Experimental Analysis of On(log n) Class Parallel Sorting Algorithms

Mubashir Ali[1], Zarsha Nazim[2], Wajid Ali[3], Aamir Hussain[4], Nosheen Kanwal[5], Mahnoor Khalid Paracha[6]

Department of Software Engineering, Lahore Garrison University, Lahore, Pakistan[1,2;]
Department of Computer Science, Muhammad Nawaz Sharif University of Agriculture, Multan, Pakistan[4];
Department of Computer Science, Bahauddin Zakariya University, Multan, Pakistan[3,5,6;]

Abstract
Amount of data is rising rapidly with the passage of time. Sorting is well known computer science problem that is widely used in various applications. So there is need of certain sorting techniques that can arrange this data as fast as possible. The analysis of sorting algorithms on performance basis has significance in understanding that which technique is most effectual in the field of data management and which approach can arrange massive data accurately in least duration of time. The performance of an algorithm is measured on the basis of time complexity, space complexity and computation complexity. Multicore computer architecture attracts the researchers towards parallel computing for attaining highest computational performance from computer systems. In this research paper, an experimental analysis is conducted to measure the performance of On(log n) class sorting algorithms in terms of execution time in parallel manner. Only same On(log n) class 12 algorithms are analyzed that leads this work towards novel results. Experimentation is performed using C++ language and OpenMP library is implemented for standard parallelism. Data size increase in terms of 2 power N. Test cases are executed with three type of following integer data; random integers, sorted integers and reversed sorted integers. State of the art results are illustrated using comparative graphs that shows the performance of different algorithms under same scenario. This research work help to select appropriate sorting technique with regard to data set and environment.

Key words:
Sorting Algorithms, Experimental Analysis, Time Complexity, On(log n) Class, Parallel Processing, OpenMP

1. Introduction

Sorting is one of the most studied problem in the world of computer science. Due to its importance, it is implemented in many computer applications. Sorting is conjugative problem having more than one techniques with diverse solutions [1]. Saving, arranging and managing data was not a big issue since six to seven decades ago. But the data began to rise at enormous speed with the domination of internet at global level. According to international statistics, every human is creating nearly 1.5 megabytes of data every second which means by year 2020 there is going to be 44 trillion zettabytes of data in digital world [2]. Managing the data that is growing at this speed requires effectual data management techniques by which the user can store and arrange data as fast as possible with efficiency.

Sorting is the data management technique which helps to arrange data in particular order. Phonebooks having contacts, different languages dictionaries, ranking in search engines, password encryption and decryption are the most common applications of sorting [3,4]. Searching becomes faster with the help of sorting. It is one of the basic operations performed in every computer's database [5]. Various sorting algorithms were developed in the past. The performance of sorting algorithms is measured by complexity analysis. Time complexity is one of the important factor for measuring performance of an algorithm. Sorting algorithms are also classified on the basis of time complexity classes. In this research, we selected 12 well-known On(log n) class sorting algorithms for experimental analysis. Number of tools exists in computing world to practice parallel computing in real life applications. Parallel computing opens many opportunities for sorting related problems. It requires significant communication bandwidth among cores, unlike many other parallel computing standards. Parallel computing can be categorized as Multicore and Multiprocessor [6]. A core is the component of the processor, its main tasks are to perform, read and execute the instructions. Multicore processors chips are made up of more than one core. The main advantage of a multicore processor over single core is that the multicore processor has options that it can either use all available cores to deal with a single task by dividing it into threads. Parallel computing is a process in which a single instruction can be divided up into different partitions and every part is executed on different layers (cores) of a system; it is also known as multithreading. Multicore processors also have capability to deal with multiple tasks at a single time [7]. The computer architecture has been categorized as instruction level parallelism and data level parallelism. Peak efficiency stage of parallel application can be reached by using multicore computing technology. Performance can be estimated after collecting the information about the execution characteristics of a sorting or problem solving

Manuscript received January 5, 2020
Manuscript revised January 20, 2020

strategy. Execution time is one of the complexity factor which help to measure the performance of any algorithm [8]. By making a change in the technique from sequential to parallel could give us effectual results in lesser execution time. The actual reason behind designing parallel sorting techniques is to attain performance standards. The performance standards of these strategies concern with a number of cores that exists in the system, core to core discontinuation, design of memory hierarchy, and costs on synchronization.

OpenMP stands for "Open Multi-Processing" library. OpenMP is a directive based parallel processing model for C++ language. OpenMP program is essentially a sequential program added with compiler directives to set out parallelism. OpenMP makes conversion of existing sequential programs into parallel programs easier. Parallel regions are where parallel execution occurs via multiple simultaneously executing threads. Every single thread contains its own program counter and executes instructions one after another, like sequential program execution. Shared variables are the means of communicating data between threads. OpenMP architecture is derived from the standard of ANSI X3H5 and having the advantages of portability and extensibility [9]. OpenMP is used to bring parallelism in various real life applications [10]. It is conducive in multi-threaded processing based applications evolution. OpenMP programming strategy composed of an assembly of directives, pragmas, functions and variables. By implementing it in C++, it provides mind blowing outputs by dealing with the tasks of parallel processing dividing and passing load efficiently [11].

