Assignment 5

Data Structures

CSCI-2720c

Prepared By:
Anjiya Kazani, John Gunerli
20 November 2021

<u>Table 1:</u>

Algorithm Type	Input Type	Comparisons	Comments
Selection sort	Ordered	49995000	The time complexity of an ordered selection sort is $O(n^2)$. Expect an output of 8-9 digits which aligns with the Big-O time complexity of this algorithm.
Selection sort	Random	49995000	The time complexity of a random selection sort is $O(n^2)$. Expect an output of 8-9 digits which aligns with the Big-O time complexity of this algorithm.
Selection sort	Reversed	49995000	The time complexity of a reversed selection sort is $O(n^2)$. Expect an output of 8-9 digits which aligns with the Big-O time complexity of this algorithm.
Merge sort	Ordered	69008	The complexity for an ordered Merge Sort is O(nlogn) for all input cases. Expect an output of 5-6 digits. This aligns with the Big-O time complexity.
Merge sort	Random	120414	The complexity for a random Merge Sort is O(nlogn) for all input cases. Expect an output of 5-6 digits. This aligns with the Big-O time complexity.
Merge sort	Reversed	64608	The complexity for anReversed Merge Sort is O(nlogn) for all input cases. Expect an output of 5-6 digits. This aligns with the Big-O time complexity.

Heap sort	Ordered	259244	The complexity for an ordered Heap Sort is O(nlogn) for all input cases. Expect an output of 5-6 digits. This aligns with the Big-O time complexity.
Heap sort	Random	246954	The complexity for a random Heap Sort is O(nlogn) for all input cases. Expect an output of 5-6 digits. This aligns with the Big-O time complexity.
Heap sort	Reversed	235056	The complexity for a reversed Heap Sort is O(nlogn) for all input cases. Expect an output of 5-6 digits. This aligns with the Big-O time complexity.

Quick -FP	Ordered	50004999	The complexity for an ordered Quick Sort-FP is $O(n^2)$ for all input cases. Expect an output of 8-9 digits. This aligns with the Big-O time complexity.
Quick -FP	Random	159534	The complexity for a random Quick Sort-FP is O(nlogn) for all input cases. Expect an output of 5-6 digits. This aligns with the Big-O time complexity.
Quick -FP	Reversed	49999999	The complexity for a reversed Quick Sort-FP is $O(n^2)$ for all input cases. Expect an output of 8-9 digits. This aligns with the Big-O time complexity.

Quick - RP	Ordered	14346045	The complexity for an ordered Quick Sort-RP is $O(n^2)$ for all input cases. Expect an output of 8-9 digits. This aligns with the Big-O time complexity.
Quick - RP	Random	50436	The complexity for a random Quick Sort-RP is O(nlogn) for all input cases. Expect an output of 5-6 digits. This aligns with the Big-O time complexity.
Quick - RP	Reversed	11328799	The complexity for a reversed Quick Sort-RP is $O(n^2)$ for all input cases. Expect an output of 8-9 digits. This aligns with the Big-O time complexity.

Table 1 Analysis:

a. Provide the total number of comparisons used by each one of the algorithms and explain whether they align with the Big O time complexity of each algorithm. In case of quick sort, compare and comment about the number of comparisons and complexity with the other quicksort implementations.

- For O(nlogn) complexities we expect an outcome of 5-6 digits and an outcome of 8-9 digits with $O(n^2)$ complexities. This is because our size of input is 10,000 so for $O(n^2)$, n^2 will be 100,000 and for O(nlogn), nlogn would be 40,000. As far as Quick sort is concerned, the complexity for FP and RP algorithms for all three input types were the same however, RP for all three was significantly faster and had lesser comparisons then that of Quick Sort FP. This is because Random Pivot minimizes the probability of bad splits.

b. Did you use extra memory space or other data structures other than the input array? If so, explain where and why?

- We did not use any other data structures other than input arrays.

c. Explain what sorting algorithms work best in what situations based on your experimental results.

