Run-Time Analysis of Sorting Algorithms

Abhishek Kumar

Abstract

In this report, a run-time comparative analysis is presented. It consists in evaluating the run-times of four well-known sorting algorithms: BubbleSort, InsertionSort, SelectionSort and MergeSort.

Each of the sorting functions with the following inputs were tested. (1) A small array with 10-15 elements, (2) A large array with at least 10000 elements, (3) A large sorted array and (4) Partially sorted large array with the first 5% and the last 5% elements sorted. All the algorithms were implemented in C++ programming language. The empirical results show that the fastest sorting algorithm is MergeSort, followed by InsertionSort, then SelectionSort and finally BubbleSort. The observation conforms to the theoretical time complexity.

Introduction

Complexity is the measure by which an algorithm can be evaluated in comparison with other algorithms that solve the same problem.

Evaluating the complexity of the algorithm will assume the estimation of:

- Time complexity: number of executions of a certain statement.
- Space complexity: the amount of supplementary memory needed.

The following table shows the comparison of sorting algorithms.

Table 1: Comparison of algorithms

Algorithm	Time Complexity		
	Best	Average	Worst
Bubble sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Insertion sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Selection sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$
Merge sort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$

Results

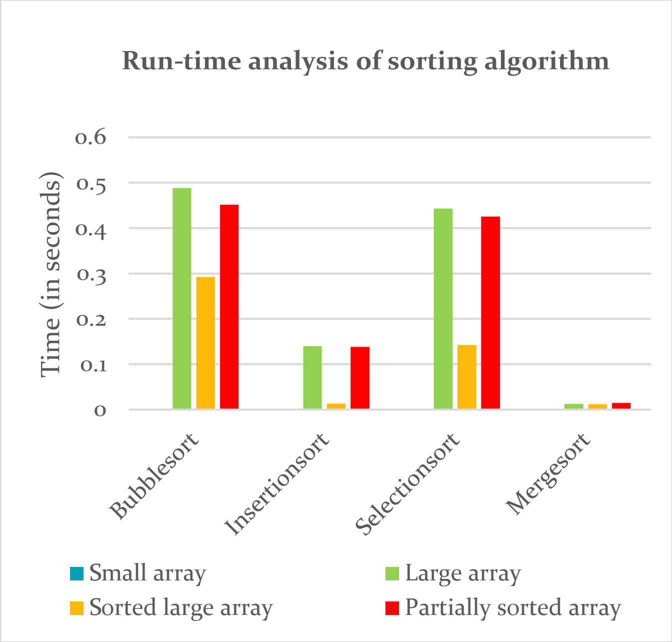
Below is the documentation of the running time of the respective algorithms for specific input instances. Here, we take the size of the array to be n = 15 for small array and n = 10,000 for large array.

Table 2: Run-time documentation of input instances for various algorithms

small array (n = 15) large array (n = 10,000)	Small array (Time in sec)	Large array (Time in sec)	Sorted large array (Time in sec)	Partially sorted array (Time in sec)
Bubble sort	3 x 10^{-5}	0.4880	0.2919	0.4514
Insertion sort	1.6 x 10^{-5}	0.1392	0.0130	0.1380
Selection sort	1.5 x 10^{-5}	0.4430	0.1425	0.4251
Merge sort	1.1 x 10^{-5}	0.0122	0.0117	0.0150

On the basis of our run-time for different input instances, we plot a graph with respect to the time taken by various algorithms.

Figure 1: Run time analysis of arrays of various sizes



Conclusion

The empirical evaluations presented in this report confirmed the theoretical complexities of the four sorting algorithms: the most efficient algorithm is Merge sort followed by Insertion sort, then Selection sort and finally Bubble sort.