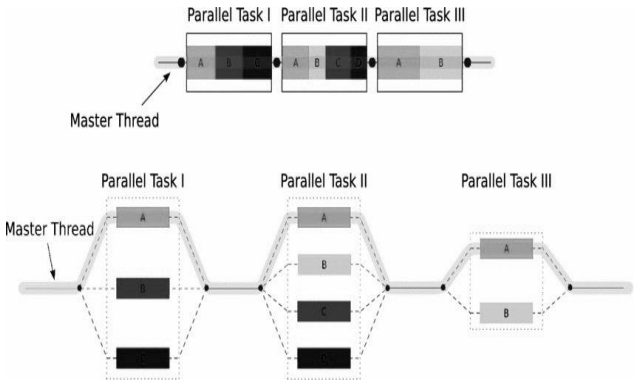

Fig. 1 Parallel processing [12]

The figure 1 is demonstrating the mechanism of parallel processing or multithreading in which master thread is controlling the parallel tasks. Comparative analysis of sorting algorithms return the most effectual techniques to efficiently sort the data [13].

This paper is arranged in the following parts. Part two discusses the background work that will help us to justify the validity of domain. Part three elaborates different sorting algorithms that are analyzed in experimental analysis. Part four elaborates the experimental setup, actual experimentation and comparative results with the help of graphs and charts. Lastly, the part five concludes the research work with outcomes and findings.

2. Background Study

In past few years, a lot of work has been done to find efficient ways to solve multiple problems simultaneously and this new field of computing is known as "Parallel computing". Parallel computing is most widely used paradigm to improve the performance of the systems. Modern day Computers are complex and they have a capability of executing different application programs on multiple processors. In simple words, parallel computing comes with the main purpose of synchronous utilization of multiple computer threads or cores to deal with a multiple computational problem. A problem is divided into different parts and instructions from each part are executed simultaneously on different processing units. In parallel computing multiple processing units or threads can perform operations independently, but they have the same memory resources.

Singh et al. distinguished the sequential scheduling and parallel scheduling of the number of quest in two stage decision problem. By utilizing the available information related to task interdependencies, the total amount of time can be calculated and amount of effort that is required for any proposed appointment of tasks to the two stages. This paper proposed a perspective for lowering either required time or required effort or both by regulating the schedule that which of the tasks should be dealt with at which time. This proposed method can be applied to information from a computer workstation design problem [14]. Unfortunately, practical Parallel Sort libraries are very hard to design, they must be carefully tuned and rely on a number of trade-off depending on the target architecture and target application. A detailed analysis of shortcomings and benefits of each selected sorting technique and an insight onto that which of the algorithm is most suitable for a specific type of application [15]. Merging algorithms for parallel computer are used to derive parallel sorting algorithms where processors interact with each other through interconnection networks like the mesh, the perfect shuffle and a lot of other sparse networks. In this paper after discussion phase about the network sorting strategies, they have shown a model which is based on shared memory allocation for parallel computation, derivation of faster algorithms from parallel enumeration sorting strategies becomes possible. In which

first goal needed to be accomplished is the ranking of all keys and then in the next step they described to rearrange the keys according to the ranking in step one. Evaluation of Parallel sorting strategies is done on the basis of a number of criteria, which is not only relevant to their time complexity but it also depends on their feasibility. They further described that the thing which is making their attractive communication schemes more suitable for implementation is that the network sorting algorithms are working on non-adaptive schedules [16]. Sorting algorithms are also implemented in various power systems. An upgradation of binary bat algorithm for effective allocation of MPUs in power systems is presented by [17]. There are a lot of applications that are time-critical like weather forecasting. In which the accurate weather forecast of upcoming days is calculated and sorted in specific order. This procedure of calculation is done again and again to fetch the updated results. The computational complexity of a weather problem is directly related to the accuracy and detail of the requested forecast simulation. Weather code are implemented in parallel fashion using OpenMP, instead of forecast on sequential manner [18].

Lakshmi et al. discussed various parallel algorithms. They elaborated in details the parallel aspect of sorting algorithms by considering different factors [19]. Recent studies are mainly focused on general-purpose sorting using a custom comparison operator that rules out many specialized algorithms from the consideration, but it is helpful when data structure is not known in advance. The main conclusion is that parallel sorting algorithms are very hard to parallelize because they are data intensive [20]. In recent studies they have classified the algorithms into three categories. First category is of network sorting algorithms. In this category they have mentioned the techniques which was basically working on non-adaptive and repeated merging orders. However, in the context of sorting networks their first initiative was that in the more prevalent model of parallel computation the fundamental parallel merging strategies were eventually embedded. In which the data is interchanged simultaneously between lines on a sparse interconnected networks and this is all done by processors. Second category is of shared memory sorting algorithms. In this category they have mentioned the algorithms which needs a memory access pattern which is more adaptable than the category one sorting strategies. They have supposed that the in parallel computation the processor is performing tasks by sharing, reading and writing the access to an enormous memory poll by various degrees of connections and number of conflict resolution approaches. Shared memory sorting strategies are not that much feasible than the network sorting techniques when it comes to hardware perspective but when it comes to execution speed they were found much faster than the network techniques. At the end they have briefly summarized that both in the network and parallel strategies the processors were utilized. Third category is of the parallel file sorting strategies. In this category they have mentioned the both types of parallel strategies external and internal parallel sorting strategies that are using parallelism to give solution of problems involving large size data [21]. The performance of sorting applications can be improved via Code optimization with deep value profiling [22]. Li et al. performed detailed experimental analysis of various sorting algorithms in parallel manner. They have adopted OpenMP for parallel computing [23].

