CONSTRASTING THE FOUR BASIC SORTING ALGORITHMS

BY

GARBA ABDULLAHI
PGD/MCS/97/98/726

A PROJECT SUBMITTED IN PARTIAL FULFILMENT OF THE DEPARTMENT OF MATHEMATICS/COMPUTER SCIENCE REQUIREMENTS FOR THE AWARD OF A POST GRADUATE DIPLOMA IN COMPUTER SCIENCE OF FEDERAL UNIVERSITY OF TECHNOLOGY, MINNA.

DECEMBER, 2001

CERTIFICATION

This is to certify that this research was carried out by Garba Abdullahi PGD/MCS/97/98/726 of the Department of Mathematics/Computer Science is fully adequate in scope and quality for the award of the Post Graduate Diploma in computer Science of Federal University of Technology, Minna.

Mallam Isah Audu
Project Supervisor

Date

Dr. N. I. Akinwande
Head of Department

Date

External Examiner

Date

ACKNOWLEDGEMENT

My gratitude is to Almighty Allah the most beneficent and the most merciful for keeping me alive and granting me the good health to undertake this task. I will also like to express my gratitude to all the lecturers of the department of Mathematics and Computer, especially the following: Professor K. R. Adeboye who urged me to carry on with the programme, Malam Isah Audu, a friend turned lecturer who also served as a source of inspiration to me and Malam Danladi Hakimi (Sarki Yakin Egwa).

My gratitude also goes to Dr. Aiyesimi, Prince R. O. Badmus. May Allah reward all your efforts.

DEDICATION

This is dedicated to my numerous friends, well wishers and colleagues.

ABSTRACT

The project studied four sorting techniques. Sorting is the process of rearranging a given set of objects in a specific physical order. The first case is a list of integer of a certain size generated and passed to each of the four sorting techniques. Four sorting techniques were considered. And in each case the time taken is recorded in to a text file. This process is repeated for a list of five different sizes.

In the second case, an array of strings will be stored in a text file (input file), a certain size of this strings will be revolved and passed to each of the four sorting techniques. Again in each case the time taken is recorded on to a text file (out file). This process if repeated for five different sizes.

Then for the two cases, a graph will be plotted of the measured time against size. Then the result from the graph will be interpreted and some conclusion drawn.

TABLE OF CONTENTS

CHAPTER ONE

GENERAL INTRODUCTION

1.1 INTRODUCTION:

This unit is an insight to what the search is all about, and discuses in details the objectives of the research work, its scope and limitation.

This project work with the title "comparism of some sorting algorithms" is a great effort in computing. It is an analysis of various sorting techniques whose result will reveal the performance of the various sorting techniques; from which a deduction will be made on the performance to determine which of the methods is best for a given set of data. Sorting is generally understood to be the process of rearranging a given set of objects in a specific physical order.

A good deal of effort in computing is related to putting data available to users in some particular order. For example, the organization of a telephone directory, entries are ordered in strict alphabetical order by name. To find a telephone number, all you need is the name of the person to which the call will be made, then using the first few letters of the surname and locating the appropriate section of the directory, you can then search several pages until a match is found for the surname. The address and telephone number of the person will be listed against the surname.

Another example of information origination is that of a bus timetable from a control station. The destinations of buses are ordered alphabetically by town or village. Other examples includes, mailing label which was printed in a zip code order and delinquent accounts are often in order according to the length of time the account has been delinquent. Objects are sorted in table of contents, in libraries, in dictionaries and almost everywhere that information stored has to be searched and retrieved.

Ordering of data has an important impact on searching. Data that are not ordered must normally be searched using a sequential scan of all the data. Ordered data lend themselves to simple search techniques.

1.2. SIGNIFICANCE OF THE STUDY

The purpose of sorting is to facilitate the later search for members of the sorted set. It allows an orderly presentation of information when producing reports. It simplify modifications, insertions, amendment and deletion of information without destroying the key order of the remaining information.

Our primary interest in sorting is devoted to the more fundamental techniques used in constructions of algorithms. In particular, sorting is an ideal, subject to demonstrate a great deal of diversity of algorithms, all having the same purpose, many of them being optimal in one sense, and most of them having advantages over the others. It is therefore an ideal subject to demonstrate the necessity of

performance analysis of algorithms. It is well suited for showing how a very significant gain in performance may be obtained by the development of sophisticated algorithms when obvious methods are readily available.

Analysis of Algorithms is of great significance to new programmers in the sense that it enables them understand an efficient way of coding algorithm.

Performance of an algorithm is important for various representation of the list data structure. The analysis reveals the method that is suitable for the user when ordering data, depending on the data type and the size of the data to be sorted.

