In Search of the Fastest Sorting Algorithm

Emmanuel Attard Cassar

emmanuel.attard-cassar@um.edu.mt

Abstract: This paper explores in a chronological way the concepts, structures, and algorithms that programmers and computer scientists have tried out in their attempt to produce and improve the process of sorting. The measure of 'fastness' in this paper is mainly given in terms of the Big O notation.

Keywords: sorting, algorithms

Sorting is an extremely useful procedure in our information-laden society. Robert Sedgewick and Kevin Wayne have this to say: 'In the early days of computing, the common wisdom was that up to 30 per cent of all computing cycles was spent sorting. If that fraction is lower today, one likely reason is that sorting algorithms are relatively efficient, not that sorting has diminished in relative importance.'[1] Many computer scientists consider sorting to be the most fundamental problem in the study of algorithms.[2] In the literature, sorting is mentioned as far back as the seventh century BCE where the king of Assyria sorted clay tablets for the royal library according to their shape.[3]

1 R. Sedgewick, K. Wayne, *Algorithms*, 4th edn. (USA, 2011), Kindle Edition, Locations 4890-4892.

2 T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, *Introduction to Algorithms*, 3rd edn. (USA, 2009), 148.

3 N. Akhter, M. Idrees, Furqan-ur-Rehman, 'Sorting Algorithms – A Comparative Study', *International Journal of Computer Science and Information Security*, Vol. 14, No. 12, (2016), 930.

Premise

My considerations will be limited to algorithms on the standard von Neumann computer. This model has an architecture that does not permit multithreading, i.e. a program executes instructions in a strictly sequential manner. Besides, I will only consider internal sorting. For the sake of brevity, I will not go into space complexity.

Throughout the paper 'n' will always indicate the number of elements to be sorted.

Radix-sort – 1880

In 1880, prior to completely-electronic computers, the Radix-sort was being used in Herman Hollerith's tabulating machines that read and processed information from punched cards.[4] In 1954 at MIT Harold H. Seward wrote the Radix-sort as a computer algorithm for the multi-purpose programmable machine.[5] Radix-sort is an integer algorithm but it can sort on keys that can be converted to integers. It works on an array even though, as often happens in algorithms, it can be modified to work on tries.[6] It is non-comparative and has time complexity of O(nd) for the worst, average, and best cases ('d' represents the number of digits in the largest number).[7] Two types of Radix-sorts are LSD (based on the least significant digit) and MSD (based on the most significant digit). LSD is fast and stable, faster than comparison sorts (these need O(n log n) comparisons and at best their time complexity is O(n log n)). MSD is not necessarily stable but the algorithm can be modified to make it stable.[8]

4 P. Kadam, S. Kadam, 'Root to Fruit (2): An Evolutionary Approach for Sorting Algorithms', *Oriental Journal of Computer Science of Technology*, (2014), Vol. 7, No. 3, 371.

5 Wikiwand, 'Radix Sort', https://www.wikiwand.com/en/Radix_sort (accessed 26 July 2017).

6 Ibid.

7 Codingeek, 'Radix Sort – Explanation, Pseudocode and Implementation', http://www.codingeek.com/algorithms/radix-sort-explanation-pseudocode-and-implementation/ (accessed 3 August 2017).

8 Wikiwand, 'Radix Sort'.

The First Sorting Algorithms on Computers – 1945 till 1960

From the early days of electronic computers (ENIAC was completed by February 1946)[9] until 1960, a number of other sorting algorithms were invented.[10] The main ones are briefly described below.

