THE INTERNATIONAL JOURNAL OF SCIENCE & TECHNOLEDGE

A Comparative Analysis of Sorting Algorithm

Asaju, Bukola Christine
Lecturer, Department of Computer Science, Federal Polytechnic, Idah, Nigeria
Ekuma, James N.
Lecturer, Department of Computer science, Federal Polytechnic, Idah, Nigeria
Abiola, Florence Funke
Lecturer, Department of Computer science, Federal polytechnic, Idah, Nigeria

Abstract
Data structure is one of the most important tools for organizing a large data. It is seen as a method to systematically manage data in a computer that leads towards efficiently implementing different data type to make it suitable for various application (Chhjed, 2014). In addition, data structure is considered as a key and essential factor for designing algorithm with good effectiveness and efficiency. Algorithms have been developed such as Merge sort, Insertion sort, Selection Sort and Quick sort, meanwhile, several efforts have been taken to improve techniques like merge sort, bubble sort, Insertion sort, selection sort, each of them has a different mechanism to reorder elements which increase the performance and efficiency of the practical application and reduce the time complexity of each one. It is worth nothing that when various sorting algorithm are been checked there are few parameters that must be checked such as complexity and execution time. In general, the complexity of an algorithm is generally written in the form of O(n) notation, where O represent the complexity of the algorithm and the value it represents the number of element operation performed by the algorithm (Jadoom, 2013). So far, several researchers have focused on how to described and improve the algorithm and ignoring data structure, while the data structures significantly affect the performance and efficiency of an algorithm. Hence, this study analyzes the Quick sort, Merge sort, Selection sort, Insertion sort, Bucket sort, Bubble Sort, and Radix sort algorithm and their behavior on large data set using the total number of students in the School of Technology Federal Polytechnic Idah. To accomplished the major tasks, proposed methodology comprises of three phases which are Introduction of the sorting algorithm, implementation and its performance.

Keywords: *Data structure, data types, sorting algorithms, effectiveness*

1. Introduction

Sorting is nothing but storage of data in a sorted order, it can be in ascending or descending order. The term sorting comes into picture with the term searching. There are so many things in our real life that we need to search, like a particular record in database, roll numbers in merit list, a particular telephone number, any page in a book etc. Sorting arranges data in a sequence which makes searching easier. Every record which is going to be sorted will contain one key. Based on the key the record will be sorted. Data structure is a way of collecting and organizing data in such a way that we can perform operations on these data in an effective way. Data structure is about rendering data elements in terms of relationship for better organization and storage. Data in a database can be sorted and filtered to makes it easier to understand or work with. Despite the importance of data organization, sorting a list of numbers or character is one of the fundamental issues in computer science. Sorting techniques attracted a great deal of research for efficiency, practically, performance, complexity and type of data structure, therefore, data management needs to involved a certain process. As a result of sorting is an importance part of data organization.

Sorting arranges data alphabetically or numerically in ascending or descending order. For example, in the Student's information Database are often referenced by Stud ID. Sorting the record by Stud ID makes finding a particular record about a student easier when viewing the table. Sorting the record by the operator field makes viewing the record handled by each operator easier. Algorithm can also be helpful to group the data on the basis of a certain requirement such as code or in the classification of data into certain groups. Thus, sorting a list of input of items list dedicated in order to produce solution for a desired result. Several researchers have focused on how to improve the use of algorithm and ignore data structures, while the data structure significantly affect the efficiency of the algorithm. Sorting is one of the basic functions of management needs to involved a certain algorithm but did not focus on the types of data structure used in them. Hence, this study analyzes the Quick sort, Merge sort, Selection sort, Insertion sort, Bucket sort, Bubble Sort, and Radix sort algorithm to evaluate their performance measure using time complexity, execution time and the size of data set used.

1.1. Statement of Problem

The issues currently with this study which need to be address are:

- Implementation of the sorting algorithms
- Analyzing and organizing the set of data collected from the school of technology
- Knowing the difference in the performance measure of each algorithm

1.2. Objectives of the Study

Based on the search background the three objectives of the research are:

- To organize, analyze a set of data obtain from the school of technology Federal Polytechnic Idah
- To implement sorting Algorithms
- To know the performance measure of the algorithms based on its complexity and data set.

1.3. Research Motivation

Despite the importance of data organization, sorting a list of input numbers is one of the important issues in computing, sorting techniques attracted a great deal of studies, for efficiency, performance, complexity and type of data structure (Gurran & Jaideep, 2011). Therefore, data management needs to involve a certain sorting process. As a result, sorting is an important part in data organization. Lots of researchers are attentive in writing the sorting algorithms but do not focus on the differences in the performance of the algorithms. Thus, the aim of this study is to use the algorithms under consideration to sort a set of data and know their performance measure using time complexity and the size of data set used.

