IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Ingénierie des Systèmes d'Information
Vol. 25, No. 1, February, 2020, pp. 113-117
Journal homepage: http://iieta.org/journals/isi

Performance Comparison of Sorting Algorithms with Random Numbers as Inputs

Suvarna Buradagunta*, Jyostna Devi Bodapati, Nirupama Bhat Mundukur, Shaik Salma

Department of CSE, VFSTR Deemed to be University, Vadlamudi 522213, Guntur District, Andhra Pradesh, India

Corresponding Author Email: bs_cse@vignan.ac.in

https://doi.org/10.18280/isi. 250115

Received: 7 September 2019
Accepted: 5 December 2019

Keywords:
random inputs, UNH sort, bubble sort, insertion sort, selection sort, merge sort, quick sort

ABSTRACT

Sorting is a huge demand research area in computer science. Sorting is a process of arranging the elements in an order. In practical application computing requires things to be in order. A comparative study is done in this study and performed the comparison for both positive and negative numbers by taking the random numbers as input. In this study, different algorithms like UNH Sort, Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Quick Sort are considered for the experimentation. From the obtained results we can conclude that, Initially when the input size is less UNH Sort is giving best when it is compared with bubble sort. When the input size increases bubble sort takes long time to perform sorting. And Quick sort takes less time to perform sorting.

1. INTRODUCTION

Sorting is a huge demand research area in computer science. Sorting algorithms are widely used in different applications like finding duplicates from a set of elements, Ranking i.e. for a given set of elements finding the minimum and maximum, median, numerical applications, Matrix chain Multiplication [1], Kruskal algorithm and in searching. Searching can be done easily after performing the sorting. We can also get the k^{th} position element easily when the elements are sorted.

Sorting is a process of arranging the elements in an order (increasing or decreasing) based on the requirement. Generally there are two types of sorting namely Internal sorting and External Sorting. In the Internal Sorting the entire data is stored in memory during sorting. Whereas External Sorting means data is stored outside i.e. hard disk and loaded into memory when needed. And it is especially used when whole data can't fit into memory.

In internal sorting we have different sorting algorithms those are: Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Quick Sort, Heap Sort and Radix Sort. From the listed methods merge sort and Quick sort the two sorting techniques performed by using the Divide and Conquer techniques. Procedures for all the sorting listed above are discussed in detail in the following sections.

The rest of the paper is organized as follows. Section 2 discuss about the related work. Section 3 discuss about the comparative study. Section 4 discuss about the experimentation and results. The conclusion and future work presented in section 5.

2. RELATED WORK

A Study of sorting algorithms began in the early 1950's and it is still going on. Yang et al. [2] done the experimental study on five sorting algorithms, Faujdar et al. [3] done the analysis on bucket sort, Suresh et al. [4] done the analysis on various combination sorting algorithms. As there are advancements in hardware, several parallel techniques are developed. Efficiency of the algorithms can be measured by both time and space complexities [5-10].

3. COMPARIVE STUDY

3.1 Selection sort

Selection sort is a sorting algorithm which takes the time complexity i.e. O (n^2). Procedure: Initially take an array of unsorted elements. Consider initial position (first element) as min and compare that element with all the remaining elements in the array (which are right to first element) and if found the smallest element then replace that smallest element with min. Then move the min to next position i.e.

$$\min\_ position = \min\_ positiion + 1$$

Similarly do the same procedure for remaining elements which are in the right side (since all left side elements are sorted). If there is no element smaller than min in the right side the move the min to next position as above.

Follow the same procedure for all remaining elements in the array until the elements are sorted [1]. Applications of Selection Sort are, it is used for small data sets and can be used in Embedded System [11].

3.2 Bubble sort

It is also known as Adjacent Sort or Exchange Sort. It can be done in passes. In this for every pass an element is bubbled up. If there are n elements, the sorting should be completed in (n-1) passes. Procedure: Initially take an array of 'n' elements. **Pass 1:** Compare first 2 numbers, if second number is smaller than first number then swap the two numbers. Repeat the same step for remaining elements in the array so that second number is compared with third element and the next compare third

element with fourth element and so on repeat the step. So that finally the maximum number goes into its original position (i.e., last). **Pass 2:** In this the procedure is same as pass 1 but here the last element is not participated in swapping and the next large element is in its correct position. Similarly follow same procedure for (n-1) passes and get all elements are sorted. Applications of bubble sort: The real time application for bubble sort is classroom where student marks are sorted in this way and assigning the rank.