Merge-sort

In 1945 John von Neumann, the great mathematician with special interest in computers, wrote the Merge-sort.[11] It is a divide-and-conquer algorithm that divides the sequence to be sorted recursively in two halves. The recursive halving continues until the subdivisions contain only one value or is empty. At this stage, the subdivided parts are recursively merged together until there are no more subdivided parts. Merge-sort is a comparison algorithm and works on arrays. It is stable and in its original form is not in-place. Tweaks on the code can make it in-place.[12] Its time complexity is O(n log n) in all three cases (worst, average, and best).[13] There are also parallel implementations of Merge-sort.[14]

Insertion-sort

Insertion-sort, which was invented by John Mauchly in 1946,[15] takes the values one by one and inserts them in a sorted sequence. During processing, the values are held in an array. The algorithm is in-place. Time complexity is $O(n^2)$ for the worst and average cases and O(n) in the best case.[16] It is efficient only on small sets of data.

Even though the Selection, Bubble, and Insertion sorts all have the same time complexity, in practice the efficiency of the Insertion algorithm is superior to the other two.[17]

9 *Encyclopaedia Britannica*, *ENIAC,* https://www.britannica.com/technology/ENIAC (accessed 4 August 2017).

10 Kadam, Kadam, 371; H.M. Mahmoud, *Sorting: A Distribution Theory* (USA, 2000), 250.

11 Kadam, Kadam, 371.

12 Codingeek, 'Merge Sort Algorithm – Explanation, Implementation and Complexity', http://www.codingeek.com/algorithms/merge-sort-algorithm-explanation-implementation-and-complexity/ (accessed 5 August 2017).

13 Wikiwand, 'Sorting Algorithm', https://www.wikiwand.com/en/Sorting_algorithm (accessed 5 July 2017).

14 Wikiwand, 'Merge Sort', https://www.wikiwand.com/en/Merge_sort (accessed 3 July 2017).

15 Kadam, Kadam, 371.

16 Wikiwand, 'Insertion Sort', https://www.wikiwand.com/en/Insertion_sort (accessed 5 August 2017).

17 Ibid.

Insertion-sort is stable, in-place, online, and adaptive. If in the input sequence each element is misplaced by no more than k positions, then the time complexity is O(nk).[18]

Over the years there were several modifications employed on the Insertion-sort to improve the O(n^2) time complexity. One of these was by D.L. Shell whose algorithm compares elements separated by a distance. On subsequent passes this distance is reduced until finally becoming 1 as in the original algorithm. The algorithm is known as Shell sort. Two Shell-sort variants require O($n^{3/2}$) and O($n^{4/3}$) running time. Another modification to the insertion-sort is the Binary-Insertion-sort (which employs the binary search), devised also by Mauchly in the same year, which quickens the way a value from the unsorted part is inserted in the sorted part. The algorithm, however, still runs on O(n^2).[19] Binary-Merge-sort (designed in 1972 by F.K. Hawang & S. Ling[20]) makes use of the Binary-Insertion-sort to speed up the Merge-sort on large data sets.[21] Another important sorting algorithm derived from Merge-sort and Insertion-sort and designed to perform well on many kinds of real-world data is Timsort, implemented by Tim Peters in 2002.[22] In 2006 Bender, Martin Farach-Colton, and Mosteiro published a new variant of Insertion-sort called Library-sort (or Gapped Insertion-sort). This algorithm leaves gaps between the sorted values so that an insertion would save time on shifting values.[23]

Counting-sort

Counting-sort (like Radix-sort), invented by Harold H. Seward in 1954, is an integer-sorting algorithm where a counter for each number is kept to see how many times it occurs in the input sequence.[24] This information is kept in an array. From this array, the sorted sequence is easily formed. It runs in O(n + k) time, where ‘k’ is the number of possible values in the range.[25]

18 Ibid.
19 Ibid.
20 Kadam, Kadam, 371.
21 Wikiwand, ‘Insertion Sort’.
22 Wikiwand, ‘Timsort’, https://www.wikiwand.com/en/Timsort (accessed 6 August 2017).
23 Wikiwand, ‘Insertion Sort’.
24 Wikiwand, ‘Counting Sort’, https://www.wikiwand.com/en/Counting_sort (accessed 6 August 2017).
25 Growing with the Web, ‘Counting Sort’, http://www.growingwiththeweb.com/2014/05/

The counting-sort is best adopted in situations where the ratio of the number of different keys as compared with the number of elements is small. It often appears as a subroutine in Radix-sort. It is not a comparison-sort. It is stable but not in-place.[26]

Parallel sorting can easily be applied in Counting-sort. The array can be subdivided into a number of sub-arrays and processing on each sub-array can be performed simultaneously. Then the results from each sub-array need to be merged together.