3. Sorting Algorithms

In this section, only the On(log n) class algorithms and their variants are discussed that are selected for experimental analysis.

A. Merge Sort

Merge sort is a divide and conquer strategy based algorithm. In this sorting technique, the input array is divided into two parts and these parts are further divided into two halves. After arranging the sub arrays into desired, the merge part combines the whole parts into one sorted array [16].

Algorithm:

1. Find out the middle integer of array to divide the array into two parts:
   middle m = (p+r)/2
2. merge_Sort called for first part:
   Call merge_Sort(array, p, q)
3. merge_Sort will be called again for second part:
   Call merge_Sort(array, p+q, r)
4. Merge the both parts sorted in step 2 and 3:
   Call merge(array, p, q, r)

B. Quick Sort

Quicksort is also based upon divide and conquer technique like merge sort. Initially, quicksort selects an element as pivot point and then it divides the given data list around the selected pivot point. The performance of quick sort is based upon the pivot selection. There exist many versions of quicksort due to pivot selection techniques [24].

Algorithm:

1. Choose the highest index value has pivot
2. Take two variables to point left and right of the list excluding pivot
3. Left points to the low index
4. Right points to the high
5. While value at left is less than pivot move right
6. While value at right is greater than pivot move left

7. If both step 5 and step 6 does not match swap left and right
8. If left ≥ right, the point where they met is new pivot

C. Heap Sort

Heap sorting is a technique which is based on the binary heap and it is a comparison based technique. This approach is similar to the selection sort in which the largest element is determined and then move the largest data element to the end of array or list. The process will go on for remaining data elements [25,26].

Algorithm:

1. Make a binary tree of available list.
2. Transform the Binary Tree into Min Heap.
3. Delete the root element from Min Heap using Heapify method.
4. Put the deleted element into the Sorted list.
5. Repeat the same until Min Heap becomes empty.
6. Display the sorted list.

D. Intro sort

Intro sort is a hybrid sort which was developed in 1997 by David musser in musser. Its performance is fast in some cases while it also comes up with average and worst performance in others. This technique is also known as introspective sorting technique. Its mechanism at beginning is same as quicksort and at the recursion step its mechanism is diverted to the heapsort when the recursive depth surpass a level based on the data being sorted. Its actual performance is near to the quicksort on quintessential sets of data elements [8].

E. Tim Sort

Tim sort is also a hybrid sorting technique which was firstly designed by Tim peters in year 2002. It was designed to give effectual results on many kinds of actual datasets. It is derived from the merge and insertion sorting strategies. Tim sort is a durable technique and beats every other sorting technique when it comes to time. It has On(log n) time complexity for worst case. The algorithm finds out the chain of the data that are already organized and uses that observation to sort the remaining data more efficiently. This is performed by combining an identified subsequence, called a run, with existing runs until targeted benchmark is archived. Tim sort is mostly used in python programming language [27].

F. Smooth Sort

Smooth sort is a variant of a heap sort. It was designed by Edsger Dijkstra and published it in year 1981. It is a comparison oriented sorting technique. It is similar to heap sort because it is also an in place strategy with upper bound On(log n). Its time complexity will come closer to O(n) if the given data is already sorted to some extent [28].

G. Library Sort

Library sort is a technique that works basically on an insertion sort. A. Bender et al. developed this technique in 2004 and they published it 2 years later in 2006. It is similar to insert sort but it comes with inconsistency in the array to speed up the successive insertions. Library sort is durable technique based on comparison. However, it has most of the chances of running in On(log n) time if we compare it to the quick sort, instead of insertion sort's O(n2). The methodology used for this advancement is very alike to the one used in skip list [29]. If we compare it to the insertion sort, the main disadvantage of library sort is that it needs an extra space for the jumps or gaps.

H. Cube Sort

Cube sort is a parallel sorting strategy. Cube sort was developed in 1992 and its hybrid version is redeveloped in 2014. It develops a self-stabilized multi-dimensional array from the data to be ordered [30]. In Sorting, when each element is inserted the cube then the elements swiftly transformed to an array. Cube sort has On(log n) time complexity for the worst case.

I. Tree Sort

A tree sort is a technique that develops a binary search tree from the data to be organized and then travel across so that the data comes out in sorted order. Douglas in 1959, Windley in 1960 and later Hibbard in 1962 proposed binary trees and implemented it in applications of sorting, searching, and data maintenance [31]. Its most often use is organizing data elements online: after every insertion, the data seen so far is available in well-organized order.

J. Tournament Sort

Tournament sort is a data organizing technique. Tournament sort is a variation of heapsort. The name of the technique is due to its resemblance to a single elimination competition where there are a lot of players or teams which plays in two-sided matches. Each match's performance compares the players with one another and the player with

best stats is promoted to play the match at the coming levels [32].