We need to obtain estimates or bounds on the runtime which the algorithm will need to successfully process a particular input. Computer time is a relatively scare and expensive resource which is often sought by the users. It is to everyone advantage to avoid runs that are aborted because of an insufficient time allocated on a job card. A good analysis is capable of finding bottlenecks in our programs.

Many programmers will find this analysis useful in application software development, which require sorting of items. It minimizes the total cost of soft ware development by minimizing the cost o f running the programme.

Finally, one will like to have some quantities standard for comparing more than one algorithm which claims to solve the same problems. The weaker algorithm should be improved or discarded. It is desirable to have a mechanism for filtering out the efficient algorithm and replacing those that have been rendered obsolete.

1.3 ANALYSIS OF RESEARCH PROBLEM

Many different sorting algorithms have been invented, each method has its own disadvantage(s) and advantage(s) over the others, which makes them to outperforms the others on some configurations of data and hard ward. The user of an algorithm may find it difficult to select a suitable algorithm for the application at hand.

It is in due cause that this analysis is undertaking to compare some sort algorithms. Performance is determined by the number of movies, the number of compares, the complexity of the inner loop, the size and nature of the sort item. The initial order of the elements being sorted. It is impossible to consider the interaction of all these variables, but some rules of thumb can be applied.

1.4 RESEARCH QUESTIONS

Below are the questions, which this project will attempt to answer.

I. Is there any sorting algorithm which is the best when the value stored in each component of an array is a whole number?

II. Is there any sorting algorithm which is the best when the value stored in each component of an array is a string?

III. Will the size of data affect the time it takes to sort a list of a given data type?

IV. Is there any sorting algorithm among the ones considered which is optimal? That is, which is so good given any date type and size?

1.5 OBJECTIVES OF THE STUDY

The objectives of the project are as follows:

i. To compare four different sorting algorithms, and deduce from the result obtain the best method for sorting an array list of string.

ii. To identify the best method for sorting an array of whole numbers.

iii. To find whether or not the size of elements to be sorted has any effect o the performance of an algorithm.

iv. To identify an optimal sorting method (if any) which is good for any given type of data type and size.

1.6 THE SCOPE OF THE STUDY

Below is the area covered by the project internal sorting is used to sort the items that are use as input. Sets of random whole numbers will be generated by the computer and these numbers will be sorted using the various sorting methods selected. Also, an array of strings (for examples UDUS students' admission number) will be sorted.

The research will be sorted. Sorting methods which are:-

Insertion sort, quick sort, bubble sort and merge sort. That is, it involves both simple and advance sort techniques.

The above algorithms will be studied in details, interprets results, drawn. Some conclusions and make recommendations.

This research is limited to only four sorting methods out of the several methods of sorting devised so far.

The sorting algorithms are implemented in one programming language alone (turbo pascal). It is restricted to sorting array of strings and whole numbers alone. Other data types are not taking into consideration.

Some of the limitations above are due to:-

(1) Insufficient time:- The time is very limited. There is no enough time to undergo the research for this analysis properly if the scope is wider than above.

(2) Lack of some language compliers is one of the problems facing computer centre in some Nigeria Higher Institutions, Sokoto State Polytechnic

inclusive that is why the algorithm used for this analysis are coded (implemented) in Pascal language alone.

1.7 DEFINITION OF TERMS

Random numbers:- Random numbers are unbiased numbers. In random numbers there can be no digit or number which is found more or less frequently than the others.

The values must be non predictive. A zero cannot foretell the appearance of another zero or some other particular value. This also applies to the size of the values. The production of a small or large value should not give any clue about the size of the subsequent values.

Complexity:- Complexity of an algorithm is the timing function of the algorithm and the order of that algorithm.

Run time:- Compilers are translators, it is a program which translates a program codes in high level language, known as the source code into an equivalent program in the machine code of the computer, known as the object code.

Compilers:- Compilers are translators, it is a program which translates a program code in high level language, known as the source code into an equivalent program in the machine code of the Computer, known as the object code.

CHAPTER TWO

LITERATURE REVIEW

2.1 INTRODUCTION

This chapter reviews the literature of related subject matter of the project.

The chapter discusses issues such as the review of algorithm and its properties and the review of sorting.

2.2 REVIEW OF ALGORITHM

Computer is an electro-mechanical machine, which takes in data as input. Process it and produce output at tremendous speed. It is also called electronic data processing (EDP) machine, which handle data, make arithmetic calculations, analyze data and make logical decision.

Any person who used digital computer to solve problems has some intuitive idea of the meaning of the word "Algorithm". Because a computer is nothing more than an electro-mechanical machine, which cannot do anything on his own without being instructed on what to do, how to do it, where and when to do it. Although there are some cases when the computer is instructed on what to do alone, and the computer will perform the task, as in the case of declarative programming language(s).