- MergeSort was the most efficient algorithm based on it's performance on all cases because it had a complexity of O(nlogn). Even though HeapSort also had a complexity of O(nlogn) on all three input types, MergeSort had 5 digits comparisons for ordered and reversed types compared to that of HeapSort which were 6. Even though they both had 6 digits for random input type, MergeSort was still faster. Overall, MergeSort is the most efficient compared to other algorithms.

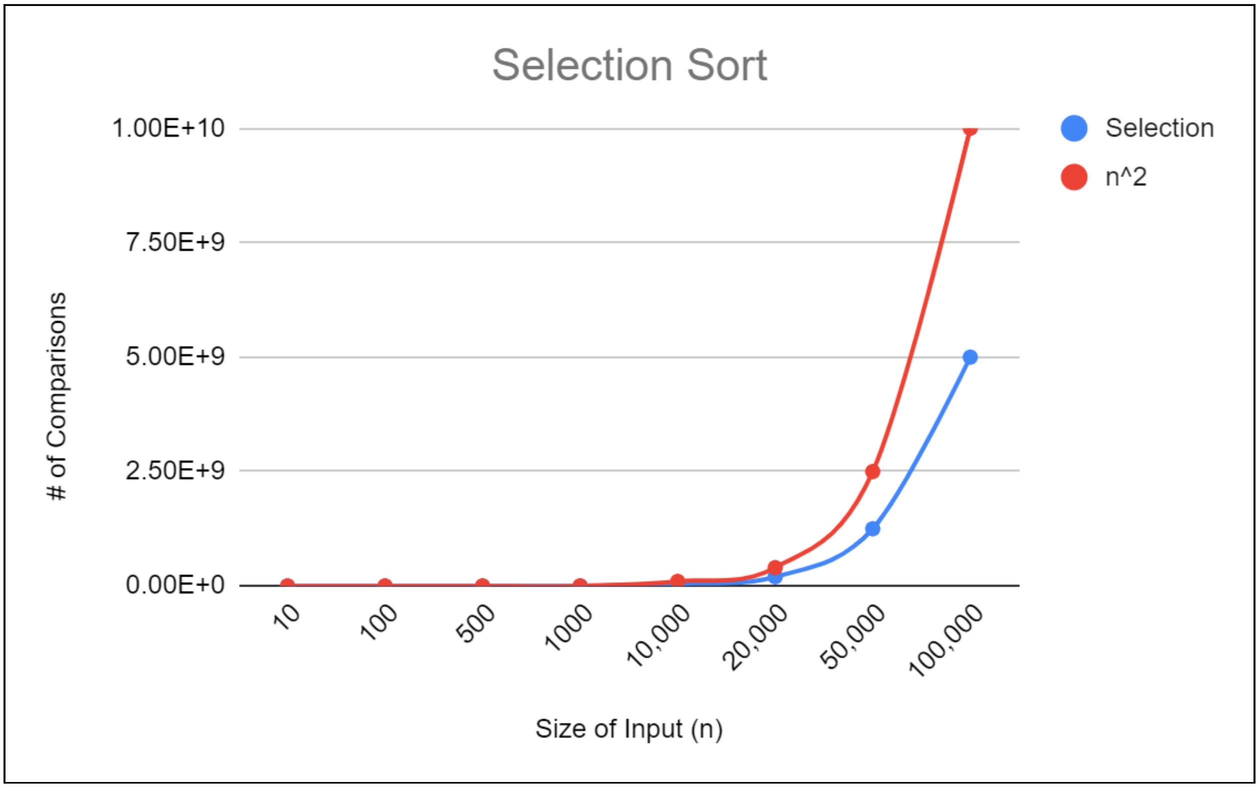
Table 2:

Algorithm	**10**	**100**	**500**	**1000**	**10,000**	**20,000**	**50,000**	**100,000**
Selection	45	4950	124750	499500	49995000	199990000	1249975000	4999950000
Merge	22	545	3834	8675	120442	260897	718081	1536350
Heap	51	1159	7986	18059	246720	533565	1467092	3134862
QuickSort-fp	29	650	5486	10663	158019	364354	975656	2313451
QuickSort-rp	17	224	89502	77536	258276	361834	687429	1739405