K. Block Sort

Block sort is a sorting algorithm integrating at least two merge operations along with insertion sort to come at On(log n) in-place durable arrangement. Kim et al. introduced one practical strategy for O(log n) in place merging in 2008. The technique gets its name by the observation that merging two organized data lists, list A and list B, is equal to dividing A into same sized blocks, inserting each A block into B with some rules, and merging AB pairs [33].

L. Comb Sort

Dobosiewicz et al. originally developed it in year 1980. Stephen Lacey and Richard Box modified and redesigned it in 1991. This technique is basically an advancement of the Bubble Sort. Bubble Sort mechanism works by always checking and managing the adjacent values. So all swappings are done one by one. By breaching the size more than 1 the comb Sort improves over a Bubble Sort [13].

4. Experimentation and Results

This section is further divided into experimental setup, actual experimentation and discussion.

A. Experimental Setup

The experimental setup is configured by using HP Elitebook 840 G2 5th generation laptop. The system based upon i5 processor (having four processing cores) clocked at 2.30 GHZ with 64 bit Windows operating system. It has 4 GB DDR3 RAM and standard GPU installed from AMD. The sorting algorithms are implemented using C++ language and specifically dev IDE. For parallelism, the standard OpenMP library is used. OpenMP itself divides the task into multiple threads and forward to available cores for attaining highest performance. We will use integer data for sorting in our experimentation and the amount of data will lie between 210 to 220 integer numbers. Table 1 shows the different type test cases that are executed for analysis

Table 1: Units for Magnetic Properties

Test Case	Nature of Data	Description
1	Random Integers	Unorganized data elements
2	Sorted Integers	Organized data elements
3	Reverse Sorted Integers	Reversed Organized elements

B. Experimentation

Experimental results are shown using graphs and charts. All the selected 12 sorting techniques are implemented and execution time in seconds is recorded for given data items. Following graphs show the state of the art comparative results.

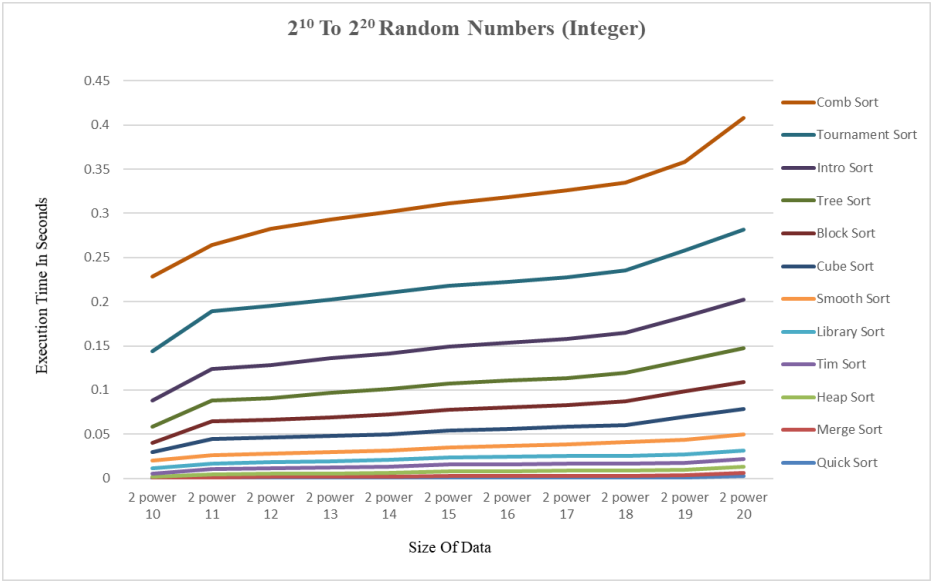

Fig. 2. Execution Time in Seconds for Sorting 2^{10} to 2^{20} Random Integers

The graph in figure 2 illustrates the processor execution time in seconds of different algorithms for sorting the random integers data. The data range or size of data is shown in x-axis that increases in standard ratio with two power n. The value of n grows from 10 to 20. While on y-axis, the actual execution time is shown. For sorting the given range of data, above mentioned 12 sorting approaches are executed. Quick sort performs very well with respect to other techniques. Merge sort, heap sort and tim sort consumes average time to sort the given range. But remaining algorithms comparatively consumes very high time for arranging the data. Specifically comb sort and tournament sort performs worst with regard to execution time.

