**Acknowledgement**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

We express our deepest gratitude to our guide Mr. H Manoj T Gadiyar, Asst. Professor, Department of Computer Science, for his valuable guidance and encouragement while doing this mini project work.

We are indebted to Dr. Thyagaraju G S, Head of the Department, and Dr. Ashok Kumar T, Principal, for their advice and suggestions at various stages of the work. We also extend our heartfelt gratitude to Prof. Guruprasad M S for his assistance.

We also extend our thanks to the management of SDM Institute of Technology, Ujire, for providing an excellent study environment, reference materials and laboratories facilities. We remain grateful to the co-operation and help rendered by the teaching and non-teaching staff of the department.

Rajashree Hebbar V

**Abstract**

Sorting algorithms have attracted a great deal of attention and study, as they have numerous applications to Mathematics, Computer Science and related fields. In this thesis, we first deal with the mathematical analysis of the Quick sort algorithm and its variants. Specifically, we study the time complexity of the algorithm and we provide a complete demonstration of the variance of the number of comparisons required, a known result but one whose detailed proof is not easy to read out of the literature.

We also examine variants of Quicksort, where multiple pivots are chosen for the partitioning of the array. The rest of this work is dedicated to the analysis of finding the true order by further pairwise comparisons when a partial order compatible with the true order is given in advance. We discuss a number of cases where the partially ordered sets arise at random. To this end, we employ results from Graph and Information Theory. Finally, we obtain an alternative bound on the number of linear extensions when the partially ordered set arises from a random graph.

**Table of Contents**

**Page No.**

**Acknowledgement i**

**Abstract ii**

**Table of Contents iii**

**List of Figures viii**

**Chapter 1 Introduction 1**

**Chapter 2 Literature Review 2**

**Chapter 3 Problem formulation**

3.1 General 3

3.2 Problem statement 4

3.3 Aim of the work 4

3.4 Objectives of the present study 4

3.5 Summary 5

**Chapter 4 Materials and Methodology**

4.1 Functional Requirements 6

4.2 Non Functional Requirements 6

4.3 Software specifications 7

4.4 Languages 7

4.5 Flow chart 1 0

**Chapter 5 Results and Discussion**  **11**

**Chapter 6 Conclusions and Scope for future work 14**

**Bibliography** **15**

**Personal Profile**   **16**

**List of Figures**

|  |  |  |
| --- | --- | --- |
|  |  | Page No. |
| Figure 4.5 | Flowchart of Quicksort | 8 |
| Figure 4.6 | Working of Quicksort algorithm | 10 |
| Figure 5.1 | Figure of the Menupage | 11 |
| Figure 5.2 | Figure describes the random numbers before sorting | 11 |
| Figure 5.3 | Figure describes the bars of unsorted elements | 12 |
| Figure 5.4 | Figure describes the process of sorting | 12 |
| Figure 5.5 | Figure describes the bars of sorted elements | 13 |
| Figure 5.6 | Figure shows sorted numbers | 13 |

**Chapter 1**

**Introduction**

One of the most widely studied practical problems in computer science is sorting: the use of a computer to put files in order. A person wishing to use a computer to sort is faced with the problem of determining which of the many available algorithms is best suited for his purpose. This task is becoming less difficult than it once was for three reasons. First, sorting is an area in which the mathematical analysis of algorithms has been particularly successful: we can predict the performance of many sorting methods and compare them intelligently. Second, we have a great deal of experience using sorting algorithm, and we can learn from that experience to separate good algorithms from bad ones. Third, if the tile fits into the memory of the computer, there is one algorithm, called Quicksort was invented by C. A. R. Hoare. which has been shown to perform well in a variety of situations. Not only is this algorithm simpler than many other sorting algorithms, but empirical and analytic studies show that Quicksort can be expected to be up to twice as fast as its nearest competitors. The method is simple enough to be learned by programmers who have no previous experience with sorting, and those who do know other sorting methods should also find it profitable to learn about quicksort. Note that the order may be numerical, alphabetical or any other transitive relation defined on the keys. In this work, the analysis deals with numerical order, where the keys are decimal numbers and we particularly focus on Quicksort algorithm and variants of it.

