Lab Assignment #2

Comparison of Sorting Algorithms

Name: Bhavyai Gupta
UCID: 30143691

Data Collected

Our sorting algorithms, by default, sort the items in ascending order. The Worst Case, Average Case, and the Best Case are determined by the initial ordering of the input array.

1. Worst Case

When the input array is sorted in descending order (aka reverse sorted), it is said to be the Worst Case. Table 1 summarizes the timings noted for different algorithms for different sizes of the input array.

		Time (ms) for sorting a reverse sorted array (Worst Case)			
Algorithm		Bubble	Insertion	Merge	Quick
Input Array Size	10	1.3539	1.3587	1.4559	1.4112
	100	1.7671	1.5922	1.601	1.6677
	1,000	8.6207	7.0343	2.8795	6.1857
	10,000	77.2967	33.0846	6.0928	17.0152
	100,000	6339.2764	1559.2708	20.5492	142.8342
	1,000,000	753806.3951	273967.6684	91.4103	StackOverflow

Table 1: Time (ms) required for sorting a reverse order array by different sorting algorithms.

2. Average Case

When the input array is in random order (aka unsorted), it is said to be the Average Case. Table 2 summarizes the timings noted for different algorithms for different sizes of the input array.

		Time (ms) for sorting an unsorted array (Average Case)			
Algorithm		Bubble	Insertion	Merge	Quick
Input Array Size	10	0.9104	1.3743	1.4369	1.2684
	100	2.8864	1.4483	1.4503	1.3813
	1,000	8.8523	5.5577	2.9804	2.0384
	10,000	150.6983	17.4492	6.3174	4.8029
	100,000	14663.6907	742.8939	19.4296	22.836
	1,000,000	1665448.87	86110.6134	146.7231	353.5663

Table 2: Time (ms) required for sorting a random array by different sorting algorithms.

3. Best Case

When the input array is already sorted in ascending order (aka sorted), it is said to be the Best Case. Table 3 summarizes the timings noted for different algorithms for different sizes of the input array.

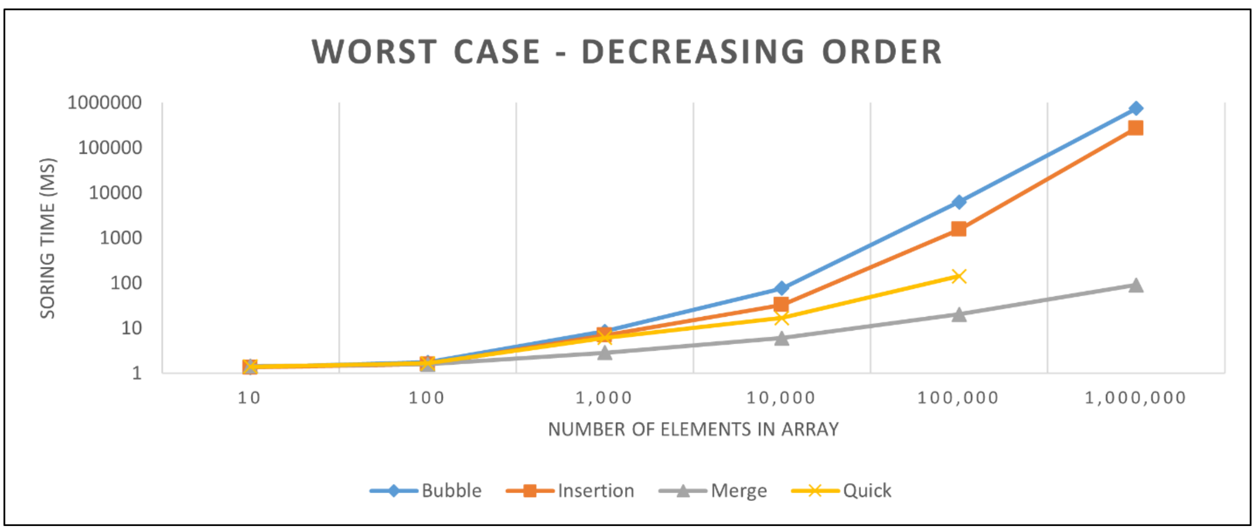
		Time (ms) for sorting a sorted array (Best Case)			
Algorithm		Bubble	Insertion	Merge	Quick
Input Array Size	10	1.3279	1.3867	1.4225	1.4307
	100	1.6232	1.3422	1.5331	1.7311
	1,000	7.284	1.4283	2.8725	7.8661
	10,000	28.6788	1.9876	5.2422	27.1456
	100,000	1196.0227	5.7565	21.941	StackOverflow
	1,000,000	115733.3627	10.1188	80.8772	StackOverflow

