CSCI 2720
Data Structures - Assignment 5 Report

Kavya Ahuja - 811129168
Dev Patel - 811262610

Algorithm	Input Type	# of Comparisons	Comments about time complexity and # of comparisons
Selection Sort	Ordered.txt	49995000	Time Complexity : O(N^2). Same amount of comparisons as Random and Reversed
Selection Sort	Random.txt	49995000	Time Complexity : O(N^2). Same amount of comparisons as Ordered and Reversed
Selection Sort	Reversed.txt	49995000	Time Complexity : O(N^2). Same amount of comparisons as Random and Ordered.
Merge Sort	Ordered.txt	69008	Time Complexity : O(Nlog2 N).
Merge Sort	Random.txt	120414	Time Complexity : O(Nlog2 N). Most number of comparisons for this Sorting algorithm
Merge Sort	Reversed.txt	64608	Time Complexity : O(Nlog2 N). Least number of comparisons.
Heap Sort	Ordered.txt	503704	Time Complexity : O(Nlog2 N). Most number of comparisons for this Sorting algorithm
Heap Sort	Random.txt	482384	Time Complexity : O(Nlog2 N)
Heap Sort	Reversed.txt	461738	Time Complexity : O(Nlog2 N). Least number of comparisons for this Sorting algorithm
QuickSort-fp	Ordered.txt	49995000	Time Complexity : O(N^2). Same amount of comparisons as Reversed
QuickSort-fp	Random.txt	148343	Time Complexity : O(Nlog2 N). Least number of comparisons for this Sorting algorithm
QuickSort-fp	Reversed.txt	49995000	Time Complexity : O(N^2). Same amount of comparisons as Ordered.
QuickSort-rp	Ordered.txt	25123552	Time Complexity : O(Nlog2 N). Most number of comparisons for this Sorting algorithm

QuickSort-rp	Random.txt	151572	Time Complexity : O(Nlog2 N). Least number of comparisons for this Sorting algorithm
QuickSort-rp	Reversed.txt	18202010	Time Complexity : O(Nlog2 N).

We used extra memory space/other data structures in the Merge function that is then used by MergeSort. In this we used a temporary array that holds the values that are then used to be set in the actual input array.

The chart above states the Number of Comparisons for each Algorithm for three input types: ordered, random, and reversed text files. From the chart, we were able to determine how each sorting algorithm reacts with different text files. For Selection Sort, all three text files had the same number of comparisons. This means selection sort is able to sort all text files at the same time. For Merge sort, the ordered and the reversed file had similar numbers of comparisons. However, the random file took double the number of comparisons to get sorted. Heap had the most number of comparisons for the ordered text file. For both quick sorts ordered and reversed took the most number of swaps. From this information, we made a chart based on which sorting algorithm performed best on which file.