Worst Case Table:

n^2	10	1000	25000	1000000	100000000	400,000,000	2500000000	10000000000
Nlog(n)	33	664	4482	9965	132877	285754	780482	1660964

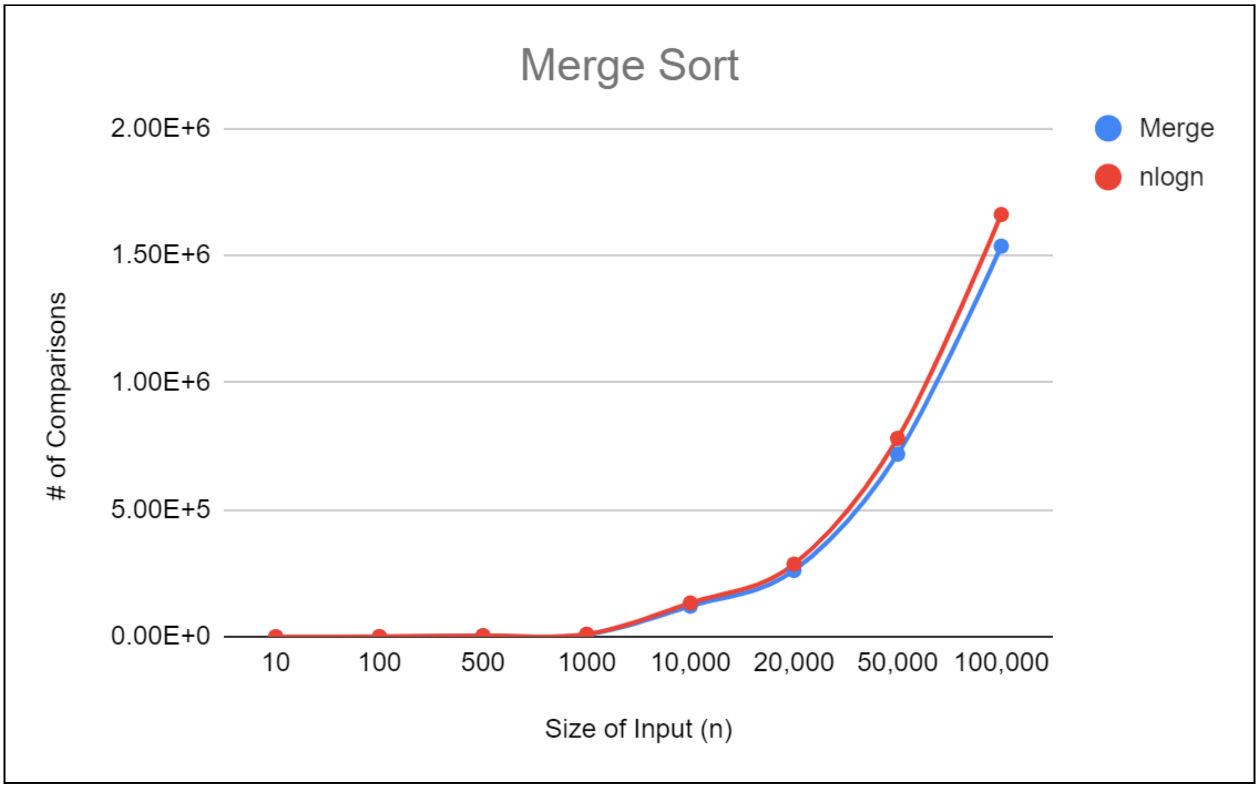
i. Selection Sort



Discussion:

In the figure above, the red line represents the worst-case time complexity for Selection Sort which is $O(n^2)$. The blue line represents the results of our experiments with Selection sort. The results, theoretical and experimental, align for the Selection Sort algorithm. The experimental results follow closely with the worst case complexity and because the red line is the worst case, we expect our experimental lines average to be less than or equal to the $O(n^2)$ line. Selection sort had a complexity of $O(n^2)$ in all input cases. No inconsistencies were found in the graphing of Selection Sort.