1.4. Related Works

The goal of this chapter is to create the significance of the general field of study. First, the data structure with its main activities and detail description of the sorting techniques. It also discusses some evaluation measure like execution time, algorithm complexity and the summary of the chapter.

1.5. Data Structure

A data structure is a specialized format for organizing and sorting data. General data structure types include the array, the file, the record, the table, the tree and so on. Any data structure is designed to organize data to suit a specific purpose so that it can be accessed and worked with in appropriate ways data structure is a systematic organization of information and data to enable it to be used effectively and efficiently especially of operations and considered a way to manage large amounts of data such as the index of intent and large cooperate database (Chhaje, 2015).
An effective and efficient algorithm is required when designing highly efficient data structures. In the real, of software design, there are several studies that have attested the importance of data structure. (Yan, 2011).

Data structure are generally based in the ability of a computer to fetch and store data at any place in its memory, specified by a pointer with a bit presenting a memory address. Thus, the data structures are based on computing the address of data items with arithmetic operation (Okun, 2014).

1.5.1. Array Data Structure

An array is a data structure use in arranging a group of data element in the form of row and column but has a singular identifier but different indicators. Index are usually used to access data element in an array. Arrays are useful in supplying and orderly structure which allow users to store large volume of data efficiently. For example, the content of any array may be changed during run time whereas the internal structure and the number of the element are fixed and stable (Yang, 2015).

An array could be fixed array because they are not change structurally after are created. This means that the user cannot add or delete to its memory location (making the array having less or more cells), but can modify the data it contains because it is not change structurally. Andrew 2010 illustrated that there are three ways in which the elements of an array can be indexed.

- Zero based index: It is the first element of the array which is indexed by subscript 0.
- One based indexing: It is the first element of the array which is indexed by the subscript.
- N-based indexing: It is the base index of an array which can be freely chosen therefore, arrays are important structure in the computer studies, because they can store large volume of data in a proper manner while comparing with the list structure which are hard to keep track and do not have indexing capabilities that makes it weaker in terms of structure. (Yang, 2015).

1.5.2. Advantages of Using Array

Data structure has weakness and strengths. List below are the advantages of array as mention by Andrew (2012).

- Array allows for faster access to any time using the index
- Arrays can be used to represent multiple data items of same type by using single name but different indexes
- Arrays are useful when working with sequence of the same type of data
- Array are simple to understand and use

1.5.3. Disadvantages of Using Array

Though arrays are very useful data structure, however, it has some bottleneck which are:

- The elements of arrays are so tired in consecutive memory locations, therefore operations like, add, delete, swap can be very difficult and time consuming
- Array items are stored memory location, sometimes there may not be enough memory location available in the neighborhood.

1.6. Execution Time

Execution time is the time taken to hold process during the running of a program. The speed f the implementation of any program depends on the complexity of a techniques or algorithm. If the complexity is low, then implementation is faster whereas, when the complexity is high the implementation is low. (Puschner, 2004). Believe that time is one of the important computer resources for two reason. The time spent for the solution and the time spent for program implementation and for providing services. Therefore, he argues that execution is the line needed by a program to process a given input.

1.6.1. Time Complexity

The time complexity of an algorithm qualifies the amount of time taken by an algorithm to
run as a function with the length of a string representing the input. The time complexity is commonly expressed using Big(O) notation, which excludes coefficients and lower order terms. When expressed this way, the time complexity is said to be described asymptotically as the input size goes to infinity. The time complexity is commonly estimated by counting the number of elementary operations performed by the algorithm, where an elementary operation takes a fixed amount of operation performed by the algorithm differs by a constant factor (Michael, 2007).

1.7. Worst Case Analysis

The Worst-case analysis anticipates the greatest amount of running time that an algorithm needed to solve a problem for any input size n. The worst case running of an algorithm gives us a bound on the computational complexity and also guarantee that the performance of an algorithm will not get worse (Marton, 2001).

1.7.1. Best Case

The best-case analysis expresses the least running time the algorithm needed to solve a problem for any input of size n. The running time of an algorithm gives a lower bund on the computational complexity. Most of the analysis of not consider that best case performance of an algorithm because it is not useful (Azrimary, 2012).

1.7.2. Average Case Analysis

Average case analysis is the average amount of running time that an algorithm needed to solve a problem for any of size n. Generally, the average running time is considered approximately as the worst-case time. However, it is useful to check the performance of an algorithm if its behavior is average over all potential sets of input data. The average case analysis is much more difficult to carry out, requirement tedious process and requires considered mathematically refinement that causes worst case analysis become more prevalent. (Anthere, 2015).