The three steps of quicksort are Divide: Rearrange the elements and split the array into two subarrays such that each element on the left subarray is less than or equal to the middle element and each element in the right subarray is greater than the middle element. Conquer : Recursively sort the two subarrays. Finally combine the elements.

**Chapter 2**

**Literature Review**

The quicksort algorithm was developed in 1959 by Tony Hoare. As a part of the translation process, insertion sort, would be slow, he quickly came up with a new idea that was Quicksort. He wrote a program in Mercury Auto code for the partition but could not write the program to account for the list of unsorted segments. On return to England, he was asked to write code for Shellsort as part of his new job. Hoare mentioned to his boss that he knew of a faster algorithm and his boss bet sixpence that he did not. His boss ultimately accepted that he had lost the bet. Later, Hoare learned about ALGOL and its ability to do recursion that enabled him to publish the code in Communications of the Association for Computing Machinery, the premier computer science journal of the time.

Quicksort gained widespread adoption, appearing,. Hence, it lent its name to the C standard library subroutine sort. thesis in 1975 is considered a milestone in the study of Quicksort where he resolved many open problems related to the analysis of various pivot selection schemes including Sample sort, adaptive partitioning by Van Emden as well as derivation of expected number of comparisons and swaps. Bentley and McIlroy incorporated various improvements for use in programming libraries, including a technique to deal with equal elements and a pivot scheme known as pseudo median of nine, where a sample of nine elements is divided into groups of three and then the median of the three medians from three groups is chosen. Jon Bentley described another simpler and compact partitioning scheme in his book Programming Pearls that he attributed to Nico Lomuto.. Lomuto's partition scheme was also popularized by the textbook Introduction to Algorithms although it is inferior to Hoare's scheme because it does three times more swaps on average and degrades to O(n2) runtime when all elements are equal, self-published source.

In 2009, Vladimir Yaroslavskiy proposed the new dual pivot Quicksort implementation. In the Java core library mailing lists, he initiated a discussion claiming his new algorithm to be superior to the runtime library's sorting method, which was at that time based on the widely used and carefully tuned variant of classic Quicksort by Bentley and McIlroy.

**Chapter 3**

**Problem Formulation**

**3.1 General**

Quicksort (sometimes called partition-exchange sort) is an efficient sorting algorithm, serving as a systematic method for placing the elements of a random access file or an array in order. Like merge sort, quicksort uses divide-and-conquer, and so it's a recursive algorithm. The way that quicksort uses divide-and-conquer is a little different from how merge sort does. In merge sort, the divide step does hardly anything, and all the real work happens in the combine step. Quicksort is the opposite: all the real work happens in the divide step. In fact, the combine step in quicksort does absolutely nothing. Quicksort has a couple of other differences from merge sort. Quicksort works in place. And its worst-case running time is as bad as selection sort's and insertion sort's: Theta(n^2) Θ(n2 ). But its average-case running time is as good as merge sort's: Theta(n \log\_2 n) Θ(n log 2n). So why think about quicksort when merge sort is at least as good? That's because the constant factor hidden in the big-Θ notation for quicksort is quite good.

In practice, quicksort outperforms merge sort, and it significantly outperforms selection sort and insertion sort. To improve the efficiency of the sort, there are clever ways to choose the pivot value such that it is extremely unlikely to end up with an extreme value. One such method is to randomly select three numbers from the set of data and set the middle one as the pivot. Though the comparisons make the sort slightly slower, a "good" pivot value can drastically improve the efficiency of quicksort.

* 1. **Problem statement**

The major disadvantage of other sorting is that the merging of two arrays requires an extra, temporary array -this means that sorting requires 2x as much space as the array itself -can be an issue if space is limited! in-case of mergesort exists, but is complicated and has worse performance -to achieve the overall running time of O(N log N) it is critical that the running time of the merge phase be linear.QuickSort is better than other sorting algorithms with same asymptotic complexity O(nlogn) (merge sort, heap sort). Even though quicksort has O(n^2) in worst case, it can be easily avoided with high probability by choosing the right pivot. Its cache performance is higher than other sorting algorithms.