Bubble-sort

Bubble-sort is a slow in-place comparison sort. The time complexities are O(n) (best case) and O(n^2) (average and worst cases). It is very popular in schools but very inefficient in practice and therefore not practical when n is large. It is slower than the insertion-sort even when the input sequence is almost sorted.[27]

In 1956, Friend E. in his paper 'Sorting on electronic computer systems' analysed the Bubble-sort referring to it as Sorting by Exchange. K. Iverson used the name 'Bubble-sort' in 1962. In 1963 it is found named as 'Shuttle-sort'.[28]

The sort compares consecutive elements and swaps if necessary. The sort may need more passes before all the sequence is sorted. In the case of sorting in ascending order, the large values move quicker to the right than small values move to the left. For this reason, a bi-directional Bubble-sort, called Cocktail-sort, was designed with the pointer reversing its direction after each pass. Another variation, called Comb-sort, compares values separated by a distance. In this way, the movement of values is quicker.[29] This distance, it was found out empirically, has to be reduced by a factor of 1.3 for the best results.[30]

Many computer scientists (e.g. Espelid, Knuth, Yao, etc.)[31] have produced formulae of time complexities for given gap sequences.

counting-sort.html (accessed 6 August 2017).

26 Wikiwand, 'Counting Sort'.

27 Wikiwand, 'Bubble Sort', https://www.wikiwand.com/en/Bubble_sort (accessed 11 July 2017).

28 O. Astrachan, 'Bubble Sort: An Archaeological Algorithmic Analysis', *ACM SIGCSE Bulletin*, Vol. 35, No. 1 (2003), 94.

29 Ibid.

30 GeeksforGeeks, 'Comb Sort', http://www.geeksforgeeks.org/comb-sort/ (accessed 6 August 2017).

31 Ibid.

Bucket-sort

Bucket-sort is a comparison sort that first divides the elements into different buckets and then sorts each individual bucket. Finally, the sorted buckets are merged. One can implement such an algorithm by means of an array of linked lists where each linked list would hold the elements of a bucket. This sort has existed since the late fifties.[32]

Counting-sort is a special case of Bucket-sort where the bucket holds just one value.

The Bucket-sort makes use of a Hash function to calculate in which bucket a value should be posted. It helps if the elements are uniformly distributed because in this case the buckets will contain more or less the same number of elements. If the buckets are unbalanced the bucket process would not be effective.

The average time complexity is $O(n + k)$ and $O(n^2k)$ for the worst case ('k' stands for the number of buckets).[33]

Quicksort 1961

In 1961 Hoare created Quicksort which is a very clever and elegant divide-and-conquer algorithm. The algorithm chooses one value from the input sequence, referred to as the 'pivot', and, by a series of comparisons and swaps, it puts all the values less than the pivot to its left and all the values bigger than the pivot to its right. If there are values equal to the pivot, these are grouped in a middle sequence which includes the pivot. By means of recursion, each left and right section is subdivided into smaller and smaller subdivisions until a subdivision is either empty or it contains only one element in which case the subdivision is obviously sorted. The algorithm, developed by C.A.R. Hoare in 1959 and published in 1961, is a non-stable comparison sort.[34]

Unlike Merge-sort, another known divide-and-conquer algorithm, Quicksort is in-place and this saves the need of auxiliary memory. Quicksort has a best and average-case complexity of $O(n \log n)$ and a worst-case complexity of $O(n^2)$. Efficiency, however, is dependent on

32 Mahmoud, 250.
33 Wikiwand, 'Sorting Algorithm'.
34 Wikiwand, 'Quicksort', https://www.wikiwand.com/en/Quicksort (accessed 4 August 2017).