3.3 Insertion sort

Insertion sort is another algorithm in which it is similar to playing cards. Procedure: The array of elements is divides into two parts. Sorted array and unsorted array. Generally, the left side elements are sorted, and right side elements are unsorted. Compare first element of unsorted array with last element of sorted array and then place that unsorted element in its original position in the sorted list. Similarly follow the same procedure for remaining elements. This can be used in Playing cards.

3.4 UNH sort

This is the new algorithm that we are going to discuss. The algorithm of the UNH sorting is explained below.

Algorithm UNHSort(A)

Begin:

for i = n-1 down to 1 do

for j = 1 to i do

if A[j] > A[j+1] then

swap(A[j], A[j+1])

end:

In every iteration of the inner loop the maximum element is reaching to the last. Like this the UNH sorting algorithm is sorting the elements. This algorithm takes time complexity O (n^2).

3.5 Merge sort

Merge sort depends on procedure called Merging. Merge sort algorithm uses divide, conquer and combine algorithm. Divide indicates that we need to divide the array of 'n' elements into n/2 parts. If array with 0 or 1 element, then need not sort it again. Divide the elements of each part again until we get separate elements. After getting the separated elements, sort the elements by using conquer technique of two sub parts. Then combine two sub parts then we get the sorted array of 'n' elements.

3.6 Quick sort

It is also called as partition exchange sort. It also follows the divide and conquers method like merge sort. Procedure: Initially take array of n elements. Here we heard about the term pivot element generally, the stating elements are considered as pivot element. So here our aim is to place the pivot element into its original position. Here we are also taking care about left and right sides. Pivot element is compared with all the remaining elements such that the numbers smaller than pivot are at left side and the numbers [12] greater than pivot element are placed at right side that's why we also considering left and right sides i.e., smaller----Pivot-----Greater---

Here the procedure how pivot element is in its position: Pivot element first start comparing from right side and if element less than pivot then swap the two numbers then pivot element position changes and then the pivot is at right side now. Compare the pivot element from left side now and if any number greater than pivot then swap the two numbers. Follow the same procedure until the pivot element placed in its original position. Finally, here we apply divide and conquer method for elements before the pivot and after the pivot and the sort the two lists by same procedure as merge sort and combine two sorted lists by placing pivot in its place and then elements are sorted.

3.7 Heap sort

Heap sort is done through the tree concept. Procedure: Consider an array of n unsorted elements. These elements can be represented in the form of the tree as the first element in an array is considered as root element and second element and third element of array are at left side and right side of root element respectively. That tree is considered as Normal Tree. Now we need to convert this tree in to Max heap tree which means the maximum element in the Normal Tree is considered as root node and ensure that root node is larger or in some cases it is equal to any one of the remaining nodes. In max heap swap the root node with last element and then redraw the tree by disconnecting the root node and all disconnected nodes are stored in another array. Again, redraw the max heap tree with remaining elements in same manner. Perform the same procedure by getting the minimum number in the array and finally, we get the sorted elements.

3.8 Radix sort

Radix sort is performed from least significant digit to most significant digit. Sorting can be completed in 'n' passes, here 'n' indicates that the maximum number of digits present in the largest number in the array elements.

Procedure: Consider queues (Q_0-Q_9) which are called as buckets. Consider the last digit of each element and place each element in respective queue number. (if digit is same then place according to sequence). The array is changed by taking the elements from Q_0 to Q_9. Repeat the same procedure until most significant digit and we get the sorted elements [13-29].

4. EXPERIMENTATION

Six sorting algorithms are selected for experimentation. For the experimentation we take the elements or numbers from 100 and tested with different size of the elements up to 2,00,000. We conducted experiments for taking both positive elements and negative elements. We generated the different size of elements with the help of random number generator. The obtained results are shown in the form of table. Table 1 for positive random numbers and Table 2 for negative numbers. And the obtained results also presented in the form of graph. Comparison is done for UNH sort, Insertion sort, Bubble sort, selection sort, and the algorithms which use the divide and Conquer techniques like Merge sort and Quick sort.