Table 3: Time (ms) required for sorting a sorted order array by different sorting algorithms.

Data Analysis

Based on the data collected in Tables 1, 2, and 3, we plot graphs to analyze the data.

1. Worst Case



Graph 1: Time (ms) required for sorting a reverse order array by different sorting algorithms.

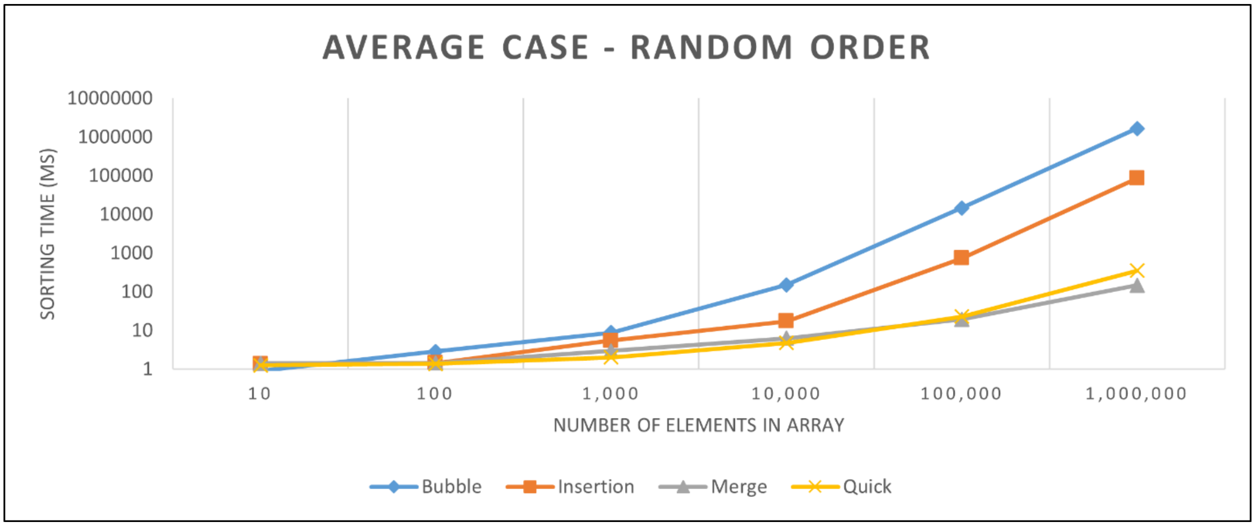
<u>Bubble Sort</u>: performs well for smaller input size (size up to 10). It becomes the worst performing algorithm when the input size increases.

<u>Insertion Sort</u>: performs the best for smaller input size (size up to 100). In general, it has a better performance than Bubble Sort.

<u>Quick Sort</u>: performs better than Insertion Sort only for large input size (ignoring the StackOver error due to naive implementation).

Merge Sort: shines as the best sorting algorithm for large input sizes.

2. Average Case



Graph 2: Time (ms) required for sorting a random array by different sorting algorithms.