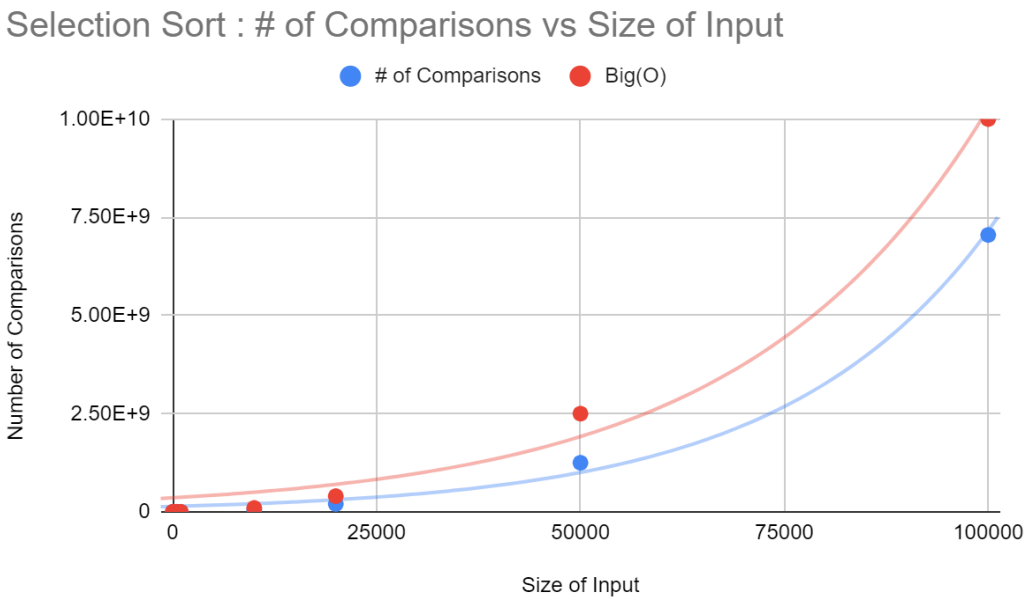
Algorithm	Most Comparisons	Least Comparisons
Selection Sort	All were same	All were same
Merge	Random.txt	Reversed.txt
Heap	Ordered.txt	Reversed.txt
Quicksort-fp	Ordered.txt & Reversed.txt	Random.txt
Quicksort-rp	Ordered.txt	Random.txt

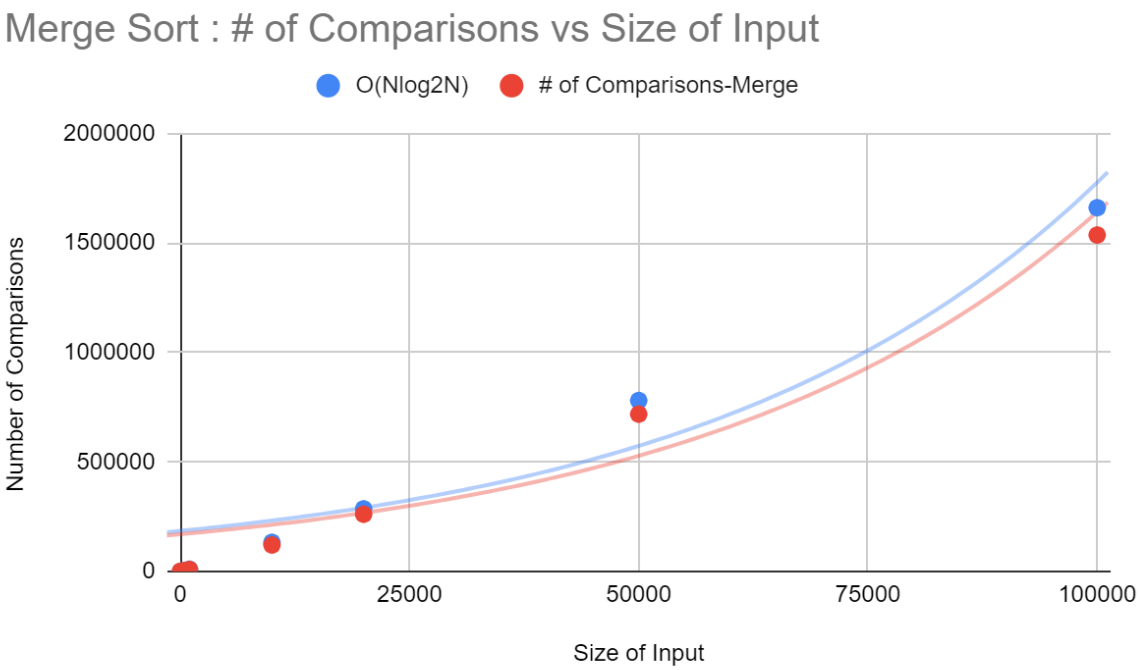
From this chart, it can be concluded that a Merge and Heap sort is the best to use with a reversed ordered file. Additionally, Quicksort is best to use with a randomized order file.