The experimentation part is organized as follows. Initially UNH Sort, Bubble sort, Selection sort and Bubble sort are compared in both positive and negative numbers in the comparison 1. In the next section, only the comparison is done for merge sort and quick sort. In the last section, the comparison is done for all the sorting algorithms.

Table 1. With positive inputs

Records\Sorting	UNH	Insertion sort	Bubble Sort	Selection Sort	Merge Sort	Quic kSort
100	0.000	0.0003	0.0003	0.0002	0.000	0.0002
1000	0.008	0.0053	0.0062	0.0050	0.002	0.0013
2000	0.024	0.0092	0.0114	0.0181	0.006	0.0017
5000	0.105	0.0681	0.0866	0.0573	0.008	0.011
10000	0.334	0.212	0.3526	0.1565	0.030	0.0225
20000	1.438	0.078	1.488	0.5206	0.040	0.0496
40000	5.784	2.7785	5.9458	1.8455	0.103	0.0746
60000	13.03	6.1519	13.573	4.1023	0.169	0.1207
80000	23.25	10.893	24.170	7.0718	0.239	0.1280
1 lack	36.37	17.090	38.007	11.023	0.302	0.1435
2 lack	147.1	67.401	150.51	45.472	0.901	0.3097

Table 2. With negative inputs

Records\Sorting	UNH	Insertion sort	Bubble Sort	Selection Sort	Merge Sort	Quick Sort
100	0.000	0.000	0.000	0.000	0.000	0.000
1000	0.012	0.005	0.006	0.006	0.002	0.000
2000	0.027	0.017	0.029	0.026	0.002	0.000
5000	0.097	0.087	0.084	0.103	0.010	0.004
10000	0.396	0.244	0.388	0.338	0.012	0.008
20000	1.515	0.801	1.494	1.299	0.043	0.017
40000	6.034	3.050	5.978	5.226	0.057	0.032
60000	13.53	6.800	13.46	11.75	0.124	0.044
80000	24.04	11.92	24.14	20.95	0.125	0.063
1 lack	37.42	18.56	37.62	32.69	0.172	0.072
2 lack	149.1	74.50	150.2	135.6	0.296	0.131

4.1 Comparison of UNH sort, Insertion sort, Bubble sort and Selection sort

Initially UNH Sort, Bubble sort, Selection sort and Bubble sort are compared with positive numbers in Figure 1, and negative numbers in Figure 2.

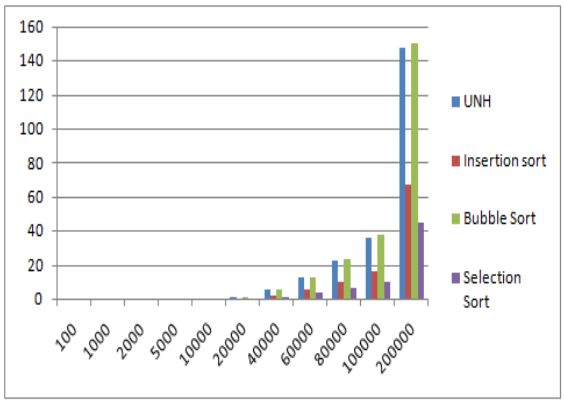

Figure 1. For positive numbers

From the obtained results we can observe that initially UNH sort taking more time compared with other sorting algorithms, but later some point of time Bubble sort is taking the large amount of time.

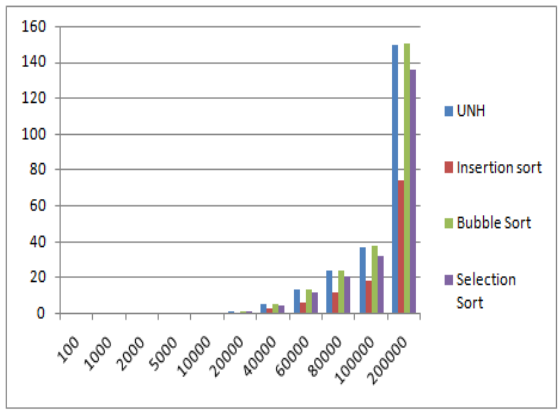

Figure 2. For negative numbers

4.2 Comparison of merge sort and quick sort

In this section the comparison is done for merge sort and quick sort. Comparison is done for the positive numbers and negative numbers. Comparison with positive inputs presented in Figure 3. Comparison with negative inputs presented in Figure 4.