1.8. Big-O Notation

Big-O notation is used to characterized upper bound of a function that state the maximum value of resources needed by an algorithm needed by an algorithm to do the execution. According to (Black, 2006) Big-O notation has two major fields of application namely. Mathematics and computer Science. In mathematics, it is usually used to show how closely a finite series approximates a given function. In computer science, it is useful in the analysis of algorithms. There are two usage of Big-O notation which are infinite asymptotic and infinitesimal asymptotic. This singularity is only in the application not in precept, however, the formal definition for the Big-O is the same for both cases, only with different limits for the function evidence. Let f(n) be function that map positive integers to positive real numbers. Say that f(n) is O(g(n) or if (n)ϵO(n))) if there exist a real constant $c > 0$ and there be an integer constant n() ≥ 1 such that if(n)$\leq$c.g(n) for every integer n$\geq$n

1.9. Sorting

Sorting is the process of arranging the set of elements which can be sorted alphabetically, descending or ascending order, or based in a certain attribute such as index number, zip code etc. (Coxon, 2001). There are many sorting techniques that have been developed and analyzed throughout the literature. This indicates that sorting is an important area of study in computer science. Sorting numbers of items can take a substantial amount of computing resources. The efficiency of an assorting techniques is based on the number of items being processed. For small collection, a complex sorting method may not be as useful compared them with respect to their running time. Sorting algorithm is one to the most basic research area in computer science, the aim is to make data easier to be updated. Sorting is a significant process in computing programming. It can be used to sort sequence of data by an ordering procedure using a type of keyword. The sorted sequence is most helpful for later updating activities such as search, insert and delete.

1.9.1. Quick Sort

Quick sort is the fasted internal sorting algorithm among other develops algorithm. Unlike merge sort, quick sort needs less memory space for sorting array. Therefore, is vastly used in most real time application with large data sets. Quick sort uses divided and conquer approaches for solving problems. It works by portioning an array into two parts, then sorting the part independently. It finds the element called pivot which divided the array into halves in such a way that elements in the left are smaller than the pivot element and element in the right are greater than the pivot element. (Pooja, 2015).

The algorithm repeats this operation frequently for both the sub arrays. In general, the leftmost or the rightmost element is selected as a pivot. Selecting the leftmost and the rightmost element as pivot was practiced since the earlier version of quick sort. Quick sort has a fast sorting algorithm on the time complexity O(nlogn) in contracts to another developer algorithms. However, selecting the leftmost or rightmost as a pivot causes the worst case run time O(nlogn) when the array is already sorted.

1.9.2. Merge Sort

The merge sort 'has a complexity of O(nlogn). The O(nlogn)worst case upper bound on merge stems from the fact that merge is O(n). The application of the merge sort produces a stable sort, which means that the applied preserves the input of equal elements in the sorted output. Merge sort was invented by Von. Neumann and Morgenstern (1945) work by divided and conquer method and it is based on the division of the array into two halves at each stage and then goes to a compare stage which finally merges these parts into the single array. This is also a comparism-base sorting algorithm such as Bubble sort, selection sort and insertion sort. In this method, the array is divided into two halves, then recursively sort these two parts and merge them into a single array. When working with small array, merge sort is not a good choice as it requires an additional temporary array to store the merge element with O(n) space.

1.9.3. Selection Sort

Selection sort is notable for its programming simplicity and in certain situation can over perform other sorting algorithms. It works by finding the smallest or highest element form the unsorted list and swap with the first element in the sorted list and then find the next smallest element form the unsorted list then swap with the second elements in the sorted list. The algorithm continues this operation until the list has been sorted. (Bharadwaj, 2014). Selection sort requires a constant amount of memory space with only the data swaps within the allocated spaces. However, like some other simple sorting methods, selection sort is also sufficient for large datasets or arrays (Mishra, 2013). Selection sort has O(n^2) time complexity, making it inefficient on large list and performs worse than the similar insertion sort but better than bubble sort.

1.9.4. Bottle Neck of Selection Sort

- It's poor efficiency when dealing with a huge list of items
- The selection sort requires n-squared number of steps for sorting elements.
- Quick sort is much more efficient than selection sort.

1.9.5. Bubble Sort

Bubble sort is simple and the slowest algorithm which works by comparing elements in the list progress elements and swapping them if there are in undesirable order. The algorithm continues this operation until it makes a pass right through the list without swapping any elements, which shows that the list is sorted. This process takes a lot of time and especially slow when the algorithm works with a large data size. Therefore, it is considered to be the most inefficient sorting algorithm with large datasets (Astrachan, 2004).
According to (Barcher, 2000). Bubble sort has a complexity of O(n^2), where n indicate the numbers of elements to be sorted. However, other simple sorting algorithm such as insertion sort have the same worst-case complexity of O(n^2) the efficiency of Bubble sort is relatively lesser than other algorithms.