**3.3 Aim of the work**

 Purpose of  Quick sort is obvious by it's name- sort the given elements. Average asymptotic sorting time is O(n log(n) ) where n is the number of elements to be sorted. Merge Sort has same order of running time but Quick Sort is constant times faster than Merge Sort.

**3.4 Objectives of the present key**

1. Quicksort describe the quick sort algorithm, it’s partition function and How it partitions an array into three parts, that is, without creating extra arrays(like merge sort).
2. Analyze its running time under different data conditions that is best, average, worst case complexity of quicksort
3. Quicksort is in pace algorithm and explains how it different from any other sorting algorithm.
4. What is the advantage and disadvantage of quick sort.

**3.5 Summary**

Quick sort, like Merge Sort, is a divide-and-conquer sorting algorithm. The premise of quicksort is to separate the "big" elements from the "small" elements repeatedly. The first step of the algorithm requires choosing a "pivot" value that will be used to divide big and small numbers. Each implementation of quicksort has its own method of choosing the pivot value--some methods are much better than others. The implementation below simply uses the first element of the list as the pivot value. Once the pivot value has been selected, all values smaller than the pivot are placed toward the beginning of the set and all the ones larger than the pivot are placed to the right. This process essentially sets the pivot value in the correct place each time. Each side of the pivot is then quick sorted. Ideally, the pivot would be selected such that it were smaller than about half the elements and larger than about half the elements. Quicksort has a couple of other differences from merge sort. Quick sort works in place and its worst-case running time is bad as selection sort’s and insertion sorts’s: Θ(n2). But its average-case running time is as good as merge sorts: Θ(n logn).

**Chapter 4**

**Materials and Methodology**

In the project “SIMULATION OF QUICKSORT”, the computer graphics concepts along with the OpenGL functions are used. This project is to represent a graphical figure in 2Dimentional model. OpenGL provides a set of commands to render 3D objects that means you provide the data in an OpenGL usable form and OpenGL will show this data on the screen. The project is intended to depict the usage of graphics for prototyping the various geometrical models. The scope of the project is to portray a 2D/3D environment that provides the user with the example of various basic transformations and shading techniques available in OpenGL. It provides most of the features that a 2D/3D graphics editor should have. It is developed in C. It has been implemented in UBUNTU platform.

**4.1 Functional Requirements**

* glClear();
* glColor();
* glutWindowSize();
* glutCreateWindow();
* glutWindowPosition();
* glutInit();
* glutInitDisplayMode();
* glTranslatef();
* glutDisplayFunc();
* glutMainLoop();

**4.2 Non Functional Requirements**

Hardware specification Processor : Intel CORE i3

Clock speed : 2.0GHz

Hard disk : 1 TB

Keyboard : QWERTY

RAM : 4 GB

Operating system : Ubuntu 18.04 LTS & Higher versions Input Output Console for interaction