the choice of the pivot. There are various strategies for the choice of the pivot. Two known schemes are the linear-median-finding and the Las Vegas algorithm.[35] Back in 1975 Robert Sedgewick's Ph.D. thesis treated many problems related to pivot selection strategies.[36] To this day Quicksort (and Merge-sort) are still the subject of modifications and tweaks in an attempt to make it as adaptable to all forms of inputs as possible.[37]

Trees and other data structures – 1962 until 1994

During this period, more sorting algorithms were invented. Some of them made use of trees. One of these is the Heap-sort devised by J.W.J. William in 1964.[38] It has a worst-, best-, and average-case performance of O(n log n). It is a comparison and in-place algorithm but it is not stable. This sorting algorithm makes use of the heap data-structure and is divided into two steps (i) place input data in a heap, (ii) remove the largest (or smallest) element and place at the end (beginning). Continue in this way until the tree is empty.[39]

A variant of Heap-sort, designed by Edsger Dijkstra and called Smooth-sort, was published in 1981. Dijkstra did not use the heap but he devised a non-binary heap based on the Leonardo numbers. He improved the best-case performance to O(n).[40]

In the early 1960s C.L. Mallows invented Patience-sort. This algorithm first finds increasing sub-sequences and it then merges them. Its best-case performance is O(n).[41]

Topological-sort, invented by Arthur B. Kahn in 1962, places data to be sorted in a directed acyclic graph and then visits the nodes to

35 S. Devadas, Lecture 6, 'Randomization: Matrix Multiply, Quicksort' of *Design and Analysis of Algorithms*, MIT open courseware, 2015, https://www.youtube.com/watch?v=cNB2lADK3_s (accessed 4 August 2017).

36 Wikiwand, 'Quicksort'.

37 H. Zhang, B. Meng, Y. Liang, *Sort Race* (2016) https://arxiv.org/ftp/arxiv/papers/1609/1609.04471.pdf (accessed 4 August 2017), 1.

38 Wikiwand, 'Heapsort', https://www.wikiwand.com/en/Heapsort (accessed 12 August 2017).

39 Ibid.

40 Wikiwand, 'Smoothsort', https://www.wikiwand.com/en/Smoothsort (accessed 12 August 2017).

41 Wikiwand, 'Patience Sorting', https://www.wikiwand.com/en/Patience_sorting (accessed 12 August 2017).

produce the sorted output. The algorithm is most fruitful in the case where not all values have a sorting relation between each other.[42]

Regarding Quicksort both Hoare and Sedgewick implemented the idea of leaving the small sub-arrays to be sorted by means of Insertion-sort. All subarrays that contain less than M elements are sorted by means of Insertion-sort. Sedgewick gives the best value of M as that between 6 and 15.[43]

J. Bentley (1984), basing his work on N. Lomuto's ideas, proposed a scheme with two pointers that start from the left and move to the right. Another software change, due to Sedgewick (1998), was to divide the sequence in three and not two divisions. The divisions are (i) smaller than the pivot, (ii) equal to the pivot, and (iii) bigger than the pivot.[44]

R.L. Wainwright developed Bsort (1985) and Qsorte (1987). Both are based on Quicksort. Bsort is designed to work best for nearly sorted lists as well as nearly sorted in reverse. In such cases the algorithm has time-complexity of $O(n)$.[45] It also improves the average-case behaviour. Qsorte (Quicksort with early exit for sorted sub-sections) uses the middle key as pivot. Sub-sections that are sorted are marked so that they will not continue to be subdivided recursively.[46]

In 1978 Sedgewick created a modified version of Quicksort, called SedgewickFast to minimize the number of swaps. For random data and also data sorted in reverse the number of comparisons is $O(n \log n)$.[47]

In 1991, B. McDaniel wrote about variations of Quicksort one of them being Quicksort-Rotate. This version rotates values under given conditions. The worst case, however, is still $O(n^2)$.[48]

qsort7 is a fast variant of Quicksort developed in 1993 by J.L. Bentley and M.D. McIlroy. This version chooses the pivot according to the size of the array. It differentiates between array sizes of less than

42 Wikiwand, 'Topological Sorting', https://www.wikiwand.com/en/Topological_sorting (accessed 12 August 2017).

43 L. Khreisat, 'Quicksort: A Historical Perspective and Empirical Study', *IJCSNS International Journal of Computer Science and Network Security*, Vol. 7, No. 12, December 2007, 54.