1.9.6. Bottle Neck of Bubble Sort

- It is very slow and runs in O(n^2) time in worst as well as average case
- There are many sorting algorithms that run in a linear time i.e. O(n) and some algorithm in O(nlogn)
- The only advantages of bubble sort are that is easy to understand
- The loop continues to run even if the array is sorted if the code is not optimized
- The bubble sort is mostly suitable for academic teaching but not for real life application.

1.9.7. Insertion Sort

Insertion sort is a simple and efficient sorting algorithm, beneficial for small size data. I work by inserting each element into the suitable position is the final sorted list. For each insertion, it takes one element and find the suitable position in the sorted list by comparing with contiguous and insert it in that position (Ching, 2008). This process is iterative until the list is sorted in the desired order. Unlike other sorting algorithms, insertion sort goes through the array list only once, requiring only a constant amount of memory space as that data is sorted within the array itself by dividing itself into two-sub array, one for sorted and one for unsorted. However, it becomes more ineffective for a greater size of input data when compared to other algorithms.

1.9.8. Bottle Neck of Insertion Sort

- It does not perform as well as other, better sorting algorithms
- With n-squared steps required for every n element to be sorted, the insertion sort does not do well in huge list
- The insertion sort is particularly useful only when sorting a list of few items

1.9.9. Heapsort

Heap is a comparison-based sorting algorithm. Heap sort can be thought of as an improved selection sort: like that algorithm, it divides its input into a sorted and an unsorted region, and it iteratively shrinks the unsorted region by extracting the largest element and moving that to the sorted region. The improvement consists of the use of a heap data structure rather than a linear-time search to find the maximum. Although somewhat slower in practice on most machines than a well-implemented quicksort, it has the advantage of a more favorable worst-case O(n log n) runtime. Heapsort is an in-place algorithm, but it is not a stable sort. Heapsort was invented by J. W. J. Williams in 1964. This was also the birth of the heap, presented already by Williams as a useful data structure in its own right. In the same year, R. W. Floyd published an improved version that could sort an array in-place, continuing his earlier research into the treesort algorithm. The heapsort algorithm involves preparing the list by first turning it into a max heap. The algorithm then repeatedly swaps the first value of the list with the last value, decreasing the range of values considered in the heap operation by one, and sifting the new first value into its position in the heap. This repeats until the range of considered values is one value in length.

1.10. The Steps Are

- Call the buildMaxHeap() function on the list. Also referred to as heapify(), this builds a heap from a list in O(n) operations.
- Swap the first element of the list with the final element. Decrease the considered range of the list by one.
- Call the siftDown() function on the list to sift the new first element to its appropriate index in the heap.
- Go to step (2) unless the considered range of the list is one element.

The buildMaxHeap() operation is run once, and is O(n) in performance. The siftDown() function is O(log n), and is called n times. Therefore, the performance of this algorithm is O(n + n log n) = O(n log n).

1.10.1. Bucket Sort

Bucket sort, or bin sort, is a sorting algorithm that works by distributing the elements of an array into a number of buckets. Each bucket is then sorted individually, either using a different sorting algorithm, or by recursively applying the bucket sorting algorithm. It is a distribution sort, a generalization of pigeonhole sort, and is a cousin of radix sort in the most-to-least significant digit flavor. Bucket sort can be implemented with comparisons and therefore can also be considered a comparison sort algorithm. The computational complexity estimates involve the number of buckets.

1.10.2. Bucket Sort Works As Follows

- Set up an array of initially empty "buckets".
- Scatter: Go over the original array, putting each object in its bucket.
- Sort each non-empty bucket.
- Gather: Visit the buckets in order and put all elements back into the original array.

1.10.3. Comb Sort

Comb sort is a relatively simple sorting algorithm originally designed by Włodzimierz Dobosiewicz in 1980. Later it was rediscovered by Stephen Lacey and Richard Box in 1991. Comb sort improves on bubble sort. when any two elements are compared, they always have a gap (distance from each other) of 1. The basic idea of comb sort is that the gap can be much more than 1. The inner loop of bubble sort, which does the actual swap, is modified such that gap between swapped elements goes down (for each iteration of outer loop)