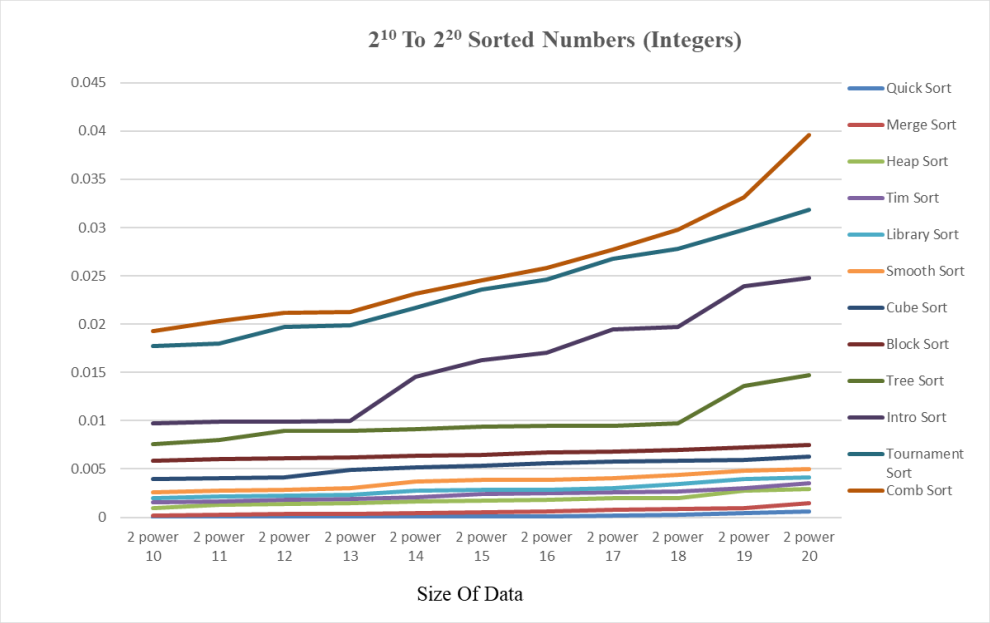

Fig. 3 Execution Time in Seconds for Sorting 2^{10} to 2^{20} Sorted Integers

The graph in figure 3 elaborates the actual processing time in seconds of various above mentioned sorting algorithms for arranging the already sorted integers. The data range is shown in x-axis that grows with ratio of two power n. The value of n exists between 10 to 20. Actual execution time and sorting techniques are shown on y-axis. Again, Quick sort and merge sort almost equally performs well with respect to other techniques. Comb sort, tournament sort and cube sort consumes very high time with respect to other techniques. Remaining all techniques take average time to sort the given range of data.

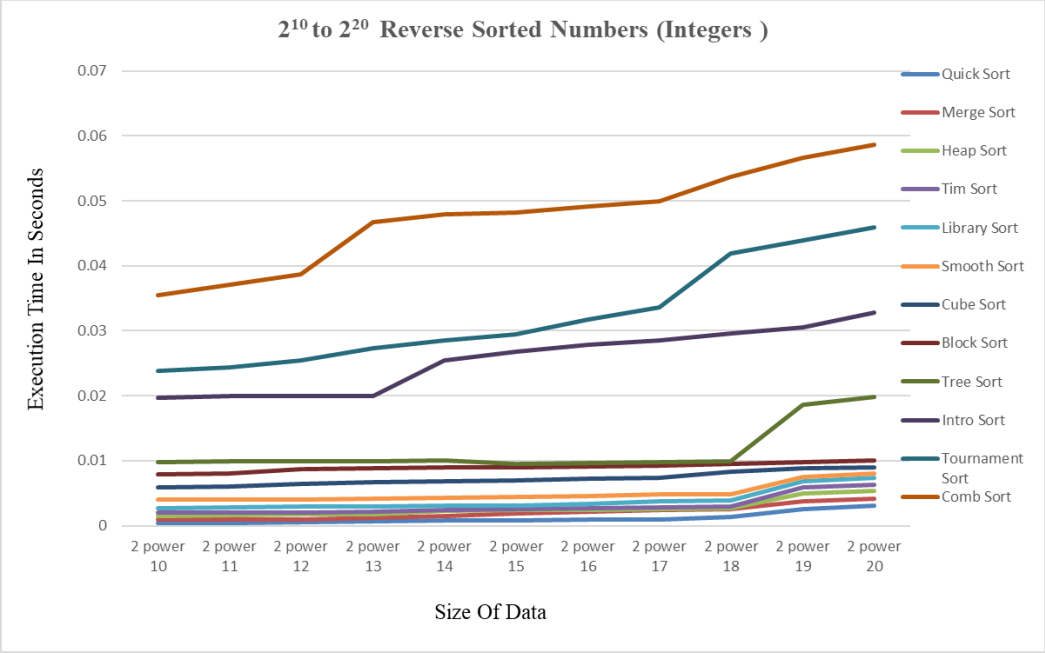

Fig. 4 Execution Time in Seconds for Sorting 2^{10} to 2^{20} Reverse Sorted Integers

The graph in figure 4 illustrates the processor execution time in seconds of different sorting algorithms for arranging the reversed integers data into sorted order. The data range or size of data is shown in x-axis that increases in standard ratio with two power n. The value of n grows from 10 to 20. While on y-axis, the actual execution time is shown. For sorting the given range of data, above mentioned 12 sorting approaches are executed. Quick sort performs well with respect to other techniques. Merge sort, heap sort, library sort, cube sort, block sort and tim sort consumes average time to sort the given range. But remaining algorithms comb sort, tournament sort, intro sort and tree sort comparatively consumes very high time for arranging the data.