Computer executes an action based on the instruction/command given to it by the programmer or user. The computer does only what it is asked to do. Garbage in, Garbage out (GIGO).

For a computer to solve a given problem, it has to follow some steps, this lead to the concept of an algorithm. Algorithms can be said to be the central concept of computer science.

The work ALGORITHM was derived from the name Al-khowarizmu who authored an Algebra text in the 9^{th} century. However, more precisely, the word was a refashioning ALGORISM which was used for several years to refer to arithmetic procedures like long division and finding square roots.

The 'S' was later interchanged with 'th' by association with arithmetic and logarithm, and the most famous algorithm historically dated from the time of the Greeks, that is, the Euclid's algorithm for calculating the greatest common division of two integers.

With the advent of computers, the term algorithm has got a new lease, since every computer program is an algorithm. According to Adhert, O. J. et al (1968); Algorithm is defined as:-

"A term derived from the word algorithm which meant the art of computing with Arabic numerals. The term algorithm is now used to denote any method of computation consisting of comparatively small number of steps or any methods of computation, whether algebraic or numerical.

From the definition above, we can say that algorithm initially meant computing with Arabic numerals, but it is now an algebraic or numerical method of computing with comparatively small number of steps.

Furthermore, Adhert, O. J. et al (1968) says that "An algorithm is a detailed logical procedure that is, it is logical procedure by which a particular problem can be solve".

Algorithm is defined by George, O. A. et al (1987) as"-

"A precise formulation of a method of doing something. In computers, an algorithm is usually a collection of procedural steps or instructions organized and designed so that computer process results as the solution of a specific problem" it is an actual way of designing a collection of steps to follow in order to solve a specific problem.

2.3 PROPERTIES OF ALGORITHMS

Algorithms are characterized by several properties, some of these properties are:-

(1) **PRECISION:-** Algorismic steps should be void of vagueness.

(2) **EFFECTIVENESS OR EFFICIENCY:-** Execution of impossible ask must be avoided in an algorithm.

(3) **FINITENESS:-** There must be an exact number of instructions in an algorithm.

(4) **TERMINATION:-** There should be a stopping criterion to terminate an algorithm, especially in a case of an instruction with repeated execution.

(5) **OUTPUT:-** An algorithm should provide an output of implementation.

2.4 REVIEW OF SORTING

Having understood what an algorithm is generally, that is, what the concepts is all about, the next step for us now to review sorting.

According to Tremblay, J. P. and Sorenson, P. G. (1967); define sorting as:- "Sorting is the operation of arranging records of a table into some sequential order according to an ordering criterion. The sort is perform according to the key value of each record".

A sorting method is called stable if the relative order of items with equal keys remains unchanged by the sorting process. Stability of sorting is often desirable if items are already ordered (sorted) according to some secondary keys, i.e. properties not reflected by the primary key itself.

There are two important and largely disjoint categories related to sorting data, there is the internal and external sorting. Internal sorting algorithms are often classified according to the amount of work required to sort a sequence of elements.

The amount of work refers to the two basic operations of sorting. Comparing two elements and moving an element from one place to another. Internal sorting involves the storage of all the data to be sorted in the main memory.

In external sorting algorithm, the data is stored in an external secondary medium, such as tape or disk, when the amount of data is too large to be stored and sorted in the main memory, and successive parts of the data are stored in the main memory. In computer, arrays are stored in the fast high-speed random access "internal" storage, array sorting is considered to be internal sorting. Because the predominant requirement that has to be made for sorting methods on arrays is an economical use of the availability store, this implies that the permutation of items which brings the items into order has to be performed "in

situ". Thus, methods which transport items from an array A to a result in array B are of no interest.

Sorting methods which sorts' items "in situ" can be classified into three principal categories to their underlying methods:

(1) Sorting by insertion

(2) Sorting by selection

(3) Sorting by exchange

From the various definitions of algorithms and sorting above, sorting algorithms can be explain as the detailed logical procedure which represents the operation of arranging the records of a table into some sequential order according to an ordering criterion. It can also be explain as the collection of a procedural steps or instructions organized and designed so that computer can rearrange elements in a specific physical order.

List are generally sorted on the value of a particular field. The sort process may involve moving entire elements or the elements of an index list or not moving elements at all. The basic mechanisms, however, are not affected by how much of each element participates in the sorting process/operation.

Below is some of the sorting algorithms definition. That is, the definition of the algorithms used for this analysis.

1. **Insertion sort:**

According to Stubbs and Webre (1987).

"Insertion sort is a sorting method in which the basic operation is the insertion of a single element into a sequence of sorted elements, so that the resulting sequence is still sorted".