44 Ibid.

45 R.L. Wainwright, 'A Class of Sorting Algorithms based on Quicksort', *Communications of the ACM*, Vol. 28, No. 4 (1985), 397.

46 R.L. Wainwright, 'Quicksort Algorithms with an Early Exit for Sorted Subfiles', *Proceedings of the 1987 Fifteenth Annual ACM Computer Science Conference* (1987), 185.

47 L. Khreisat, 'Quicksort: A Historical Perspective and Empirical Study', 57.

48 Ibid., 58.

7, between 7 and 40, and greater than 40. Still, the worst-case employs O(n^2) time-complexity.[49]

L.R. Ford and S.M. Johnson in 1959 published the Merge-Insertion sort (name given by Donald Knuth since the algorithm contains both merge and insertion techniques). The algorithm seeks to minimize the number of comparisons. It is most suitable for small values of n (5 or 6).[50]

In 1964 K.E. Batcher invented the Merge-Exchange (comparison) sort that merges pairs of sorted sub-sequences. It is particularly useful when a number of comparisons can be performed simultaneously.[51]

L.J. Woodrum in 1969 and A.D. Woodall in 1970 wrote about the Merge-sort worked out with linked lists instead of the sequential array. This solution is faster by about 10 to 20 percent and uses less memory space. It is also stable.[52]

Another Merge-sort is Natural-Merge-sort where the input sequence is scanned and sorted consecutive elements are grouped and then merged.[53]

Introsort – 1997

Introsort was devised by David Musser. It has an average and worst-case performance of O(n log n). It is a hybrid algorithm with a fast average performance in practice. It applies Quicksort but switches to Heap-sort when a subsequence is short. Introsort was written for the C++ Standard Library. Musser applied the median-of-3 for the choice of the pivot and reported improved times when compared with the original Quicksort. The Microsoft .NET Framework Class Library as from version 4.5 (2012) makes use of Introsort.[54]

49 A. Shorman, A. Shorman, *An Experimental Study of Quicksort Algorithm and Its Variants Reveals New Variant,* https://www.academia.edu/31489885/Project_Title_An_Experimental_Study_of_Quicksort_Algorithm_and_Its_Variants_Reveals_New_Variant_TABLE_OF_CONTENTS, (accessed 12 August 2017), 7, 10, 11.

50 D.E. Knuth, *The Art of Computer Programming*, 2nd edn., Vol. 3: Sorting and Searching (USA, 1998), 184, 186, 381.

51 Ibid., 111, 381.

52 Ibid., 166, 381.

53 Ibid., 160.

54 Wikiwand, 'Introsort', https://www.wikiwand.com/en/Introsort downloaded (accessed 11

In a paper by Zhang et al[55] running times of sorting algorithms are recorded. When compared with the sorts Insertion, Heap, Shell, Merge, Quicksort, C library qsort and Timsort, Introsort resulted the fastest on a randomized sequence of 200,000 values. Overall (taking elements reversely sorted, nearly sorted, etc) it had fourth average time. Quicksort was the fastest.

In the same year FlashSort was introduced. Its creator is Karl-Dietrich Neubert. If the input data is uniformly distributed its time complexity in the best case is O(n), O(n+r) in the average case ('r' is the range of numbers to be sorted), and O(n^2) in the worst case.[56]

It is based on the classification of elements and not on comparisons. Although classification algorithms are known to require a lot of memory space, FlashSort requires less than 0.1n.[57]

Sorting in O(n log log n) – 1998

The 1998 paper 'Sorting in Linear Time?'[58] presents an algorithm that sorts n integers in time proportional to n log log n. Stefan Nilsson (one of the authors of the paper) in an article written in 2000[59] claims that this algorithm is the fastest if one goes by the number of operations executed to sort integer data on a von Neumann computer.