1.10.4. Radix Sort

Radix sort is a non-comparative integer sorting algorithm that sorts data with integer keys by grouping keys by the individual digits which share the same significant position and value. A positional notation is required, but because integers can represent strings of characters (e.g., names or dates) and specially formatted floating point numbers, radix sort is not limited to integers. Radix sort dates back as far as 1887 to the work of Herman Hollerith on tabulating machines. Most digital computers internally represent all of their data as electronic representations of binary numbers, so processing the digits of integer representations by groups of binary digit representations is most convenient. Two classifications of radix sorts are least significant digit (LSD) radix sorts and most significant digit (MSD) radix sorts. LSD radix sorts process the integer representations starting from the least digit and move towards the most significant digit. MSD radix sorts work the other way around. LSD radix sorts typically use the following sorting order: short keys come before longer keys, and keys of the same length are sorted lexicographically. This coincides with the normal order of integer representations, such as the sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11. MSD radix sorts use lexicographic order, which is suitable for sorting strings, such as words, or fixed-length integer representations. A sequence such as "b, c, d, e, f, g, h, i, j, ba" would be lexicographically sorted as "b, ba, c, d, e, f, g, h, i, j". If lexicographic ordering is used to sort variable-length integer representations, then the representations of the numbers from 1 to 10 would be output as 1, 10, 2, 3, 4, 5, 6, 7, 8, 9, as if the shorter keys were left-justified and padded on the right with blank characters to make the shorter keys as long as the longest key for the purpose of determining sorted order.

1.10.5. Shell Sort

Shellsort, also known as Shell sort or Shell's method, is an in-place comparison sort. It can be seen as either a generalization of sorting by exchange (bubble sort) or sorting by insertion (insertion sort). The method starts by sorting pairs of elements far apart from each other, then progressively reducing the gap between elements to be compared. Starting with far apart elements, it can move some out-of-place elements into position faster than a simple nearest neighbor exchange. Donald Shell published the first version of this sort in 1959. The running time of Shell sort is heavily dependent on the gap sequence it uses. For many practical variants, determining their time complexity remains an open problem.

According to (Okuda, N 2012) Shell sort is a generalization of insertion sort that allows the exchange of items that are far apart. The idea is to arrange the list of elements so that, starting anywhere, considering every hth element gives a sorted list. Such a list is said to be h-sorted. Equivalently, it can be thought of as h interleaved lists, each individually sorted.[6] Beginning with large values of h, this rearrangement allows elements to move long distances in the original list, reducing large amounts of disorder quickly, and leaving less work for smaller h-sort steps to do. If the list is then k-sorted for some smaller integer k, then the list remains h-sorted. Following this idea for a decreasing sequence of h values ending in 1 is guaranteed to leave a sorted list in the end.

1.10.6. Related Work

Research has published a considerable amount of work on sorting techniques. While looking into large or growing literature, it appeared that sorting techniques has been proven to be successful for data structures. Thus, the data structures have an impact on the efficiency of this sorting techniques. (Ching, 2014) discussed and reviewed the performance of sorting algorithms where comparison of the algorithms was base on the time of implementation. It was found that for small data, other techniques like insertion, Merge, Selection, Heap etc. perform well but for large data quick sort techniques is considered fast. (Pooja, 2014) examine several sorting algorithms and discussed the performance analysis of these sorting based on their complexity while testing them with the list of data structure, it was found that quick sort has the highest complexity and faster in large list. In the work of (Chhjed, et al, 2015), four algorithms which are insertion sort, Heap sort, Quick sort, and Bubble sort were compared, although all these techniques are of O(n^2) complexity, it was found that they produced different terms of execution time. They design a new sorting algorithm names index sort, to check the performance of these four algorithms then compare with other four sorting algorithms based on their run time and founded that the Index sort is faster than the other sorting algorithms.

2. Methodology

The proposed framework involved three phases namely, Implementation of the sorting algorithms, calculating the complexity and comparative analysis.

The first is implementation the 10 sorting algorithms, which are Quick sort, Insertion sort, Merge sort, Selection sort, Bubble sort, Combo sort, Heap sort, Bucket sort, Radix sort and Shell sort.

The second phase calculate the complexity of these 10-sorting algorithm using Big O notation.

The third phase is comparing and analyzing these sorting algorithms with performance measure based on runtime and size of data set.

2.1. System Design

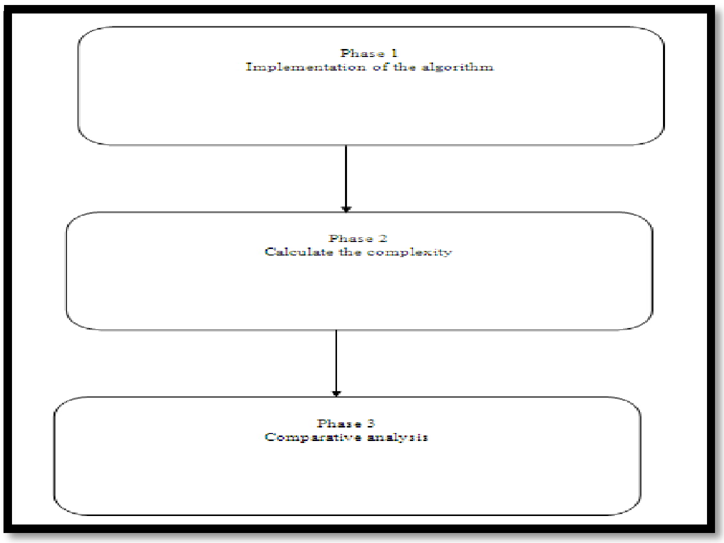

Figure 1: Three Phases of the Work

2.2. Implementing the Sorting Algorithms

This phase encompasses the implementation of the ten (10) sorting algorithms. The ten (10) algorithms are Quick sort, Insertion sort, Merge sort, Selection sort, Bubble sort, Combo sort, Heap sort, Bucket sort, Radix sort and Shell sort using java programming language.