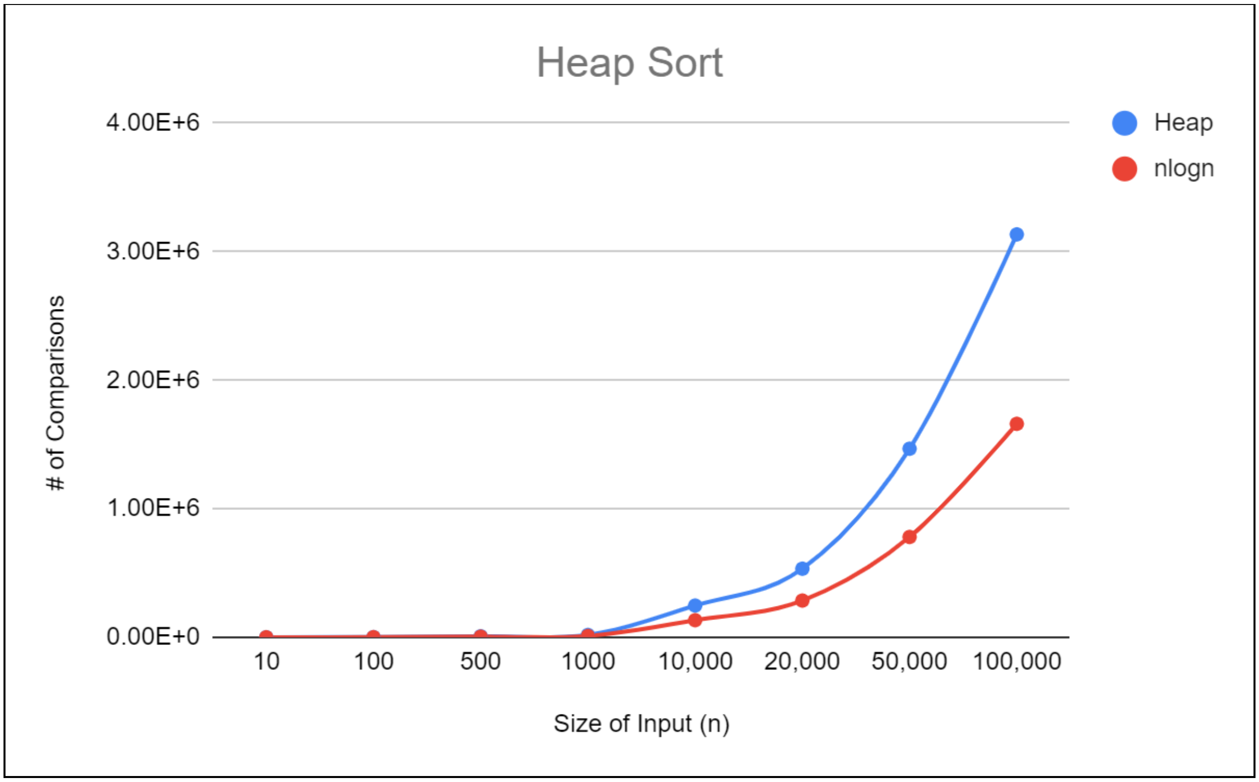
ii. Merge Sort



Discussion:

In the figure above, the red line represents the worst-case time complexity for Merge Sort which is O(nlogn). The blue line represents the results of our experiments with Merge sort. The results, theoretical and experimental, align for the Merge Sort algorithm and follow very closely with the nlogn curve. We expect our experimental lines average to be less than or equal to the O(nlogn) line; this proves that Merge sort has a complexity of O(nlogn) in all input cases and achieves its maximum efficiency with the 3 input types. No inconsistencies were found in this.