Some algorithms, like Radix-sort, work in linear time on condition that the word length (w) of the computer is greater or equal to log n. The Andersson *et al.* algorithm does not depend on word length. It works in two phases. In the first phase, Radix sorting techniques are used to reduce the size of the numbers. In the second phase, packed merging is performed on the shortened numbers.[60] This result improves the o($n\sqrt{\log n}$) expected time for integer sorting obtained Fredman and Willard[61] in 1990.

August 2017)

55 H. Zhang, B. Meng, Y. Liang, 2.

56 Wikiwand, 'Sorting Algorithm'.

57 Khreisat, 59.

58 A. Andersson, T. Hagerup, S. Nilsson, R. Raman, 'Sorting in Linear Time?', *Proceedings of the 27th Annual ACM Symposium on the Theory of Computing* (1995), 427.

59 S. Nilsson, 'The Fastest Sorting Algorithm?', *Dr. Dobb's* (1 April 2000), 38.

60 Ibid.

61 M.L. Fredman, D.E. Willard, 'Surpassing the Information Theoretic Bound with Fusion

Timsort and the Han-Thorup Algorithm – 2002

Timsort is a widely used hybrid (built on the Merge and Insertion sorts) sorting algorithm with an average and worst-case performance of O(n log n) and a best-case performance of O(n). It is stable and adaptive. It was implemented by Tim Peters in 2002 for use in the Python programming language. Peters designed it to perform well on many kinds of real-world data.[62] He used techniques suggested by Peter McIlroy in a 1993 paper.[63]

Timsort divides the input data in a series of sub-sequences (runs) of data ordered in non-descending or descending order. The descending runs are reversed and then all runs are merged. If a sub-sequence is shorter than the value of variable minrun (a number usually from 32 to 64 inclusive) a number of elements that follow it are added to it using Insertion-sort. This is one of a number of measures taken by this algorithm to quicken the process of sorting. Another measure is making the number of runs to be merged equal or close to a power of two.[64]

Timsort keeps track of runs by pushing a reference onto a stack. It executes an almost in-place sort. The merge process is optimized to save on the number of comparisons. In this process, binary search is used. When merging starts, the algorithm may make use of galloping if, for a particular consecutive number of times, only values from one sub-sequence are chosen.[65]

Since Python version 2.3, Timsort has been its standard sorting algorithm. It is also found in Java SE 7 to sort arrays of non-primitive types.

In 2002 Y. Han and M. Thorup published a paper[66] in which they described a complex randomized integer sorting algorithm with O($n\sqrt{\log \log n}$) expected time and linear space. This lowers the upper

Trees', *Journal of Computer System Sciences*, Vol. 47 No. 3 (1993), 425.

62 Wikiwand, 'Timsort'.

63 P. McIlroy 'Optimistic Sorting and Information Theoretic Complexity', *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms* (1993) 467–674.

64 Wikiwand, 'Timsort'.

65 Ibid.

66 Y. Han, M. Thorup, 'Integer Sorting in O($n\sqrt{\log \log n}$) Expected Time and Linear Space', *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science* (2002), 135–44.

bound reached by Andersson et al[67] in 1995. Also, if the integers are bounded by U, the sorting can be done in O($n\sqrt{\log\log U}$) expected time. They assume that each integer fits in a machine word (so that each operation on integers would require only one instruction).