2.3. Calculating the Complexity of Sorting Algorithms

In this phase, the sorting algorithms are tested sung the collected data set. Then the program calculates the runtime complexity of each algorithm which are Quick sort, Insertion sort, Merge sort, Selection sort, Bubble sort, Combo sort, Heap sort, Bucket sort, Radix sort and Shell sort using Big-O notation concept. This concept is used to measure the complexity of algorithm and it is useful in the analysis of algorithm for efficiency.

2.4. Comparative Analysis

In this phase, the sorting algorithms are tested again using the data set, However, this time, they are tested in terms of their efficiency based on the complexity, execution time per nanoseconds and size of input dat. The performance measure is analyzed on two different behavior which are $O(n^2)$ and O(nlogn) in order to perform the required analysis.

2.5. Dataset Gathering / Description

The input date is set of integers collected from the school of technology Federal Polytechnic Idah on the total number of students in the school which comprises seven (7) main department which are:

- Computer Science has all level NDI, NDII, HNDI, HNDII making the sum total of 483
- Library Information Science (LIS) has NDI and NDII making the sum total of 185
- Leisure and Tourism all level NDI, NDII, HNDI, HNDII making the sum total of 157 students
- Hospitality Management and Technology all level NDI, NDII, HNDI, HNDII making the sum total of 306 students
- Food Science and Technology all level NDI, NDII, HNDI, HNDII making the sum total of 441
- Science Laboratory and Technology all level NDI, NDII, HNDI, HNDII making the sum total of 788 students
- Mathematics and Statistics all level NDI, NDII, HNDI, HNDII making the sum total of 166
- From the analysis above the total number of students in school of technology Federal Polytehcni Idah is a sum total of two thousand, six hundred and twenty-six (2626).

Department	Level I ND	Level II HND	Sum Total
Computer Science	351	132	483
Mathematics/Stat	194	72	266
SLT	576	212	788
LTM	85	72	157
LIS	185		185
FST	305	136	441
HMT	183	123	306
Grand Total		2626	

Table 1: Dataset Collection

2.6. Data Set Groups

A data set is a random integer number with several files of integers, selected at random to be used to test the ten (10) sorting methods. The files are of different sizes, ranging between 001 and 2,626 Then the number used for these data set are divided to ten (10) groups of different intervals as shown in Table

Group	Size
Group 1	001-163
Group 2	264-525
Group 3	326-788
Group 4	789-1051
Group 5	1052 – 1214
Group 6	1315 – 1577
Group 7	1578 – 1840
Group 8	1841 – 2103
Group 9	2104 – 2366
Group 10	2367 – 2626

Table 2: Dataset

2.7. Implementation and Analysis

One of the main objectives of this study is to compare the sorting algorithm from a practical perspective. For the reason they are factors that prompt the interest of the study, which are explained in the following section.

2.8. Overall Running Time

They are two major reasons to calculate the running time of an algorithm, the fact is to understand how the time grows as a function with respect to its input parameters and the latter is to compare two or more algorithms for the same problem. The overall time complexity of an algorithm state that the amount of time taken by an algorithm regarding to the amount of memory access performed, the number of comparisons between integers, and the number of times some inner loop is executed.

2.9. Implementation of Sorting Techniques

All the sorting algorithm under study namely, Bubble, Selection, Insertion, Bucket, Quick, Merge and Radix sort are implemented in Java programming language based on the array of test data collected from the school of technology, Federal Polytechnic Idah. All the sorting algorithm were tested from the input of the test data of length 001 – 2626. All the sorting algorithms were executed on machine with operating system equipped with AMD EI-1200 APU with Radeon (tm) HD graphics 1.40 GHz and installed memory RAM 2.00GB. The CPU time was taken in per nanosecond. The executed test data a set to the static data structure were illustrated in the following subsection.