**4.3 Software specification**

• OpenGL libraries

**4.4 Languages**

• C

**4.5 Architecture:**

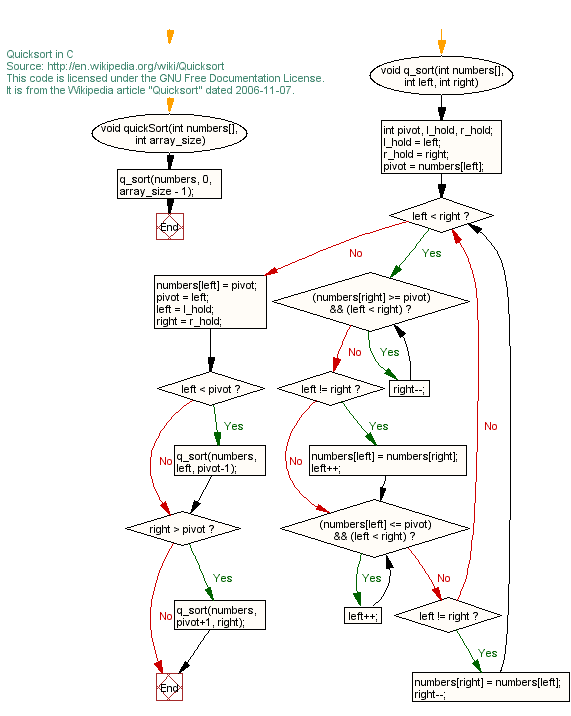
****

Figure 4.5: Flow Chart of quicksort

**4.6 Quick Sort Algorithm: Steps on how it works:**

1. Find a “pivot” item in the array. This item is the basis for comparison for a single round.
2. Start a pointer (the left pointer) at the first item in the array.
3. Start a pointer (the right pointer) at the last item in the array.
4. While the value at the left pointer in the array is less than the pivot value, move the left pointer to the right (add 1). Continue until the value at the left pointer is greater than or equal to the pivot value.
5. While the value at the right pointer in the array is greater than the pivot value, move the right pointer to the left (subtract 1). Continue until the value at the right pointer is less than or equal to the pivot value.
6. If the left pointer is less than or equal to the right pointer, then swap the values at these locations in the array.
7. Move the left pointer to the right by one and the right pointer to the left by one.
8. If the left pointer and right pointer don’t meet, go to step 1.