2. **Merge sort:**

Stubbs and Webre (1987) state that:

"Merge sort is a sorting method in which two sub list, each already sorted are merged together to form one aggregate list that is also sorted"

3. **Quick sort**

Stubbs and Webre explain Quick sort as:- "A method which consist of a series of steps, each of which take a list of elements to be sorted as input. The output from each step is a rearrangement of the elements so that one elements is in it's sorted positive and there are two sublist that remain to be sorted. One less than and the other greater than the sorted element".

4. **Bubble Sort**

Another well-known sorting methods is the bubble sort, according to Jean Tremblay, J. P. and Sorenson P. G. state that:- "in bubble sort items are interchanged immediately upon discovering that they are out of order".

CHAPTER THREE

METHODOLOGY

3.1 DATA TYPE

The test data used in this comparison are of two types:

I Numeric

II Strings

(1) **NUMERIC DATA TYPE:-** under this category numbers are used as data, this is further divided into:-

1. Integer
2. Real

1. **Integer**:- Integer whole numbers will be generated using random number generator. All the numbers generated will be use as test data in this comparism. The using generated numbers will consist of different values. The numbers are generated as follows:

```
Procedure Generation (Var Table: data; Var N: integer);
Var 1, Numb, seed: integer
Procedure Randomize (var seed: integers);
Var hr, min, sec, sec100:word;
Begin
Gettime (hr, min, sec, sec100);
Seed: = hr*360 – min * 60 + sec + sec100:
```

```
End;
Function Random (Var seed: integer): Real;
Const M = Maxint; A = 2743; C = 5923;
Begin
Seed: = (seed * A + C) MOD M;
If seed < 0 then seed: = seed + m;
Random: = seed /m;
End;
Begin
Randomize (seed);
For i: = 1 to N do
If (i > 1) and (table [i + 1] <> table [i] then
Tablle [ i ]: = Trunc (Random (seed) * 10000);
End;
```

THE PROCEDURE GENERATION: this generates the random numbers it calls procedure randomize.

PROCEDURE RANDOMIZE: this initializes the random number generator, it calls an in built function gettime use in finding the seed which the function random will use.

FUNCTION RANDOM: returns a random number Y of type integer in the range 0 <=Y <Argument.

2. **Real:-** the time taken to sort the elements is of type real, it is use in this project to analyze the performance of the setting methods. An in built function gettime (argument) is use to get the executing time. For example:

Gettime (hr, min, sec, sec100);

(ii) **STRINGS:** A string is a finite sequence of characters; in this project strings will be sorted using different sorting methods. Examples of string that will be use are student's Admission number in UDUS. In particular, admission number of students from the department of Mathematics.

3.2 SOURCE OF DATA

The data use for the research work are the random numbers generated and the students admission numbers obtained from the registration forms of course in the Mathematics Department collected directly from Mathematics Department Exams Officer. That is the data is from random number generating function and the department Exam Officer. Therefore, the source data is a secondary source.

3.3 SELECTION OF DATA SIZE

Different sizes of the data are used in sorting array of integer. Also, when sorting array of strings, the elements that were sorted (admission number) were of different sizes.

3.4 METHOD OF DATA COLLECTION

Random number generating function is used to generate the data, and these sets of numbers are passed to each of the sorting algorithms that are considered in this project. The size of the number generated, the name of the sorting algorithms and the time taken to sort this numbers are recorded on to a text file (output file). The program consists of a main menu which comprises six options one to six. Representing generate list, insertion sort, bubble sort, merge sort, quick sort and display the result.

If the user select one, a message will appear on the screen as "ENTER THE NUMBER OF ELEMENTS TO GENERATE". If the user type the desired number list of random number will be generated. And then the system will take you to the main menu again. You then select 2, 3, 4, 5, to sort the number generated using the four sorting methods respectively. In each case, the time it takes to sort the list by a particular method is recorded in a text file against the name of the method. The system will take you back to the main menu again, if you select, you are generating another list of different sizes.

If you type the number, the system will take you back to the main menu, then by selecting 2,3,4, and 5 these numbers also will be sorted by the four sorting methods respectively. In each case recording the time against the name of the method in a text file. This process continues until at least five different sizes

were generated and sorted. To see the result press 6 and press 0 to escape from the main menu and the system will take you back to the program.

Then after obtaining the various sizes and time a graph of measured time against the elements sizes will be drawn, containing the four methods used (insertion, bubble, merge and quick sorts).

From the graph generated, result will be interpreted and deductions made, conclusion will be drawn based on this data type.

Secondly, an input file will be created, in which admission numbers will be stored, a program will be coded which call this file, retrieve a certain size out of the four methods. This program also consists of a main menu, which comprises five numbers one to five. Representing generate list, insertion sort, bubble sort, merge sort and Quick sort.