2.10. Number of Operation Performed

In general, there are two different types of operation performed in most of the sorting algorithms, such as comparism operation and assignment operation. For instance, to sort an array of integer the algorithm needs to compare each element in that array to output appropriate position in the sorted array, this method is called as comparison operation, it measures the overall running time of an algorithm, due to the fact that if the input size grows then the number of comparisons will also increase. In assignment operation, each time when the desired position found, the algorithm needs to swap the element by using a temporary variable. This will also react the tuning time of an algorithm. However, when compared with the assignment of operation is ounce to an assignment operation in the running time is comparatively less.

2.11. Memory Consumption

The memory required for an algorithm during run tune is another important concern while selecting an optimal algorithm for a given problem. When we talk about the memory consumption, it generally means main memory if RAM. Memory or space complexities state that amount of memory taken by an algorithm with regards to the amount of input of an algorithm. Certainly, each program needed some amount of memory of input algorithm, certainly each program needed some amount of memory to store the program itself and also the data required during execution. Moreover, the major consideration is the additional amount of memory utilized by the program, for the reason that some algorithm might need extra memory, according to the size of input and it is not suitable for the machine with minimum resources.

2.12. Software Profiling

Software profiling is the investigation of a program's behavior using information collected during the execution of the program. The aim of this analysis is to determine which part of a program need to improve in order to extends the efficiency. Some of the performance enhancement measures are increasing the overall speed, minimizing the memory usage, determine the need of time consuming functions and calls, avoiding the unnecessary computation, avoiding re-computation by storing the results for future use. There are various techniques used by profilers based on the different quantities of interest such as event-based, statistical instrument and simulation method.

2.13. Design and Implementation

In the previous section of this work, software profiling and its usage were discussed along with the quantities of interest that considered for this study. Based on the above analysis I have design a Java program with the ability to collect desired data during execution. The design includes the various sorting algorithms. The program is capable of creating 2626 random numbers of integers. The size of the array elements varied from 0 to 2626 and generate another data set randomly from the data set once each algorithm completed its execution with certain array size. The rand() function was used to generate random numbers and rand()

Function was used as a seed to initialized the random number generator every time the program is being executed. In general, the maximum value returned by rand() function is 2626 however, this program has the ability to generate more infinites range. The data collected during the experiment was the running time of an algorithms and the operation performed. In order to measure the running time of each sorting algorithm the clock () function was used and divided by CLK TCK to the time in nanoseconds. Data gathered string the execution of the program was later used for comparing various sorting algorithms and their efficiency against various experiments. In this case Java program, instead of passing arrays to each algorithm, we have declared arrays globally and then accessed from each algorithm,

All the experiments were conducted on the model machine described below as my test bed. AMD E1-1200 APU with Randeon(tm) HD Graphics 1.40 GHz, 2.00GB RAM, 64bit Operating system, Java Neat bean IDE, 360 GB HDD.

2.14. Algorithms and Their Implementation

In this study, most of the sorting algorithms were implement based on the standard algorithms explained in the literature. However, some algorithms were deviated from standard method to improve overall efficiency during the execution.

2.15. Test Case Scenarios

In this study, various experiment was conducted in the above-mentioned model machine to observe the behavior of each algorithm. In each experiment for each of the algorithm randomly generated array values were integers and the data collected during the experiment was the ruining time of each sorting algorithm. The size varied from 0 – 2626 for each of the sorting algorithm.

2.16. Comparison of Sorting Algorithms

In the previous section, we have discussed the design and implementation of various algorithms considered for the study along with various test experiments which are conducted for the evaluation of practical efficiency of sorting algorithms. In this chapter we have discussed the experiment results besides analysis and comparison of various sorting algorithms with the help of experiment results. According to the theoretical study, we have concluded and compared the time complexity of various algorithms in three different cases such as average case, best case and worst case along with the stability and method of working. The describe below represent the variation where n defines the number of items to be sorted, k defines the range of number in the list.

Algorithms	Best case	Worst case	Average case
Bubble Sort	O(n)	O(n^2)	O(n^2)
Bucket Sort	O(n^2)	O(n^2)	O(n^2)
Shell Sort	O(nlog) n))	O(nlog(n))^2)	O(nlog(n))^2)
Selection sort	O(n^2)	O(n^2)	O(n^2)
Insertion sort	O(n^2)	O(n^2)	O(n^2)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)
Quick Sort	O(nlogn)	O(n^2)	O(nlogn)
Heap sort	O(nlogn)	O(nlogn)	O(nlogn)
Radix sort	O(n)	O(n)	O(n)
Comb	O(n)	O(n^2)	O(n^2)

Table 3: Comparison of Sort Algorithms

2.17. Details of Experiment

In this section, the experiment result and analyzed are discussed the data being gathered during the carious experiments. Result of these experiment are described and analyzed with the table of data and observed behavior.