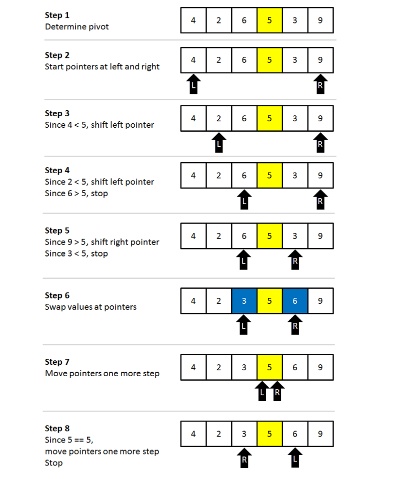


Figure 4.6:working of quicksort algorithm

**Chapter 5**

**Results and Discussion**

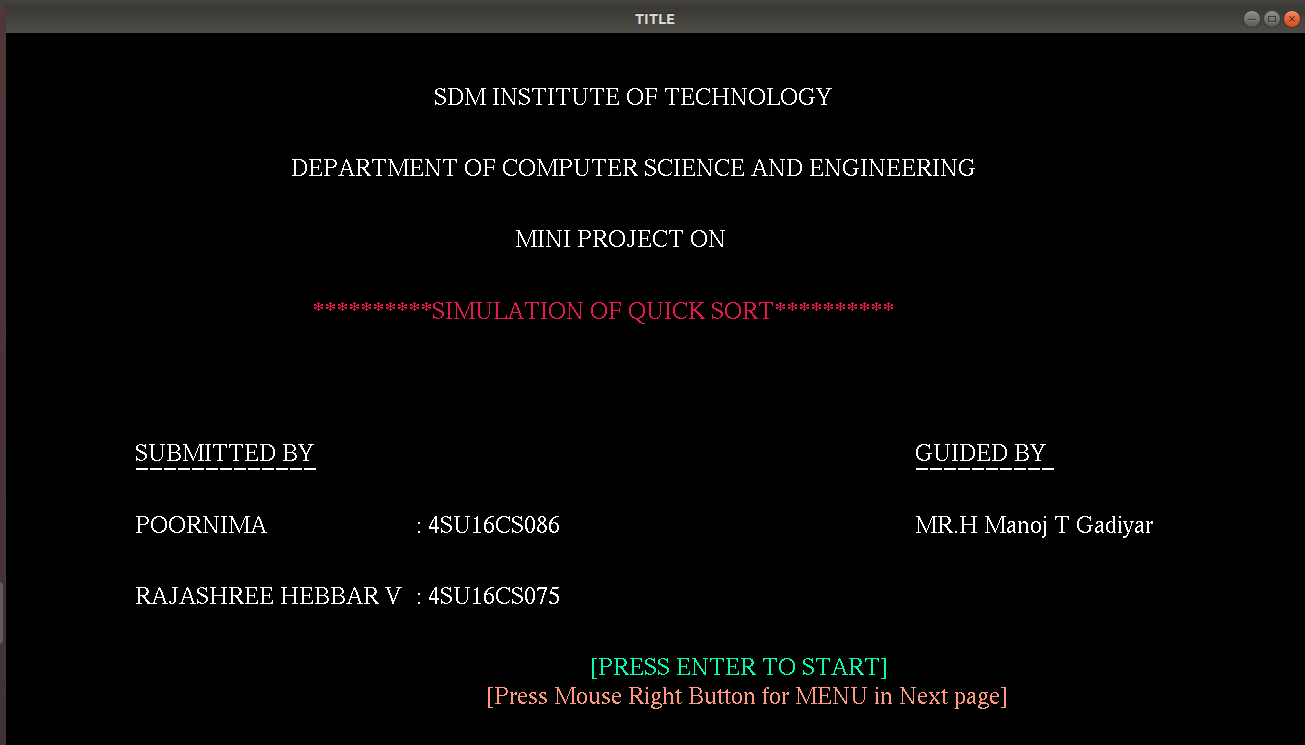


Figure 5.1: Figure of the Menupage

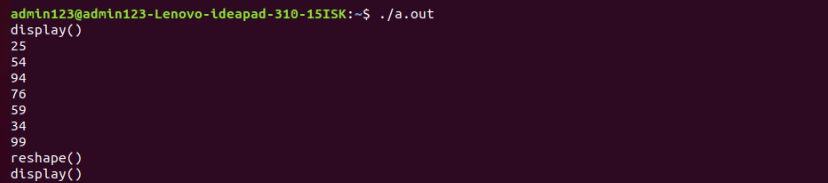


Figure 5.2: Figure describes the random numbers before sorting.