If the user select 1, a message will appear on the screen as "ENTER THE NUMBER OF ELEMENTS TO RETRIEVE" if the user type the desire number, admission numbers of that size will be retrieved from the file. And then the system will take you to the main menu again. You then select 2,3,4,5 to sort these elements using inserts, bubble, merge and quick sort respectively. In each case, the time taken to sort out the elements by a particular method is recorded in atent file against the name of the methods. The system will take you back to

the main menu again, if you select 1, you are retrieving another list of different size. You type the number, the system will take you back to the main menu, then by selecting 2,3,4 and 5 these numbers also will be sorted by the four sorting methods respectively. In each case recording the time against the name of the method in a text file. This process continues 8520/until at least five differente sizes were generated and sorted. To see the result press 6 and press 0 to escape from the main menu and the system will take you back to the program, then after obtaining the various sites and time, a graph of measured time against the element size will be drawn, containing the four methods used (insertion, bubble, merge and quick sorts).

From the graph generated, result will be interpreted and deductions, made, conclusion will be drawn based on this data type.

Then the two sets of results will be compared that is result obtained when the data is from an array of integer and when it is from an array of strings will be compared and a final conclusion made.

3.5 DESCRIPTION OF SORTING ALGORITHMS

The sorting algorithm used in this project are described below:

3.5.1 INSERTION SORT

The separation performed by insertion sort is inserting an element in a sorted list. This is performed by scanning the elements below the element to be sorted. Each element that is smaller compared to the element to be inserted is moved up by the position. As soon as an element is found that is large compared with the element to be inserted, the seen terminates and the insertion occurs ahead of the larger elements.

For example, if the input data to be sorted is an array of five elements 360,75, 280,235,534 say. The fifth element is 534, when considered by itself, it is a sorted list of length one. The transition from fig. 1(a) to 1(b) consist of inserting 235 in the sorted list, 235 is inserted at the top of the list because it is less than 534. The sorted segment is of length two. From fig. 1(b) to (c) the transition is accomplished by inserting 280 in between 235 and 534, since it is less than 235. This is down by moving 235 up to make room for 280, and the sorted sublist has a length of three.

To obtain fig. (1) 75 is inserted at the top of the sorted list since it is smaller than all the elements in the sorted subset. Finally, fig (e) is obtained by inserting 360 in the list. This can be achieved by moving 75, 235 and 280 up to make room for the insertion of 360 in between 280 and 534. The figures referred to above are as shown below:-

Figure 1

360	360	360	360	75
75	75	75	95	235
280	280	235	235	280
235	235	280	280	360
534	534	534	534	534
A	B	C	D	E

In each of the figures above, the number(s) written boldly show the sorted sub – list.

Algorithm insertion sort.

Step 1 [start insertion] For K ←N – 1 down to 1 do through step 5

Step 2 [take next key] set j ← K + 1; and save ←d[k]

Step 3 [compare save! D[j]] while save > d[j] do

Set d [j – 1] ← d (j); and j ← j + 1

Step 4 [compare j:n] IF j > N THEN goto 5

Step 5 [insert item] set d [j – l] ← save

3.5.2 MERGE SORT

A procedure merge sort is used to implement the sort. It begins by comparing pairs of elements, one from each sub list. The smallest element is appended to a sorted list and is replaced by the next element from its sublist. This continues

until there are no more elements in one of the sublist. The remaining in the other sub lists are the appended to the sorted list, and the sort is completed.

For example, if A and B are two sorted sublists. A is of length four and B is of length six. Supposing both A and B are already sorted. To sort both A and B and put the result into C we have:

A	70	55	30	5						
B	60	50	45	20	10	3				
C	70	60	55	50	45	30	20	10	5	5

Length of C is the same as the length of A and B the general algorithm perform simple.

1. Merge two ordered sub-tables into a temporary vector
2. Copy the temporary vector into k.

Below is an algorithm for simple merge.

Procedure simple-merge (first, second, third).

Step 1 (initialize) 1 ← First and J ← Second and L ← 0

Step 2 (compare corresponding elements and output the smallest)

Repeat while 1 < Second and J < = third

If K [I] < = k [J] THEN

L ← I + 1

Temp [L] ← K [L]

ELSE

L ← L + 1

Temp [L] ← K [J]

J ← J + 1

Step 3 [copy the remaining unprocessed elements in output area]

If 1 > = second THEN

Repeat while j < = Third

L ← L + 1

Temp {I} ← K {J}

J ← J + 1

ELSE

Repeat while 1 < Second

L ← L + 1

Temp [L] ← k {I}

1 ← 1 + 1

Step 4 [copy elements in temporary vector into original area}

Repeat for 1 = 1,2,3,..... L

K [First, 1 + 1] ← temp [I]

Step 5 {finished} return.

