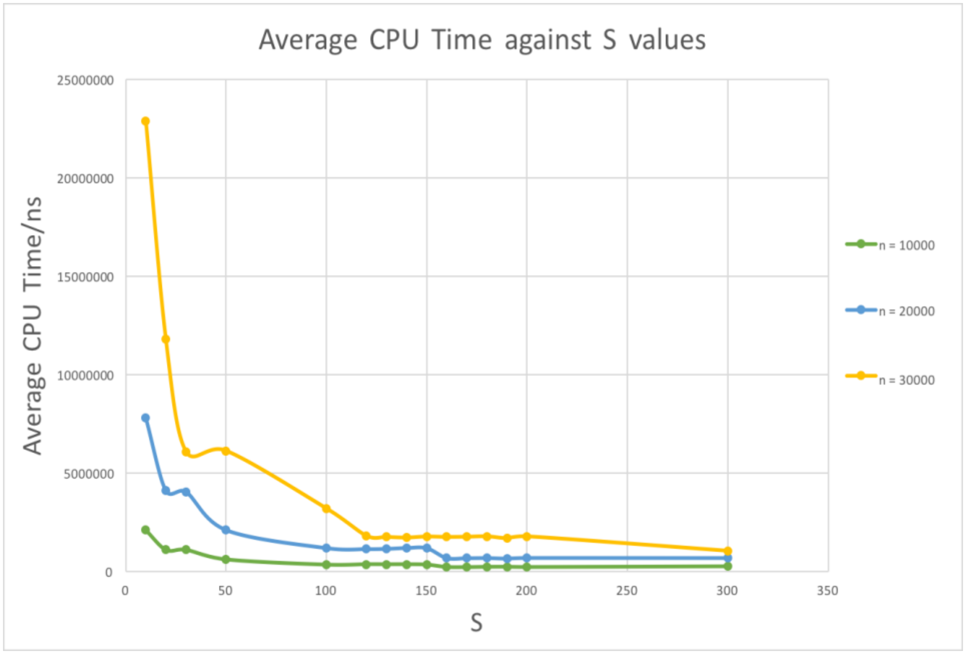

Figure 4. Graph of the average CPU time of the modified Merge Sort with different values of S for n = 10000-30000

From both figures, it can be concluded that when **n** is large and **S** is small, the average CPU time is the highest. As **S** increases from 10 to 100, the average CPU time decreases rapidly, especially for larger **n** values. The average CPU time is then seen to be lowest when **S** is between 130 and 170 for **n** = 1000-3000, and between 120 and 180 for **n** = 10000-30000. Thus, from this observation we would like to conclude that the optimal threshold value is 150.

5. Conclusion

In conclusion, modified Merge Sort algorithm, which is a combination of Merge Sort and Insertion Sort, is more efficient than the original Merge Sort in sorting array with a large size. It is because over a large array size, the average number of key comparison, as well as the average CPU time for the modified Merge Sort algorithm are lower than those of the original Merge Sort. Moreover, as the modified Merge Sort incorporates a threshold value of **S**, it has been observed that 150 is the optimal threshold value to achieve the best sorting performance in terms of the lowest average CPU time.