2.18. Test Case

Data Elem	Bubble	Bucket	Shell	Selection	Insertion	Merge	Quick	Radix	Comb	Heap
263	41033	811873	44697	79868	55688	114307	190512	359042	38835	39567
525	49894	427186	24913	31508	11724	33706	90127	55688	40301	39568
788	48361	21982	17586	34439	19784	57886	86463	203701	26379	40300
1051	24481	39568	146550	19051	16833	32973	134823	56421	27111	28577
1314	46895	41033	14655	30775	13189	32241	80601	55688	38835	23447
1577	27111	40301	14654	19051	12456	69351	81334	56421	40301	40300
1840	22713	38835	15388	18319	11724	33706	71075	98187	73274	39568
2103	29310	35171	24913	19051	10259	58619	128228	371498	39568	27112
2366	32973	38102	16120	36043	21982	32973	150943	40300	24913	39588
2626	26379	40301	14655	29310	21250	39568	124565	23447	39568	22715

Table 4: Test Case

3. Results

Each algorithm performs the sorting by generating 15 integers randomly from the maximum range of integers and then calculate the complexity by using the notation of the selected algorithm each for the best case, worst case and average case. As shown below:

- For Quick sort have:
- Best case: O(nlogn), Worst case: O(n^2), Average case: O(nlogn)
- Therefore,
- Best case is calculated as 15log15
- Worst Case calculated as: 15^2
- Average case calculated as : 15log15

- When the case is combined, the produce the total time taken by each algorithm base on their notation analysis which is measure in nanoseconds.
- In my experiment the Quick sort has 124565 nanoseconds which has happen to be the best among the 10 algorithm under consideration.

3.1. Discussion of Result

In this research work, several experiments were carried out and has obtained performance for each sorting algorithm. The study checks the performance measure of each algorithm in terms of overall running time and size of data. The results of experiments carried out shows besides, it was simple to implement and thus it would be well suited for small and large data sets the relative performance of selection sort and insertion sort differed on different type of input items. For instance, insertion was more efficient than selection sort in integer data items. When the number of data grew the improved quick sort had the least run time among the comparism based sorting techniques. On this study, the empirically observed results show that when the size of data items doubles, the running time goes up by a factor of 3 for quadratic methods and takes slightly more than 2 for linear logarithmic methods. Furthermore, the running time of each algorithm changes relatively with different type of input data, such as integers. The result shows that large integers take minimum runtime. However, the different types of input data did not affect the asymptotic order in most methods.

4. Conclusion

This paper presented the comparative analysis of different sorting algorithms with the result of practical performance. The main findings of this study are that the factor such as running time, the size of dataset and memory used are critical to efficiency of sorting algorithms.

Experimental result clearly shows that the theoretical performance behavior is relevant to the practical performance of algorithms in some the test case. In addition, the experiment result proved that algorithm's behavior will change according to the size of element to be sorted and type of input elements. Which concluded that each algorithm has its own advantages and disadvantage. However, this does not reduce the need of simple sorting algorithm in some cases due to the fact that many sophisticated simple sorting algorithms such as insertion sort and shell sort despite of its quadratic performance. As a result, there are more than two algorithms which are used to solve problem as a hybrid method. Thus, selecting efficient algorithm depends on the type of problem. This concludes that sorting algorithm is problem specific.

5. References

i. Al-Kharabshel, K.S (2013). "Review on sorting algorithms, A comparative study" International Journal of Computer Science and Security (IJCSS). Jackson Publication, United State of America.

ii. Anwar, N. F (2016). "Comparative study on sorting Techniques in Static Data Structure" Faculty of Computer Science and Information Technology

iii. Ashok, K. K (2014). "A survey, Discussion and Comparism of Sorting Algorithms" Department of Computer Science Umea University

iv. Batcher, K.E, (2003). "Sorting network and their applications" Spring joint computer comference. ACM Publishers. U.K

v. Black, P.E (2006). Disctionary of Algorithms and Data Structures. United State Standard Institute of Science and Technology. United Kingdom.

vi. Chhajed, n. (2015). "A comparative Based Analysis of four different sorting algorithms in data structure with their performance" International Journal of Advance Research in Compter Science and Software Engineering.

vii. Ching, K. (2013). Technological University, department of computer science. Hill publishing house, Houghton.

viii. Coxon, A.P.M (2001). "Reveiew of data collection, sorting and analysis" Sage blinks publiscations.

ix. David, H.K. (2011). "comparative analys on index sorting" International Jounral of Experimental Algorithms (IJEA). Bloms publications.

x. Liu, Y. (2013). "Review on Quick sort algorithm based on multi-core linux. In Mecharonic Science. Electrical Engineering and Computer (MEC). International Confeence on (IEEE).

xi. Pooja, K. (2013). Comparative Analysis & Performance of Different Sorting Algorithm in Data Strucuture. Hirani Institute of Polytehcnic, Pusad, Maharashtra India.