Comparison of Time Efficiency Classes of the Sort Algorithms

Syed Abdullah Hassan

21801072

CTIS 264

16.05.2020

Executive Summary

This report compares the time efficiency classes of sort algorithms. It will first discuss the theoretical part of the algorithms, including the average, best, and worst case for each of the algorithms in the report. Then it will be followed by empirical data taken from implementation of the algorithms and using them to sort three different types of test arrays. First being randomly generated array, second being an already sorted array, finally third being a sorted array but in reverse order. The report will analyze how each of the algorithms deal with each of the test cases. The report will also analyze the change in speed of each of the sorting algorithms when the size of the test data is varied. The algorithms will be analyzed for large test data and for small test data and the speed of the sorting will be compared. For the scope of this report, Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Quick Sort, and Heap Sort will be compared.

1. Selection Sort

Being a simple sorting algorithm, it works extremely well for small files. It has the time efficiency: Ω(n^2), θ(n^2), and O(n^2). Each item in this algorithm is moved at most no more than once [4]. Because of O(n^2) time complexity, it is inefficient in sorting large lists of data. However, selection sort is still the algorithm that does the least amount of data moving. It might be a better choice in data sets where the data is considerably larger than the keys [8].

#	Number of Elements	Running Time(s)	Dataset Order
1	1000	0.0676	Ordered
2	10,000	16.8130	Ordered
3	1000	0.2668	Random
4	10,000	16.7893	Random
5	1000	0.2714	Reverse
6	10,000	17.0534	Reverse

2. Bubble Sort

Bubble sort is an algorithm that repeats; it is comparing each pair of adjacent items and swapping them if they are in the wrong order. It repeats until no swaps are required, indicating that the list is sorted [13][23]. It has a O (n2) Time complexity means that its efficiency decreases as list becomes bigger [12][24]. It works at Ω(n) in its best case. Bubble sort is a good choice for sorting small arrays.

#	Number of Elements	Running Time(s)	Dataset Order
1	1000	0.0000	Ordered
2	10,000	0.0066	Ordered
3	1000	0.4306	Random
4	10,000	32.1357	Random
5	1000	0.7128	Reverse
6	10,000	44.2858	Reverse

3. Insertion Sort

Insertion sort's working is close to that of selection sort. It is a simple sorting algorithm that builds the final sorted list one item one by one [18]. It has O(n2) time complexity, it is much less efficient on large lists than algorithms like quick sort, heap sort, or merge sort. However, insertion sort provides several advantages like simple implementation and efficient for small data sets [10][17].

#	Number of Elements	Running Time(s)	Dataset Order
1	1000	0.0009	Ordered
2	10,000	0.0074	Ordered
3	1000	0.2167	Random
4	10,000	14.5801	Random
5	1000	0.3658	Reverse
6	10,000	28.7096	Reverse

4. Merge Sort

Merge sort is a divide and conquer algorithm. It divides the list into two equal sub lists, then sorts the sub lists recursively [19]. It has an O (n log n) time complexity. Merge sort is a stable sort and is more efficient at handling slow-to-access sequential media. Merge sort is often the best choice for sorting a linked list [11][20]. The best case Ω (n log(n)), average case θ(n log(n)), and worst case O(n log(n)) of the algorithm. This algorithm works good for worst cases.

#	Number of Elements	Running Time(s)	Dataset Order
1	1000	0.0103	Ordered
2	10,000	0.1647	Ordered
3	1000	0.0139	Random
4	10,000	0.1763	Random
5	1000	0.0112	Reverse
6	10,000	0.1473	Reverse

5. Quick Sort

Pivot is used in quick sort and that element is fixed in its place by moving all the elements less than that to its left and all the elements greater than that to its right. Since it partitions the element sequence into left, pivot and right. This is called sorting by partitioning. It is an O (n log n) Time complexity in average case [21], [22]. Quick sort has Ω(n log(n)) as the best case, θ(n log(n)) as the average case, and O(n^2) the best case.