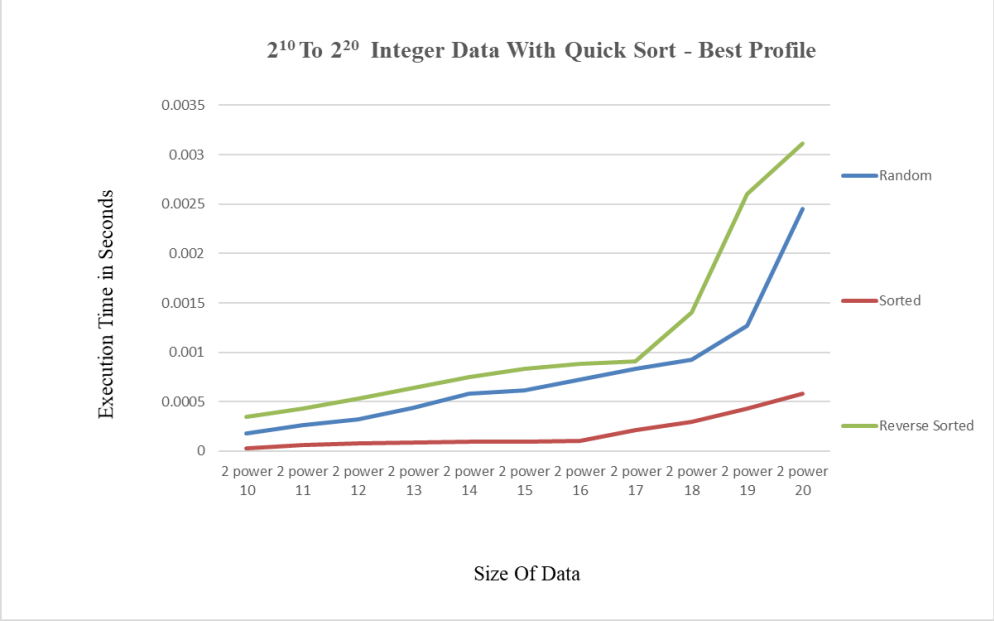

Fig. 5 Best Profile of Execution Time in Seconds for Sorting 21^0 to 2^{20} Size Data of All Test Cases

The graph in figure 5 shows the best profile of execution time in seconds of quick sort for arranging the given data. The range of data is shown on x-axis that grows from 2 power 10 to 2 power 20. The graph shows the processing time of all three cases; random data, sorted data as well as reversed sorted data. Comparatively, quick sort performs very well with respect to other algorithms. All the algorithms fall under the same time complexity class but show different execution behavior for different nature of data. Overall for integers data, quick sort performs best.

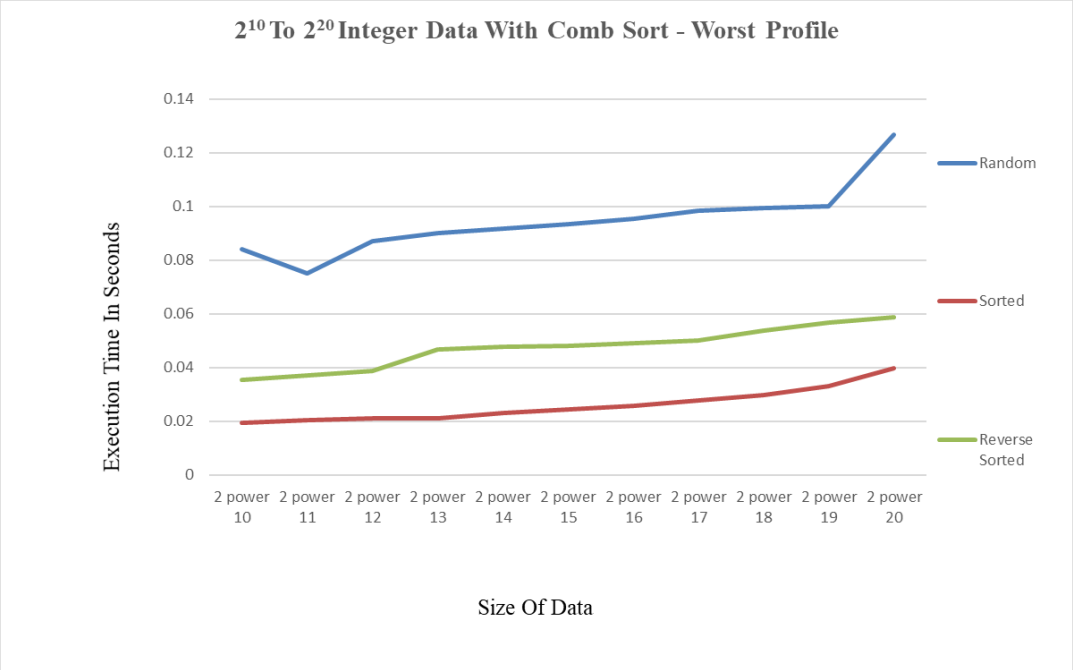

Fig. 6 Worst Profile of Execution Time in Seconds for Sorting 2^{10} to 2^{20} Size Data of All Test Cases

The graph in figure 6 illustrates the worst profile of execution time in seconds of comb sort for arranging the given data. The range of data is shown on x-axis that grows from 2 power 10 to 2 power 20. The graph shows the processing time of all three cases; random data, sorted data as well as reversed sorted data. Comparatively, comb sort performs inefficient with respect to other algorithms. In all test cases, comb sort consumes more execution time for sorting the given range of data. The results of tournament sort and intro sort are also unacceptable for time critical applications.