From the obtained results quick sort is taking less time when it is compared with merge sort in both positive and negative numbers.

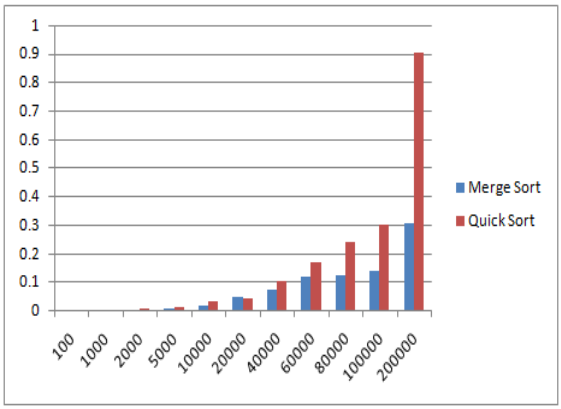

Figure 3. For positive numbers

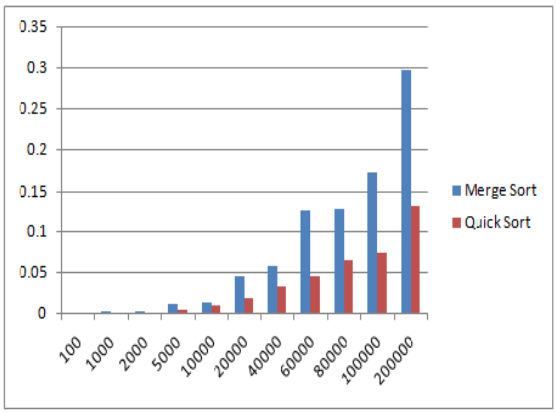

Figure 4. For negative numbers

4.3 Comparison of all the sorting algorithms

The experimentation results for different sizes of all the sorting algorithms which are described in this study are given in Table 1 and Table 2. Table 1 is for the positive elements. Table 2 is for the negative inputs.

The obtained results are placed in the form of graphs. Figure 5 describes about with the positive inputs. And Figure 6 describes about with negative numbers.

When all the algorithms are compared Quick sort is taking less execution time and Bubble sort is taking large amount of time to perform the sorting.

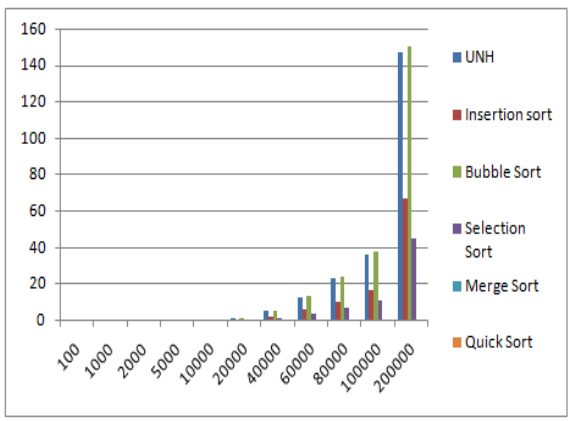

Figure 5. For positive inputs

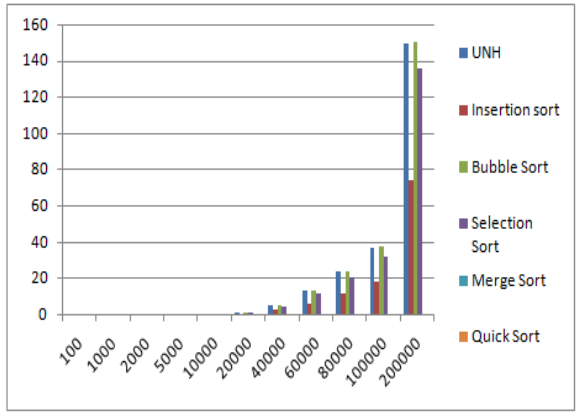

Figure 6. For negative inputs

5. CONCLUSION

To perform sorting different sorting algorithms are performing a vital role. Hence in this study, several sorting algorithms are compared by doing the experimentation with random positive and negative numbers. Experimental results are saying that quick sort is taking the less amount of time irrespective of the data size and type of the data. In future, new algorithms can be used to observe the behavior of UNH and other algorithms.