3.5.3 QUICK SORT

If the elements to be sorted are stored in an array of a N components, d [1], d[2], d[3], ...d[n] say, then one step of the quick sort process would rearrange the elements. Each step of the Quick sort partitions a given list into three disjointed sub-lists. One of these sub-lists is a single element that is in its sorted position. The other two sub-lists share a common property each of them contains elements that are either larger than or smaller than the element that is in its sorted position. This permits each of these sublist to be sorted without reference to any element in the other sublist. Each step of Quick sort replaces the problem of sorting one long list by the problem of sorting two short list.

Figure 2a

53 59 56 52 55 58 52 57 54

To sort the elements in fig. 2 above so that 53 is in its sorted form using Quick sort we have:

Figure 2b

52 51 53 56 58 57 54

From fig 2b, is in its sorted position. All elements smaller than 53 and are arranged to the left of 53 and all elements greater than 53 are to the right of 53.

ALGORITHM QUICK SORT

Step 1 [initialize push-down] set k ← 1; Left [k] ←1, Right [k] ← 1;

Step 2 [iterate] While k> 0 Do through step 13; and stop.

Step 3 [pop of push down] set L ← Left [k]; R ← Right [k]; k←k-1.

Step 4 [sort large set] While R-L> =m do through step 12.

Step 5 [initialize] set 1←L; and Mid ←A [I].

Step 6 [compare Mid: A[J]] while Mid <A[J] Do set J←J-1.

Step 7 [pass compare?] if J<=1 THEN set A[I] ←Mid, and goto step 12.

Step 8 [interchange] set A [I] ←A[J]; A[J] ←Mid, and 1] ←1+1.

Step 9 [compare] A[I]: Mid]n while A[I] < Mid do set 1←1+1.

Step 10 [pass complete?] If L<=1 THEN set I←J; and go to step 11.

Step 11 [interchange] set A [J] ←A[I]; J←J-I; and go to step 5.

Step 12 [push down] set K←K + I; IF R-I <= I – L THEN set Left [K}←I-1; Right [K] ←R; and R←I-I ELSE.

Step 13 [start insertion] FOR J←L+1 to R DO THROUGH STEP 15.

Step 14 [take next key] set B←A[J], and I←J-I.

Step 15 [compare B:A[I]] while B<A[I} and I>=L do set A[I+1] ←A[I] and I←I + 1.

Step 16 [inset] set A [I+1] ←B

3.5.4 BUBBLE SORT

When bubble sort is used, there is always N – 1 passes required. During the first pass, the content of components one and two are compared, if they are out of order, then they are interchanged this process is repeated for elements in component two and three. There and four and so on. This method will cause

items with small elements to move or "Bubble up" After the first pass, the largest element will be in the position. On each successive pass, elements with the next largest value will be place in position N-1, N-2,...2, respectively, thereby resulting in a sorted order.

After each pass through the table, a check can be made to determine whether any element interchanges were made during the pass. It no interchanges occurred, then the table must be sorted and no further passes are required.

Procedure Bubble Sort (K, N,)

Step 1 [initialize) Last ←N (entire list assumed unsorted at his point)

Step 2 [Loop on pass index) Respect through step 5 for pass = 1,2,...N-1

Step 3 [initializes exchanges to counter for this pass] EXCHS←O

Step 4 [Perform pairs wise comparisons on unsorted elements] Repeat

FOR I=1,2,...,Last -1 IF K [I] > [I+I] THEN K[I] <> K [I+1]; EXEHS←EXEHS + 1

Step 5 (were any exchanges made o this pass?] IF EXEHS = O THEN

Return (Mission accomplished, return early} ELSE

LAST ← LAST – 1 (maximum of passes required).

CHAPTER FOUR

DATA ANALYSIS

4.1 INTRODUCTION

This chapter deals with the analysis of data. The result obtained from running the coded program is presented below. The execution time of algorithms can be affected by the capacity of the system on which the program is run, and also by the cycle in which the processor is in.

The time given below is obtained by running the coded program on a particular system. (American Mega trends 80386 with a memory capacity of 686k, 1024k ext, and operating system of Ms-Dos version 6.22)

For systems with higher capacity, the time is expected to be smaller than the one given below.