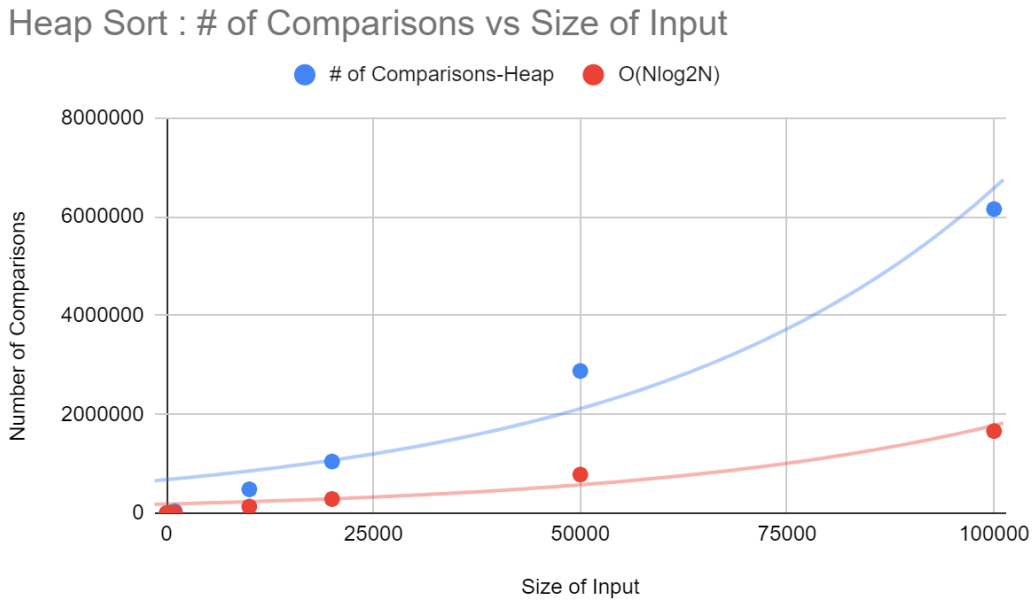
Size of Input vs # of Comparisons

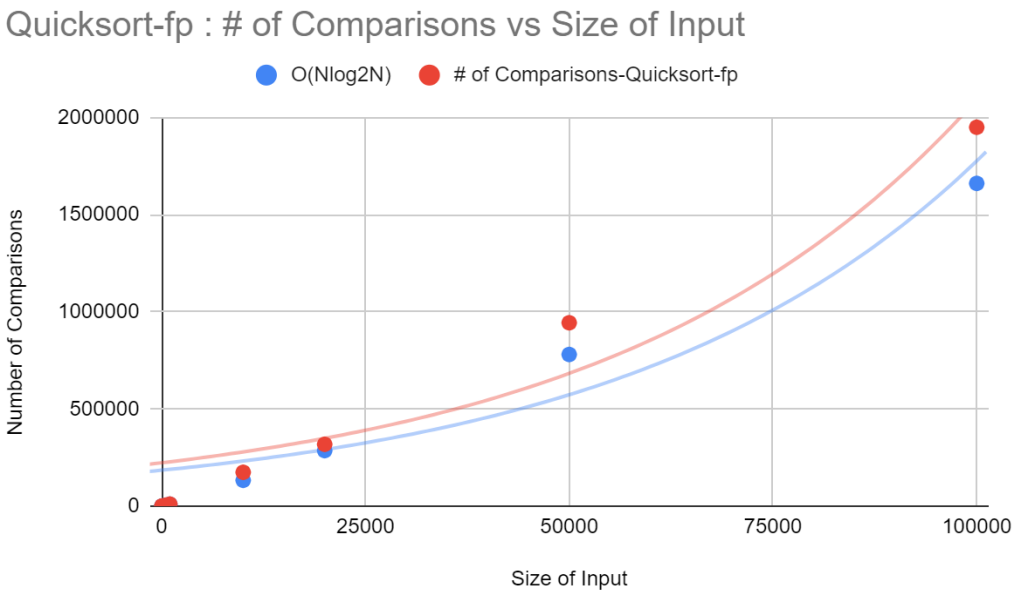
Algorithm	10	100	500	1000	10000	20000	50000	100000
Selection	45	4950	124750	499500	49995000	199990000	1249975000	70498270400
Merge	23	546	3834	8684	120534	261019	718127	1536329
Heap	92	2148	15441	35014	482329	1044722	2877791	6154885
Quicksort-fp	27	647	5023	10421	174072	318155	943700	1949928
Quicksort-rp	23	613	4413	10010	162623	358687	993750	2034667