REFERENCES

[1] Suvarna, B., Maruthi Padmaja, T. (2018). Enhanced matrix chain multiplication. Journal of Cyber Security and Mobility, 7(4): 409-420. https://doi.org/10.13052/jcsm2245-1439.743

[2] Yang, Y., Yu, P., Gan, Y. (2011). Experimental study on the five sort algorithms. International Conference on Mechanic Automation and Control Engineering, Hohhot, China, pp. 1314-1317. https://doi.org/10.1109/MACE.2011.5987184

[3] Faujdar, N., Saraswat, S. (2017). The detailed experimental analysis of bucket sort. 7th International Conference on Cloud Computing, Data Science & Engineering, Noida, India, pp. 1-6. https://doi.org/10.1109/CONFLUENCE.2017.7943114

[4] Suresh, A., George, A.K. (2018). Performance analysis of various combination sorting algorithms for large dataset to fit to a multi-core architecture. 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), Coimbatore, India, pp. 51-56. https://doi.org/10.1109/ICICCT.2018.8472956

[5] Kunth, D.E. (1975). The art of computer programming: Sorting and searching, Addison-Wesley. Volume 3. Sorting and searching, Second Edition.

[6] Gugale, Y. (2018). Super sort sorting algorithm. 2018 3rd International Conference for Convergence in Technology (I2CT), Pune, India, pp. 1-5. https://doi.org/10.1109/I2CT.2018.8529769

[7] Faujdar, N., Ghrera, S.P. (2015). Analysis and testing of sorting algorithms on a standard dataset. 2015 Fifth International Conference on Communication Systems and Network Technologies, Gwalior, India, pp. 962-967.