A. Discussion

After experimentation, we draw graphs on the basis of comparative tables for analyses of results for all On(log n) class sorting algorithms. Results show that the quick sort is the efficient sort with all data types, whether the data is sorted, random or reverse sorted. It holds its position as the best sorting algorithm in all of the cases for small to large amount of data. Merge sort and heap sort are near to each other when comes to performance. They also show efficient results in terms of execution time and considered as the 2nd best sorting techniques. After these algorithms, tim sort, library sort, smooth sort, cube sort and block sort performance is average as they are also near to each other when it comes to performance. In last, comb sort, tree sort, intro sort and tournament sort are the algorithms with worst results in terms of highest execution time. These techniques have continuous growth in execution time for random data but when these are executed with sorted and reverse sorted data, the growth becomes irregular. Sudden upshifts could be clearly seen in graphs.

5. Conclusion and Future Work

Sorting is essentially used in many applications of computer science. Number of sorting algorithms has been developed and categorized with the help of time complexity classes. Various algorithms of same complexity class may perform different with different nature of data. The same class sorting algorithms are further analyzed using execution time under same experimental setups. In this research, an experimental analysis is conducted among different sorting algorithms of On(log n) class. 12 well-known algorithms are

executed under same experimental setup with three test cases. Test cases consist of random integers data, sorted integers and reversed sorted integers that grows from 2 power 10 to 2 power 20 size. All the algorithms are executed in parallel manner to achieve the highest performance from CPU. OpenMP library is used to implement standard parallelism. Experimental results show that the quick sort performs outstanding for all test cases. Merge sort and heap sort are also near to quick sort. Tim sort, library sort, smooth sort, cube sort and block sort performs average in terms of execution time. Comb sort, tree sort, intro sort and tournament sort performs worst in terms of execution time. This experimental analysis is done on integers data. May be the behavior of sorting algorithms differ for other nature of data.

In future, we will analyze the same algorithms on different type of data like floats, logarithmic, statistical and available standard datasets. We will analyze these algorithms under different experimental setup with Java using MPJ Express parallelism library to identify the behavior of sorting techniques in different environments.

References

[1] Amato NM, Iyer R. A comparison of parallel sorting algorithms on different architectures. Texas A M Univ. … [Internet] 1998;Available from: http://parasol-www.cs.tamu.edu/dsmft/publications/psort-tr.pdf%5Cnpapers2://publication/uuid/890192B7-5654-4280-955A-4EF8912C08A6

[2] "Big Data" Is No Longer Enough: It's Now All About "Fast Data" [Internet]. [cited 2019 Oct 31];Available from: https://www.entrepreneur.com/article/273561

[3] Albattah W. Analysis of passwords: Towards understanding of strengths and weaknesses. Int. J. Adv. Appl. Sci. 2018;5:51–60.

[4] et al. F. Finding the top conferences using novel ranking algorithm. Int. J. Adv. Appl. Sci. 2017;4:148–52.

[5] Cormen TH. Introduction to algorithms. MIT Press; 2009.

[6] Sharma SK. Performance Analysis of Parallel Algorithms on Multi-core System using OpenMP. Int. J. Comput. Sci. Eng. Inf. Technol. 2012;2:55–64.

[7] Chandra R. Parallel programming in OpenMP. Morgan Kaufmann Publishers; 2001.

[8] Musser DR. Introspective Sorting and Selection Algorithms. Softw. Pract. Exp. [Internet] 1997;27:983–93. Available from: http://www.cs.rpi.edu/%7B~%7Dmusser/gp/Intro sort.ps

[9] Krastev P. Introduction To Parallel Computing and Openmp Objectives :

[10] Tan et al. Load-balanced parallel architectures for 2-D water quality model PARATUNA-WQ on OpenMP. Int. J. Adv. Appl. Sci. 2017;4:94–9.

[11] Mateescu G. Parallel Computing with OpenMP on distributed shared memory platforms. 2002;Available from: http://cs.pub.ro/%7B~%7Dapc/2003/resources/openmp/openmp.pdf

[12] Huang S-C. Parallel Computing and OpenMP Tutorial. 2013;

[13] Elkahlout AH, Maghari AYA. A comparative Study of Sorting Algorithms Comb , Cocktail and Counting Sorting. Int. Res. J. Eng. Technol. [Internet] 2017;4:1387–90. Available from: https://irjet.net/archives/V4/i1/IRJET-V4I1249.pdf

[14] Alter S. Defining information systems as work systems: implications for the IS field. Eur. J. Inf. Syst. [Internet] 2008 [cited 2019 Feb 10];17:448–69. Available from: https://www.tandfonline.com/doi/full/10.1057/ejis.2008.37

[15] Cormen TH, Leiserson CE, Rivest RL, Stein C. Section 35.1: The vertex-cover problem. MIT Press and McGraw-Hill; 2001.

[16] Shen HL. Optimal parallel algorithm of merge sort based on OpenMP. Appl. Mech. Mater. 2014;556–562:3400–3.

[17] Ravindra M, Rao RS. An upgraded binary bat algorithm approach for optimal allocation of PMUs in power system with complete observability. Int. J. Adv. Appl. Sci. 2017;4:33–9.