Library-sort – 2006

Library-sort is a tweaked Insertion-sort proposed by Michael A. Bender, Martín Farach-Colton, and Miguel Mosteiro in a 2006 paper.[68] Its time complexities are: O(n^2) in the worst case, O(n) in the best case and O(n log n) in the average case (with high probability).[69]

The main idea of the library-sort is to insert gaps between the elements so that an insertion in the sorted part does not require the time-consuming shift of elements. Library-sort is a stable and online comparison sort. Because of these gaps, memory consumption is high. Library-sort uses rebalancing so that the gaps are equally spaced. To save time in finding where an element should be inserted, the binary search is used.[70]

Recent algorithms 2004–2017

In 2004 R. Sinha and J. Zobel[71] introduced Burst-sort on strings. It makes use of a compact trie and the concept of buckets. This simple, fast, and efficient algorithm was tested empirically by Sinha and Zobel. It is fast, owing to the fact that it attains a low rate of cache misses. Its average and worst time complexity is O(wn) where ‘w’ is the word length.[72]

67 Andersson, Hagerup, Nilsson, Raman.

68 M.A. Bender, M. Farach-Colton, M.A. Mosteiro, ‘Insertion Sort is O(n log n)’, *Theory of Computing Systems*, Vol. 39, No. 3 (2006) 391–397.

69 Wikiwand, ‘Library Sort’, https://www.wikiwand.com/en/Library_sort (accessed 21 August 2017).

70 Ibid.

71 R. Sinha, J. Zobel, ‘Cache-Conscious Sorting of Large Sets of Strings with Dynamic Tries’, *Journal of Experimental Algorithmics*, Vol. 9, Article 1.5 (2004).

72 Wikiwand, ‘Burstsort’, https://www.wikiwand.com/en/Burstsort (accessed 25 August 2017).

In 2006 Ayala-Rincón *et. al*[73] wrote a variant of the Merge-Insertion-sort calling it $_4$FJ. They succeeded in reducing the administrative space to 33% of the original Merge-Insertion-sort without increasing the number of comparisons. This is attained by recursively working on over a quarter of the input instead of half.[74]

Also in 2006, Sundararajan and Chakraborty[75] proposed SS06 (called 'new sorting algorithm' in their paper) which is based on Quicksort. It requires an auxiliary array as big as the array holding the input sequence. There are no swaps. Every element is compared to the pivot. If the element is smaller than the pivot it is put in the first empty position in the auxiliary array. If not, it is placed in the last empty position. It takes O(n log n) in the average case and O(n^2) in the best and worst cases.[76]

In 2009 V. Yaroslavskiy[77] experimented with Quicksort with two pivots, i.e. partitioning in three parts. He showed that this method is faster when compared with known implementations.

In 2013, Lam and Wong[78] introduced Rotated-Library-sort. The original Insertion-sort runs in O(n^2), Library-sort runs in O(n log n) while this new algorithm runs in O($\sqrt{n}\log n$). It also requires less space than Library-sort.

In 'Sort Race', a 2016 paper,[79] the authors experimented with their own versions of Quicksort and Merge-sort. They timed different sorting programs to compare their duration. When the sorts Insertion, Heap, Shell, Merge, Quicksort, qsort (the sorting function in the GNU C Standard Library), Timsort and Introsort were executed on random data, Introsort was the fastest with Quicksort a close second. Overall (i.e. considering also times for reversely sorted input, nearly sorted and few unique) Quicksort was the fastest.

73 M. Ayala-Rincón, B.T. de Abreu, J. de Siqueira, 'A variant of the Ford-Johnson algorithm that is more space efficient', *Information Processing Letters*, Vol. 102, No. 5 (2007), 201–7.

74 Ibid.

75 K.K. Sundararajan, S. Chakraborty, 'A new sorting algorithm', *Applied Mathematics and Computation*, Vol. 188, No. 1 (2007),1037–38.

76 Shorman, Shorman, 7, 10.

77 V. Yaroslavskiy, *Dual-Pivot Quicksort Algorihm*, http://codeblab.com/wp-content/uploads/2009/09/DualPivotQuicksort.pdf, (accessed on 20 August 2017).

78 F. Lam, R.K. Wong, 'Rotated Library Sort', *Proceedings of the Nineteenth Computing: The Australasian Theory Symposium* (2013), 21–6.