https://doi.org/10.1109/CSNT.2015.98
[8] Abdel-Hafeez, S., Gordon-Ross, A. (2017). An efficient O(N) comparison free sorting algorithm. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 25(6): 1930-1942. https://doi.org/10.1109/TVLSI.2017.2661746
[9] Mandal, P.K., Verma, A. (2019). Novel hash-based radix sorting algorithm. IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), New York City, USA, pp. 0149-0153. https://doi.org/10.1109/UEMCON47517.2019.8992938
[10] Osama, H., Omar, Y., Badr, A. (2016). Mapping sorting algorithm. Sai Computing Conference, London, UK, pp. 488-491. https://doi.org/10.1109/SAI.2016.7556025
[11] Kumari, S., Singh, D.P. (2014). A parallel selection sorting algorithm on GPUs using binary search. 2014 International Conference on Advances in Engineering & Technology Research (ICAETR-2014), Unnao, India, pp. 1-6. https://doi.org/10.1109/ICAETR.2014.7012819
[12] Patel, Y.S., Singh, N.K., Vashishatha, L.K. (2014). Fuse sort algorithm: A Proposal of divide & Conquer based sorting approach with O(n log logn) time and linear space complexity. 2014 International Conference on Data Mining and Intelligent Computing (ICDMIC), New Delhi, India, pp. 1-6. https://doi.org/10.1109/ICDMIC.2014.6954240
[13] Yildiz, Z., Aydin, M., Yilmaz, G. (2013). Parallelization of biotonic sort and radix sort algorithms on many core GPUs. 2013 International Conference on Electronics, Computer and Computation (ICECCO), Ankara, Turkey, pp. 326-329. https://doi.org/10.1109/ICECCO.2013.6718294
[14] Durad, M.H., Akhtar, M.N., Haq, I.U. (2014). Performance analyisis of parallel sorting algorithms using MPI. 12th International Conference on Frontiers of Information Technology, Islamabad, Pakistan, pp. 202-207. https://doi.org/10.1109/FIT.2014.46
[15] Khurana, M., Faujdar, N., Saraswat, S. (2017). Hybrid bucket sort switching internal sorting based on the data inside the bucket. 2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, pp. 476-482. https://doi.org/10.1109/ICRITO.2017.8342474
[16] Zhong, C., Ke, Q., Liu, J., Huang, Y.R. (2011). Thread-level parallel algorithm for sorting integer sequence on multi-core computers. 2011 Fourth International Symposium on Parallel Architectures, Algorithms and Programming, Tianjin, China, pp. 37-42. https://doi.org/10.1109/paap.2011.57
[17] Guigang zheng, shaohua teng, wei zhang, xiufen fu(2009). a cooperative sort algorithm based on indexing. 13th international conference on computer supported cooperative work in design.pp 704-709. https://doi.org/10.1109/cscwd.2009.4968141
[18] Ullah, S., Khan, M.A., Khan, M. A., Akbar, H., Hassan, S.S. (2015). Optimized selection sort algorithm for two dimensional array. 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), Zhangjiajie, China, pp. 2549-2553. https://doi.org/10.1109/fskd.2015.7382357
[19] Ghaffarizadeh, A., Ahmadi, K., Flann, N.S. (2011). Sorting unsigned permutations by reversals using multi-objective evolutionary algorithms with variable size individuals. 2011 IEEE Congress of Evolutionary Computation (CEC), New Orleans, LA, USA, pp. 292-295. https://doi.org/10.1109/CEC.2011.5949631
[20] Khurana, M., Faujdar, N., Saraswat, S. (2017). Hybrid bucket sort switching internal sorting based on the data inside the bucket. 6th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO), Noida, India, pp. 476-482. https://doi.org/10.1109/icrito.2017.8342474
[21] Chen, P.N., Gao, M.Y, Huang, J.Y., Yang, Y.X., Zeng, Y. (2018). High-speed color sorting algorithm based on fpga implementation. 2018 IEEE 27th International Symposium on Industrial Electronics (ISIE), Cairns, QLD, Australia, pp. 235-239. https://doi.org/10.1109/isie.2018.8433831
[22] Faujdar, N., Ghrera, S.P. (2015). A detailed experimental analysis of library sort algorithm. 2015 Annual IEEE India Conference (INDICON), New Delhi, India, pp. 1-6. https://doi.org/10.1109/INDICON.2015.7443165
[23] Lucas', K.T., Jana, P.K. (2009). An efficient parallel sorting algorithm on OTIS mesh of trees. 2009 IEEE International Advance Computing Conference (IACC 2009), Patiala, India, pp. 175-180. https://doi.org/10.1109/IADCC.2009.4809002
[24] Faujdar, N., Ghrera, S.P. (2015). Analysis and testing of sorting algorithms on a standard dataset. 2015 Fifth International Conference on Communication Systems and Network Technologies, Gwalior, India, pp. 962-967. https://doi.org/10.1109/CSNT.2015.98
[25] Peters, H., Schulz-Hildebrandt, O., Luttenberger, N. (2012). A novel sorting algorithm for many-core architectures based on adaptive bitonic sort. 2012 IEEE 26th International Parallel and Distributed Processing Symposium, Shanghai, China, pp. 227-237. https://doi.org/10.1109/IPDPS.2012.30
[26] Lipu, A.R., Amin, R., Mondal, M.N.I., Mamun, M.A. (2016). Exploiting parallelism for faster implementation of bubble sort algorithm using FPGA. 2nd International Conference on Electrical, Computer & Telecommunication Engineering (ICECTE), Rajshahi, Bangladesh, pp. 1-6 https://doi.org/10.1109/ICECTE.2016.7879576
[27] White, S., Verosky, N., Newhall, T. (2012). A CUDA-MPI hybrid bitonic sorting algorithm for GPU clusters. 2012 41st International Conference on Parallel Processing Workshops, Pittsburgh, PA, USA, pp. 588-589. https://doi.org/10.1109/ICPPW.2012.82
[28] Wang, L.W., Zhu, Y.Q., Pan, Y.F. (2005). FCM algorithm and index cs for the signal sorting of radiant points. Proceedings of the Fourth International Conference on Machine Learning and Cybernetics, Guangzhou, China, pp. 4415-4419. https://doi.org/10.1109/ICMLC.2005.1527716
[29] Tarasiuk, P., Yatsymirskyy, M. (2018). Optimized concise implementation of batcher's odd-even sorting. IEEE Second International Conference on Data Stream Mining & Processing, Lviv, Ukraine, pp. 449-452. https://doi.org/10.1109/DSMP.2018.8478515