4.2 TABULATION OF DATA

Table 4.1(a): FOR SORTING ARRAY LIST OF INTEGER INITIALLY IN RANDOM ORDER

Time taken to sort (Us)

No. of items	Insert	Bubble	Merge	Quick
200	1.2100	1.3200	1.1600	1.1600
400	2.2000	3.5200	2.0300	0.3300

600	3.3500	5.3800	1.7500	0.3300
800	4.500	9.8900	1.7600	0.5500
1000	7.1400	15.3800	1.4300	0.6600
1200	10.1000	29.9700	1.5900	0.7600
1400	13.3600	26.4200	1.8100	0.7600
1600	13,1200	38.8900	2.0300	1.1000
1800	16.3200	40.5900	3.5700	0.9900
2000	18.8400	47.1800	2.4200	1.2100

Table 4.(b): FOR SORTING ARRAY LIST OF STRING

Time taken to sort (us)

No. of items	**Insert**	**Bubble**	**Merge**	**Quick**
20	1.1000	0.6600	0.7200	0.7700
40	0.7100	0.6600	0.6600	1.2100
60	0.8200	0.7200	0.7200	0.6600
80	1.7500	0.7100	0.6000	1.1000
100	0.7700	0.7700	0.6600	1.2100
120	0.8200	1.0400	0.8300	0.7100
140	0.8300	0.9900	0.9300	0.7700
160	0.9300	1.1500	0.9900	1.2100