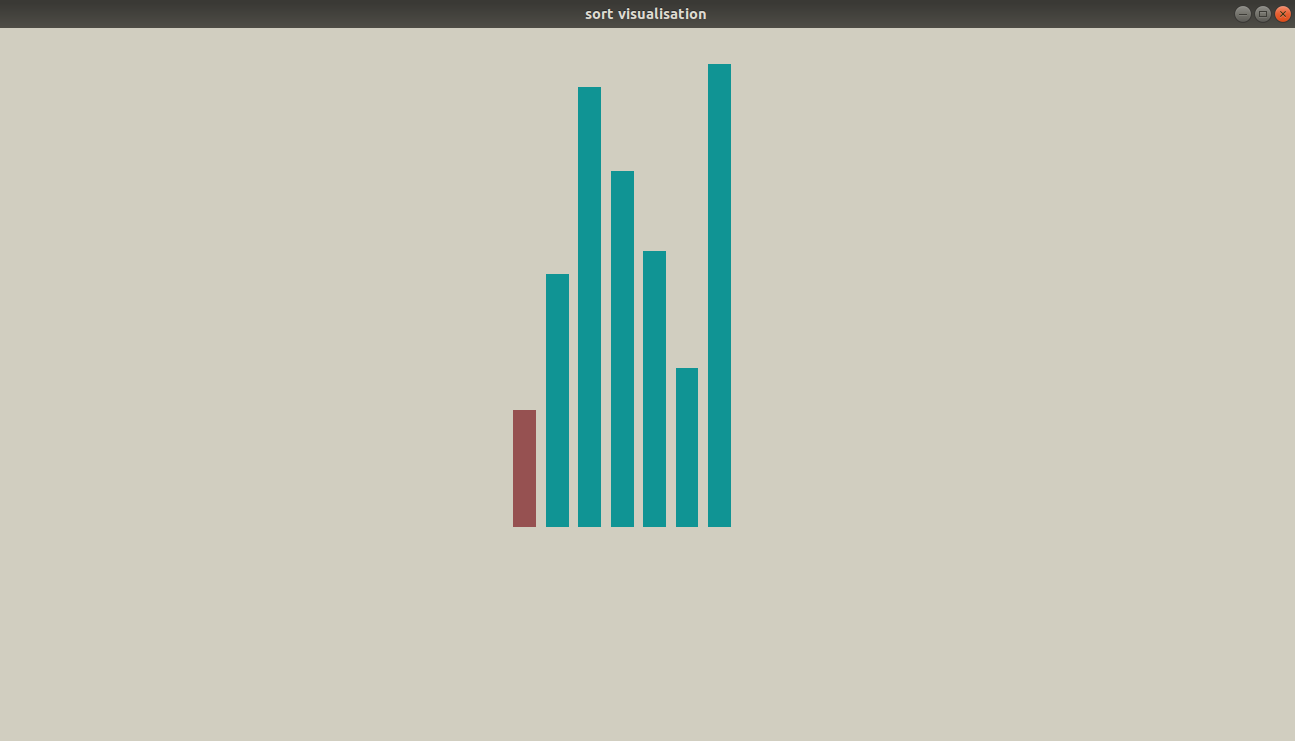


Figure 5.3:Figure describes the bars of unsorted elements.