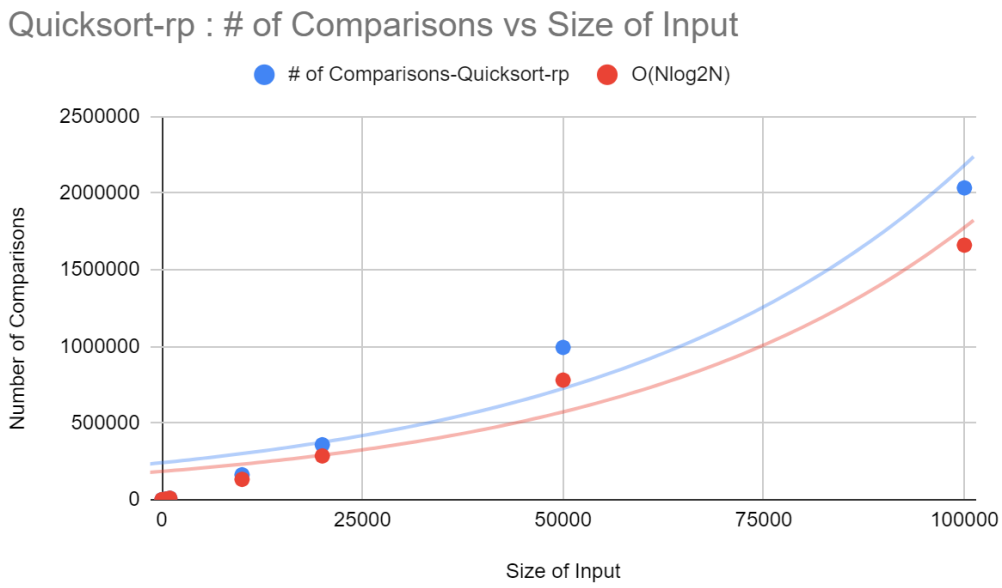
From the chart above, it can be determined that Selection takes more comparisons compared to the other sorting algorithm. This can be due to the Big O complexity difference. Selection sort has a Big O complexity of O(N^2) versus the other sorting algorithms that have a complexity of O(Nlog2 N). The best sorting algorithm seems to be Merge, due to it having the least number of comparisons. Both quick-sort algorithms have similar numbers of comparisons. The graphs below show the same information as the chart.




Merge Sort : # of Comparisons vs Size of Input
O(Nlog2N)
# of Comparisons-Merge
2000000
1500000
1000000
500000
0
Number of Comparisons
0
25000
50000
75000
100000
Size of Input


Heap Sort : # of Comparisons vs Size of Input
# of Comparisons-Heap
O(Nlog2N)
8000000
6000000
4000000
2000000
0
Number of Comparisons
0
25000
50000
75000
100000
Size of Input





The graphs show the number of comparisons versus the size of the input. The graphs also have the time complexity graph to compare to the experiment results. The time complexities graph is similar to the results of our experiment. Merge sort's graph is most similar to time complexity. The Heap sort graph is the least similar to the time complexity. The inconsistency could be due to the randomized data. Based on the graph, it is determined that Merge sort is the best sorting algorithm to use, overall. It is because it takes the least amount of comparisons.