[18] Akhmetova D, Iakymchuk R, Ekeberg O, Laure E. Performance study of multithreaded MPI and Openmp tasking in a large scientific code. Proc. - 2017 IEEE 31st Int. Parallel Distrib. Process. Symp. Work. IPDPSW 2017 2017;756–65.

[19] Lakshmi varahan S, Dhall SK, Miller LL. Parallel Sorting Algorithms. Adv. Comput. 1984;23:295–354.

[20] Wu H. Parallel Computing Using GPUs. Statistics (Ber). 2011;6:90–4.

[21] Zecena I, Zong Z, Ge R, Jin T, Chen Z, Qiu M. Energy consumption analysis of parallel sorting algorithms running on multicore systems. 2012 Int. Green Comput. Conf. IGCC 2012 2012;6–11.

[22] Khan MA. Improving performance through deep value profiling and specialization with code transformation. Comput. Lang. Syst. Struct. 2011;37:193–203.

[23] Li JM, Zhang J. The performance analysis and research of sorting algorithm based on OpenMP. 2011 Int. Conf. Multimed. Technol. ICMT 2011 2011;3281–4.

[24] Yaroslavskiy V. Dual-Pivot Quicksort [Internet]. 2009. Available from: http://iaroslavski.narod.ru/quicksort/DualPivotQuicksort.pdf

[25] MacKay D. Heapsort, Quicksort, and Entropy [Internet]. users.aims.ac.za/~mackay; 2005. Available from: http://users.aims.ac.za/%7B~%7Dmackay/sorting/sorting.html

[26] Katajainen J. The Ultimate Heapsort. Aust. Comput. Sci. Commun. [Internet] 1998;20:87–96. Available from: http://hjemmesider.diku.dk/%7B~%7Djyrki/Myris/Kat1998C.html

[27] Auger N, Nicaud C, Pivoteau C. Merge Strategies: from Merge Sort to TimSort. hal-01212839 [Internet] 2015;Available from: https://hal-upec-upem.archives-ouvertes.fr/hal-01212839

[28] Dijkstra EW. Smoothsort – an alternative to sorting in situ (EWD-796a) [Internet]. Center for American History, University of Texas at Austin; Available from: http://www.cs.utexas.edu/users/EWD/ewd07xx/EWD796a.PDF

[29] Faujdar N, Ghrera SP. A detailed experimental analysis of library sort algorithm. 12th IEEE Int. Conf. Electron. Energy, Environ. Commun. Comput. Control (E3-C3), INDICON 2015 2016;1–6.

[30] Cypher R, Sanz JLC. Cubesort: A parallel algorithm for sorting N data items with S-sorters. J. Algorithms 1992;13:211–34.
[31] Singh A, Garg D. Implementation and Performance Analysis of Exponential Tree Sorting. Int. J. Comput. Appl. 2011;24:34–8.
[32] Tournament sort - Wikipedia [Internet]. [cited 2019 Nov 13];Available from: https://en.wikipedia.org/wiki/Tournament_sort
[33] Fenwick P, Fenwick P. Block Sorting Text Compression - Final Report. 1996 [cited 2020 Jan 21];Available from: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.7580

Mubashir Ali received his MCS and MS degree in Computer Science from Bahauddin Zakariya University Multan, Pakistan. He is currently associated as a Lecturer with Department of Software Engineering, Lahore Garrison University, Pakistan. His work has appeared in many leading journals and international conferences. His research interests include Design and Analysis of Complex Algorithms, Internet of Things, Wireless Sensor Networks and Cloud Computing.

Zarsha Nazim received her MPhil degree in Computer sciences recently from Forman Christian University, Lahore. She is currently working as a lecturer in Garrison university Lahore, Pakistan. Her research interests include Software engineering, Human Computer Interaction and usability evaluation. She has several publications in prestigious journals and conferences.

Wajid Ali received his BSCS and MSCS degrees respectively in 2016 and 2018 from Department of Computer Science, Bahauddin Zakariya University Multan, Pakistan. He is currently working in Information Technology Department of Food Authority Punjab, Pakistan. His resaerch interests are Algorithm Analysis, Parallel Processing and Performance Estimation.

Aamir Hussain received the Ph.D. degree in computer science and technology from the School of Computer Science and Technology, Wuhan University of Technology, China, in 2016. He is currently an Assistant Professor with the Department of Computer Science, Muhammad Nawaz Shareef University of Agriculture Multan, Pakistan. His research interests include Wireless Sensor Networks, Internet of Things, and Software-Defined Networks (SDN).

Nosheen Kanwal completed her MCS in 2016 and MSCS in 2018 from Bahauddin Zakariya University Multan, Pakistan. Her research interests including Algorithms, Netowrks and botnet malware detection. Now she is serving as SSE(CS) at Education Department Punjab and as resaercher under the umbrella of BZU. She also works on different manuals of BS Classes.

Mahnoor Khalid Paracha completed his MSCS from Bahauddin Zakariya University, Multan, Pakistan. Currently she is affiliated with Punjab School Education Department (PSED) and serving in multiple institutions as a visiting faculty member. Her research areas are Algorithms Analysis, Network Protocol Analysis and Data Communication.