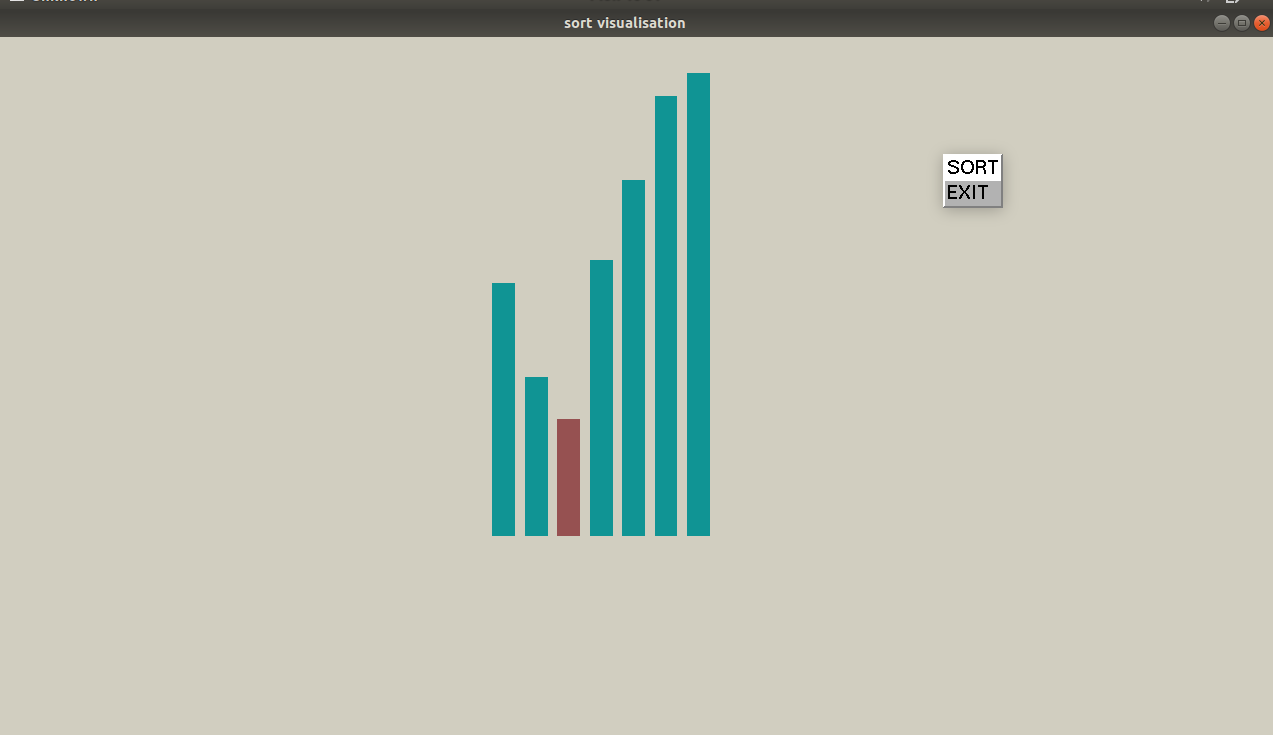


Figure 5.4: Figure describes the process of sorting

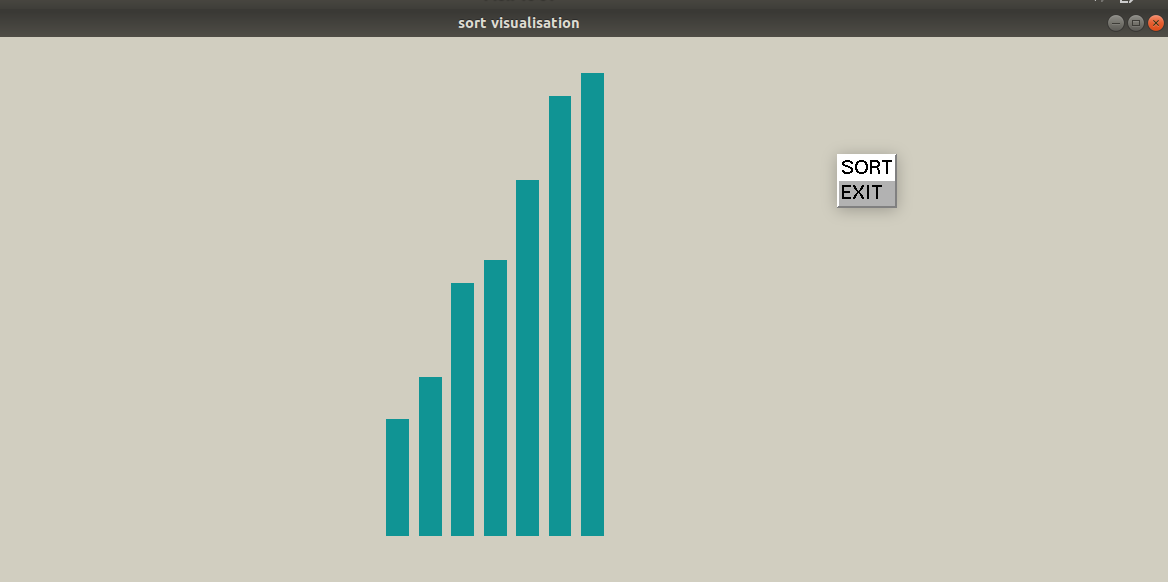


Figure 5.5: Figure describes the bars sorted elements

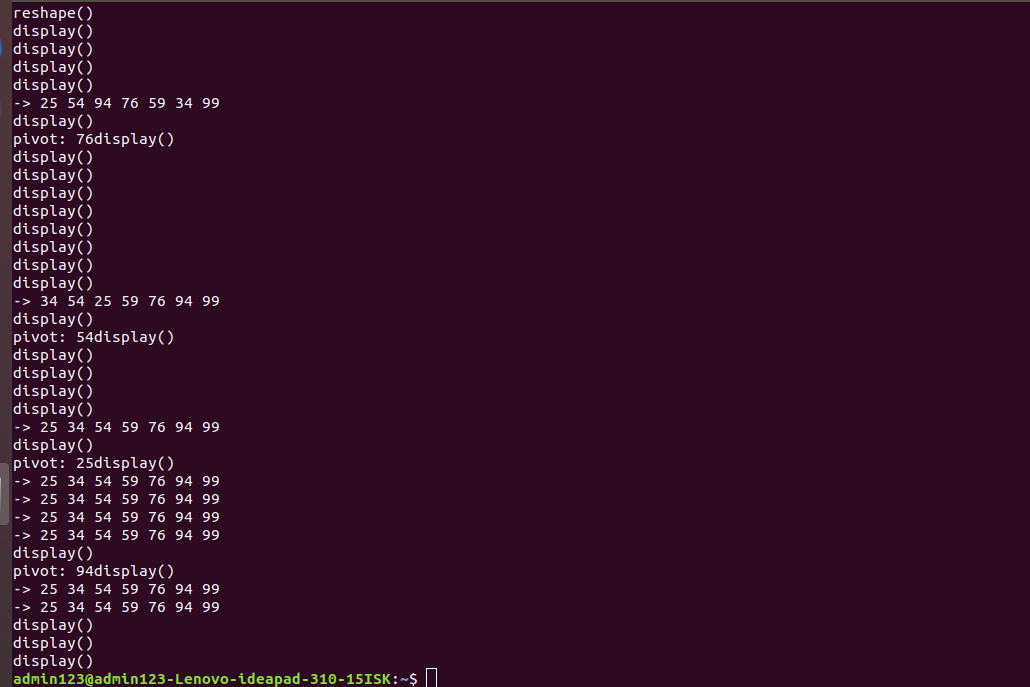


Figure 5.6 :Figure shows sorted number

**Chapter 6**

**Conclusion and Scope for future work**

Quicksort turns out to be the fastest sorting algorithm in practice. It has a time complexity of (N log(N)) on the average. However, in the (very rare) worst case quicksort is as slow as Bubblesort, namely in O(N2 ). There are sorting algorithms with a time complexity of O(N log(N)) even in the worst case, e.g. Heapsort and Mergesort. But on the average, these algorithms are by a constant factor slower than quicksort. It is possible to obtain a worst case complexity of O(N log(N)) with a variant of quicksort (by choosing the median as comparison element).

Quicksort based on a dynamic pivot selection technique. In order to determine the pivots for the next level recursive calls, the integer average of the values larger than the pivot is passed as the pivot value of the recursive call for the right sub array. Likewise, the integer average of the values less than the pivot is passed as the pivot value of the recursive call for the left sub array. This pivot selection technique helps in successively splitting the array into nearly equal halves which in turn improves the efficiency of the Quicksort algorithm.

Using the dynamic pivot selection technique, the worst case scenario in a typical QuickSort algorithm is turned into a best case for the QuickSort algorithm with Θ(n) runtime requirement. The QuickSort algorithm runs with Θ(n log n) in the worst case which is confirmed by experiments. Disadvantages of quicksort is difficulty of implementing the partitioning algorithm and the average efficiency for the worst case scenario. 3D implementation of quicksort can be done in future and testing process