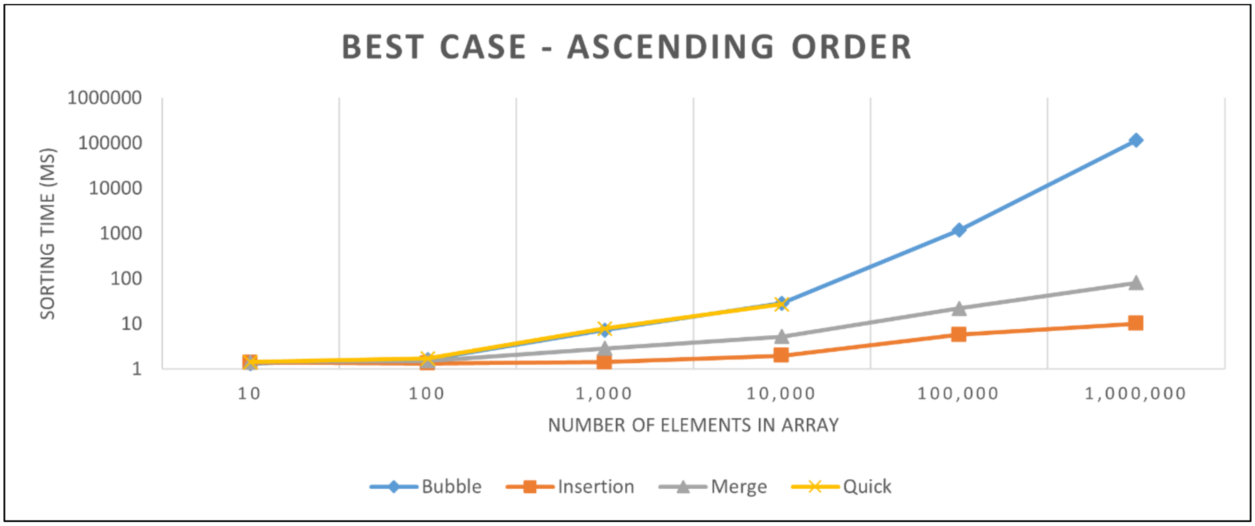
Bubble Sort: performs well for smaller input size (size up to 10). It becomes the worst performing algorithm when the input size increases.

Insertion Sort: performs well for smaller input size (size up to 100). In general, it has a better performance than Bubble Sort.

Quick Sort: performs better than Insertion Sort for every input size. It performs even better than Merge Sort for sizes up to 10,000.

Merge Sort: shines as the best sorting algorithm for large input sizes.

3. Best Case



Graph 3: Time (ms) required for sorting a sorted order array by different sorting algorithms.

Bubble Sort: performs well for smaller input size (size up to 10). Its performance decreased when the input size increases.

Insertion Sort: shines as the best sorting algorithm for every input size.

Quick Sort: performs comparably to Bubble Sort for input size up to 10,000.

Merge Sort: performs better than Bubble Sort and Quick Sort.

Complexity Analysis

With the help of asymptotic analysis, we find the below Time Complexities of the algorithms in different cases.

1. Worst Case Time Complexities

Bubble Sort : $O(n^2)$
Insertion Sort : $O(n^2)$
Quick Sort : $O(n^2)$
Merge Sort : $O(n \log(n))$

2. Average Case Time Complexities

Bubble Sort : $O(n^2)$
Insertion Sort : $O(n^2)$
Quick Sort : $O(n \log(n))$

Merge Sort : O(n log(n))

3. Best Case Time Complexities

Bubble Sort : $O(n^2)$
Insertion Sort : O(n)
Quick Sort : O(n log(n))
Merge Sort : O(n log(n))

Interpretation

Bubble Sort and Insertion Sort have same Time Complexities in worst and average case. But, in general, the Insertion Sort performs a lot better than Bubble Sort.

Quick Sort and Merge Sort have same Time Complexities in average case and best case. Empirical data shows Quick Sort works better in average case for input sizes up to 10000, while Merge Sort works better in the best case for every input size.

Conclusions

- The typical implementation of the Bubble Sort is the worst performing algorithm and should be avoided for sorting.
- Insertion Sort should be the preferred algorithm when input size is smaller or when the input is partially sorted.
- Either Quick Sort or Merge Sort could be used when input size is larger.
- Merge Sort is not an in-place sorting algorithm as it requires an O(n) extra space to work with. This might cause some problems in some systems where memory is limited.
- Quick Sort is an unstable but in-place and faster algorithm (faster than Merge Sort in average case up to a size of 100,000). However, its performance is implementation dependent - it depends on how the partitions are chosen in the algorithm.