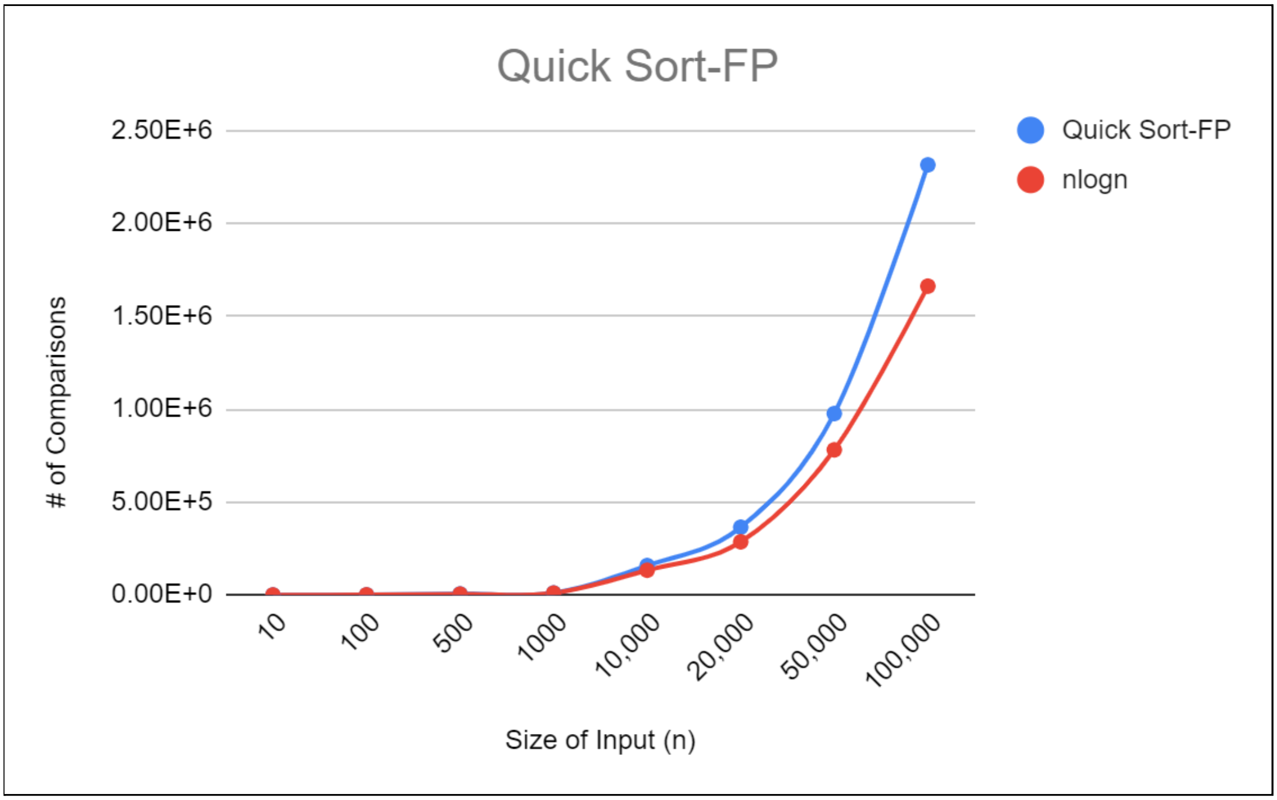
iii. Heap Sort



Discussion:

In the figure above, the red line represents the worst-case time complexity for Heap Sort which is O(nlogn). The blue line represents the results of our experiments with Heap sort. The results, theoretical and experimental, don't fully align for our Merge Sort algorithm but still follows closely with the nlogn curve. We expect our experimental line's average to be less than or equal to the O(nlogn) line; this graph however displays some inconsistencies because the Heap Sort line is above the worst case complexity. This is because in randomization there could be an issue in creating duplicate values and our Heap Sort algorithm doesn't check for those.

iv. Quick Sort-FP



Discussion:

In the figure above, the red line represents the worst-case time complexity for Quicksort-FP which is O(nlogn) and the blue line represents the results of our experiments with QuickSort-FP. Even though the two lines are close in alignment, our QuickSort line is above the nlogn line which means our algorithm is slightly worse then the theoretical complexity therefore inconsistent. The reason for inconsistency could be because the first value could be smaller than the rest which would make partitioning into smaller arrays more difficult therefore this algorithm achieves maximum efficiency for random data only. When given ordered or reversed data, the algorithm has a $O(n^2)$ complexity making it less efficient. Overall, the experimental results coincide with the average Big-O complexity O(nlogn).

v. Quick Sort-RP

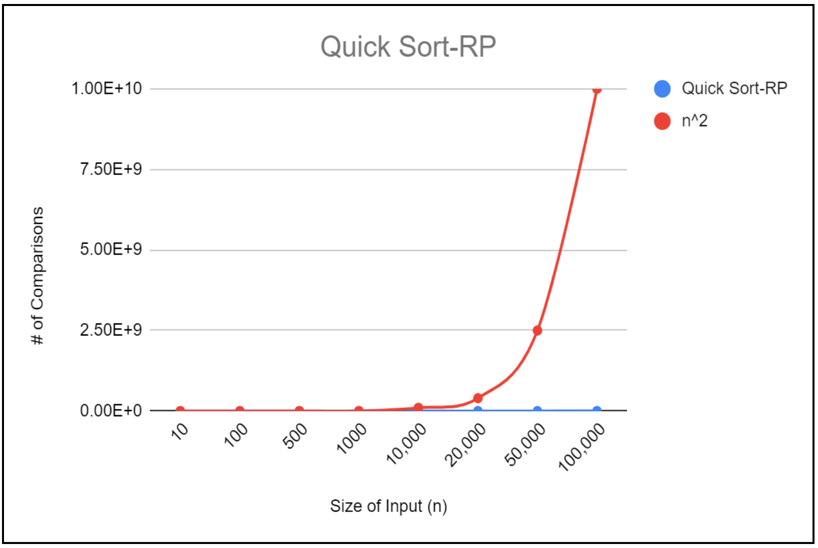


Figure A (worst case)

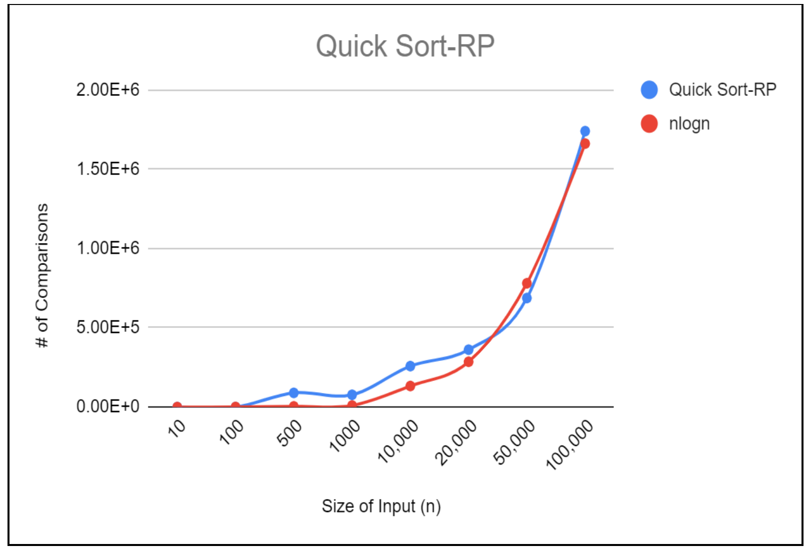


Figure B (avg case)

Discussion:

In the figures above, the red line represents the average (nlogn) and worst (n^2) time complexity for Quicksort-RP and the blue line represents the results of our experiments with QuickSort-RP. In Figure A, our algorithm proves to be more efficient than the worst case however in Figure B, our algorithm closely aligns with the O(nlogn) line. This is because in a random pivot, it ensures the possibility of getting the worst case complexity is lower, making it closer to the nlogn line. Quick Sort - RP algorithm achieves maximum efficiency for random data only. When given ordered or reversed data, this algorithm is less efficient with a O(n2) complexity. Overall, the experimental results coincide with the average Big-O complexity O(nlogn).