**Bibliography**

1. Donald Hearn & Pauline Baker: Computer Graphics with OpenGL Version, 3rd Edition, Pearson Education,2011
2. Edward Angel: Interactive Computer Graphics- A Top Down approach with OpenGL, 5th edition. Pearson Education, 2008
3. James D Foley, Andries Van Dam, Steven K Feiner, John F Huges Computer graphics with OpenGL: pearson education
4. [Online] http://en.wikipedia.org/wiki/Quicksort
5. [Online] http://cs.indstate.edu

**Personal Profile**

|  |  |
| --- | --- |
| **Mr. H Manoj T Gadiyar,**  Asst. Prof.  Project Guide | **Prof. H Manoj T Gadiyar** B.E from VTU Belagavi in the year 2006 , M.Tech from VTU Belagavi in the year 2015.Pursuing PhD in the area of “Consumer Privacy in Cloud “,  Life member of ISTE, Life member of Kannada Sahitya Parishad,  Nodal officer of Spoken Tutorial in association with VTU,  Coordinator for E-sikshana, KTech Innovation Hub and FOSS at SDMIT ujire,  B.E. Canara Engineering College, Mangalore M.Tech SDM College of Engineering and Technology, Dharwad.  His Subject of Interest is Cloud Technologies , IoT and Software Engineering. |
| ` | Name: Poornima  USN: 4su16cs075  Address:  6th sem CSE,SDM Educational Society (R)  Dharmastala Ujire,D.K-574240  E-mail ID: poornimaupadhyaya12@gmail.com  Contact Phone No: 7022560679 |
|  |  |
|  | Name: Rajashree Hebbar V  USN: 4su16cs086  Address:  6th sem CSE,SDM Educational Society (R)  Dharmastala Ujire,D.K-574240  E-mail ID:rajashreehebbar50@gmail.com  Contact Phone No: 8105934892 |