79 H. Zhang, B. Meng, Y. Liang.

Then five different Merge-sorts were created (mer2, mer3 … mer6) and compared with qsort, Timsort, and Neat-sort (a practical adaptive algorithm that reads the input values one by one and creates sorted lists that are then merged). On random values Timsort was the quickest with one version of Merge-sort (mer6) a close second. Then different types of Quicksort were tested against Bentley and McIlroy's qsort function, 3-way Quicksort, 2-way Quicksort, Introsort, mer6, Timsort, and qsort. One hybrid version of Quicksort was the best performer.

In 2012 Dalhoum *et al.*[80] presented a Quicksort algorithm, called MQuickSort, based on a dynamic pivot selection. The algorithm is data-dependent and this fact increases the chances of splitting the array or list into relatively equal sizes. The proposed algorithm converts the worst-case into a best-case behaviour with Θ (n) execution time. The algorithm is capable of recognizing a sorted array or sub-array that does not require further processing. The idea is to find the pivot by calculating the average of the values.

Conclusion

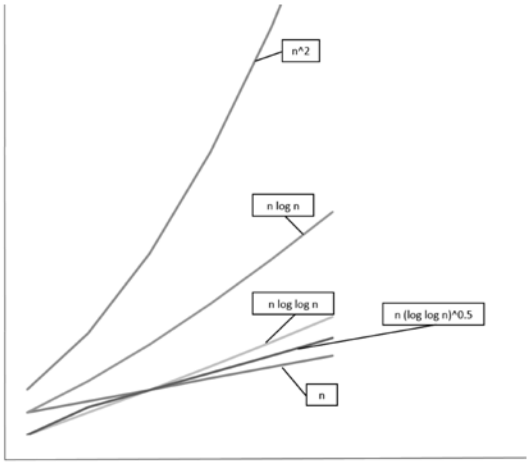

Fig. 1 The graphs of the functions n^2, n log n, n log log n, $n\sqrt{\log\log n}$, and n

80 A. L. Abu Dalhoum, T. Kobbay, A. Sleit, M. Alfonseca, & A. Ortega, 'Enhancing QuickSort Algorithm using a Dynamic Pivot Selection Technique', *Wulfenia Journal,* Vol. 19, No. 10 (2012), 543–52.

Besides the sorting algorithms seen here there were other attempts and tweaks to find faster sorting algorithms. Compression,[81] specialized hardware structures,[82] external sorting,[83] amongst others, were all studied. This study treated only internal sorting. The quality of sorting was evaluated mainly by means of the Big O notation. It is very evident from the graph above that the algorithm that mostly drew near the O(n) lower limit, which cannot be lowered for obvious reasons (this is because each entry has to be read), is the Han-Thorup algorithm.

Emmanuel Attard Cassar, BA (Ed.) (Hons.) (Melit.), also graduated Dottore in Scienze dell'Informazione at the University of Pisa. He lectures in computer science at the Junior College University of Malta. His main interests are algorithms and the history of mathematics and computing. He also practices literary writing. He published *Vacanze romane 2012* (2013) and *Fuq Din l-Art Ħelwa* (2016). He is also the founder of EspRimi, a multi-disciplinary group interested in the expression of various forms of art and sciences.

81 A. Lo´ Pez-Ortiz, M. Mirzazadeh, M. Ali Safari, H. Sheikhattar, 'Fast String Sorting Using Order-Preserving Compression', *ACM Journal of Experimental Algorithmics*, Vol. 10, Article 1.4 (2005), 1–12.

82 S. Abdel-Hafeez, A. Gordon-Ross, 'An Efficient O(N) Comparison-Free Sorting Algorithm', *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 25, No. 6 (2017), 1930–42.

83 K. Yaron, Y. Hadas, 'External Sorting on Flash Storage: Reducing Cell Wearing and Increasing Efficiency by Avoiding Intermediate Writes', *The VLDB Journal*, Vol. 25, No. 4 (2016), 495–518.