#	Number of Elements	Running Time(s)	Dataset Order
1	1000	0.0074	Ordered
2	10,000	16.8130	Ordered
3	1000	0.0099	Random
4	10,000	0.1117	Random
5	1000	0.0101	Reverse
6	10,000	0.1011	Reverse

6. Heap Sort

Heap sort uses the heap data structure. Heap is a special tree-based data structure. A binary tree is said to follow a heap data structure if all nodes in the tree follow the property that they are greater than their children i.e. the largest element is at the root and both its children and smaller than the root and so on. Such a heap is called a max-heap. If instead, all nodes are smaller than their children, it is called a min-heap. Heap sort has Ω(n log(n)) complexity for best case, θ(n log(n)) as the average case, and O(n log(n)) as the worst case. That means that it is a good algorithm for sorting bigger data sets.

#	Number of Elements	Running Time(s)	Dataset Order
1	1000	0.0123	Ordered
2	10,000	0.2456	Ordered
3	1000	0.0189	Random
4	10,000	0.0683	Random
5	1000	0.0201	Reverse
6	10,000	0.1031	Reverse

References

[1] P.Adhikari, Review on Sorting Algorithms, "A comparative study on two sorting algorithm", Mississppi state university, 2007.

[2] T. Cormen, C.Leiserson, R. Rivest and C.Stein, Introduction To Algorithms, McGraw-Hill, Third Edition, 2009,pp.15-17.

[3] M. Goodrich and R. Tamassia, Data Structures and Algorithms in Java,John wiley & sons 4th edition, 2010,pp.241-243.

[4] R. Sedgewick and K. Wayne, Algorithms,Pearson Education, 4th Edition, 2011,pp.248-249.

[5] T. Cormen, C.Leiserson, R. Rivest and C.Stein,Introduction to Algorithms, McGraw Hill, 2001,pp.320-330.

[6] H. Deitel and P. Deitel, C++ How to Program, Prentice Hall, 2001,pp.150-170.

[7] M. Sipser, Introduction to the Theory of Computation,Thomson,1996,pp.177-190.

[8] S. Jadoon , S.Solehria, S.Rehman and H.Jan.(2011,FEB). " Design and Analysis of Optimized Selection Sort Algorithm".11. (1),pp. 16-21. Available: http://www.ijens.org/IJECS%20Vol%2011%20Issue%2001.html.

[9] Aditya Dev Mishra & Deepak Garg.(2008,DEC). “Selection of Best Sorting Algorithm ", International journal of intelligent information Processing. II (II).pp. 363-368. Available: http://academia.edu/1976253/Selection_of_Best_Sorting_Algorithm.

[10] http://en.wikipedia.org/wiki/Insertion_sort

[11] http://en.wikipedia.org/wiki/Merge_sort

[12] http://en.wikipedia.org/wiki/Bubble_sort

[13] http://www.techopedia.com/definition/3757/bubble-sort

[14] I. trini, k. kharabsheh, A. trini, (2013,may)."Grouping Comparison Sort", Australian Journal of Basic and Applied Sciences.pp221-228.

[15] R. Sedgewick, Algorithms in C++, Addison–Wesley Longman,1998,pp 273–274.

[16] A. Levitin, Introduction to the Design & Analysis of Algorithms, Addison–Wesley Longman, 2007, pp 98–100.

[17] http://corewar.co.uk/assembly/insertion.html.

[18] T. Cormen, C.Leiserson, R. Rivest and C.Stein, Introduction To Algorithms, McGraw-Hill, Third Edition, 2009,pp.15-21.

[19] Katajainen, Jyrki; Pasanen, Tomi; Teuhola, Jukka (1996,MAR). "Practical in-place mergesort". Nordic Journal of Computing. (3). pp. 27–40. [20] Kronrod, M. A. (1969). "Optimal ordering algorithm without operational field". Soviet Mathematics - Doklady (10). pp. 744.

[20] Kronrod, M. A. (1969). "Optimal ordering algorithm without operational field". Soviet Mathematics - Doklady (10). pp. 744.

[21] T. Cormen, C.Leiserson, R. Rivest and C.Stein, Introduction To Algorithms, McGraw-Hill, Third Edition, 2009,pp.145-164. [22] A. LaMarca and R. E. Ladner.(1997), "The Influence of Caches on the Performance of Sorting." Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms,. pp. 370–379.

[22] A. LaMarca and R. E. Ladner.(1997), "The Influence of Caches on the Performance of Sorting." Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms,. pp. 370–379.