FIG. 4.2 (a) TIME TAKEN TO SORT AN ARRAY OF INTEGER INITIALLY IN RANDOM ORDER

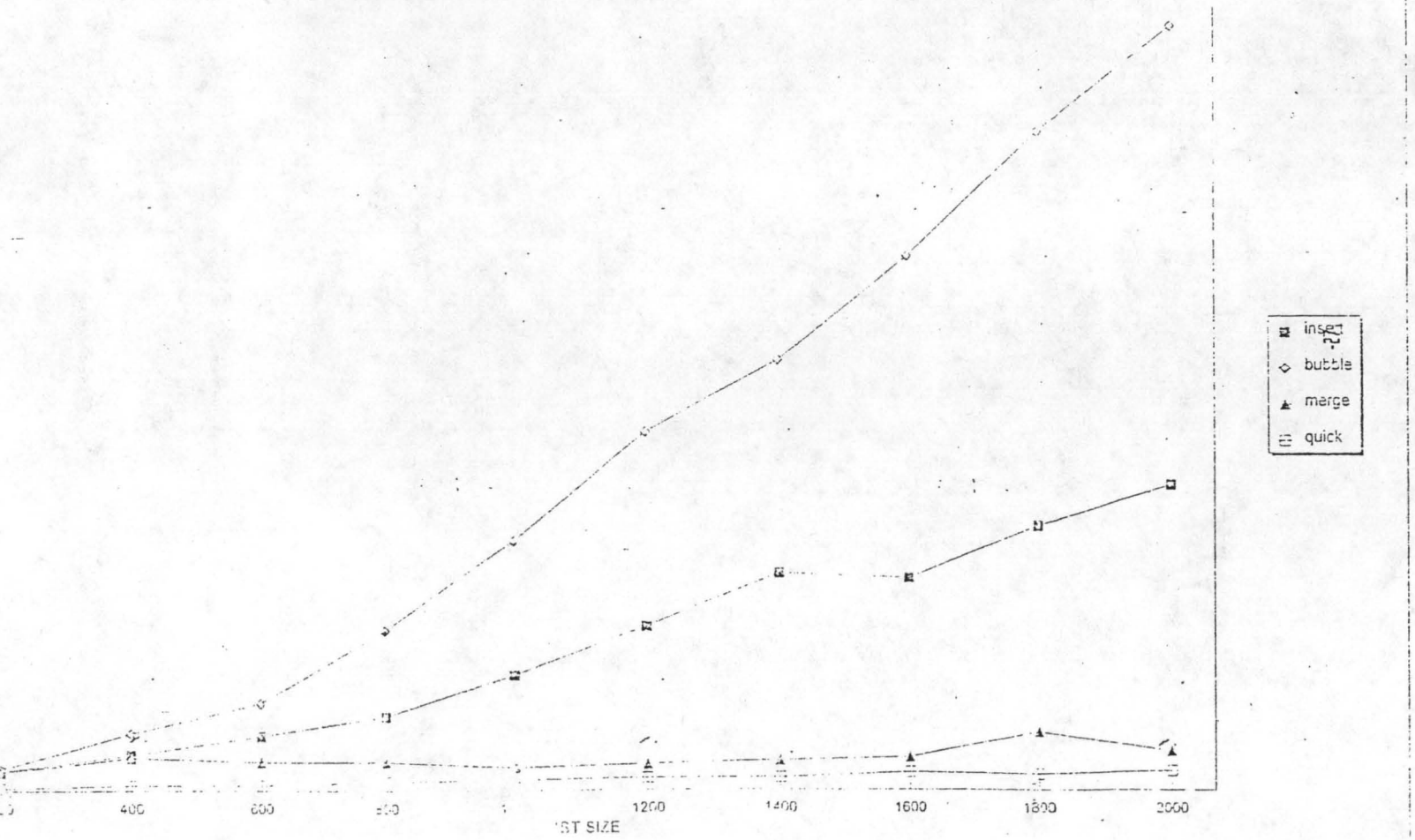
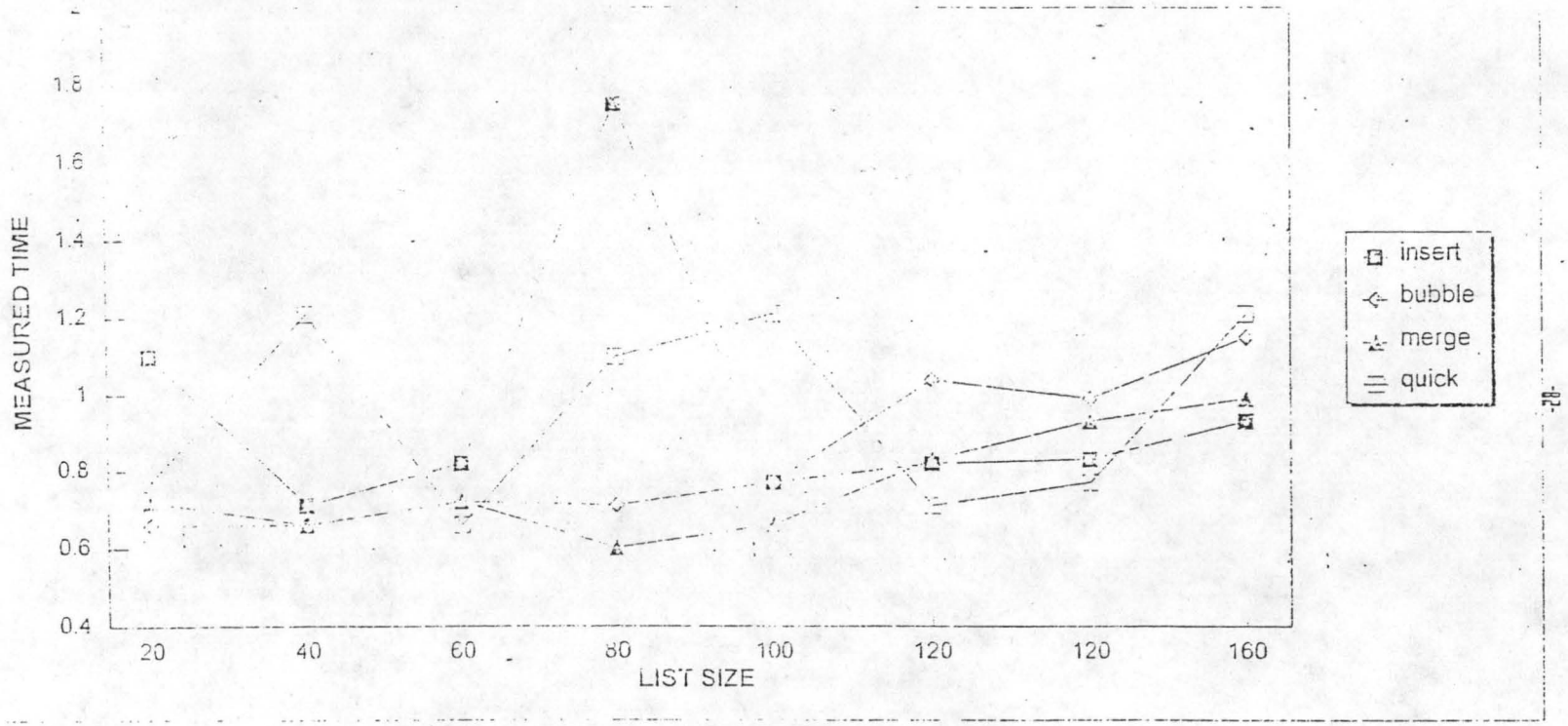


Fig 4.2(b) : TIME TAKEN TO SORT AN ARRAY LIST OF STRING



4.3 Analysis of Data From Sorting Array of Type Integer

From the graph, it can be observed that, the time taken for the Bubble sort to sort elements increases rapidly as the list size increases. While for the insert sort, the time taken to sort elements increases as fasten rate then in merge and Quick sorts as the size increases. For the merge sort, the time taken to sort elements in the list fluctuates at a very low rate. While for the Quick sort, as the size increases there is a very little or no change at all for the time taken to sort elements in the list.

In a not shell, the Quick sort is the fastest in terms of time taken to sort elements in an array of type integer and closely followed by merge sort, while insert sort is the moderate of the four considered methods of sorting, and the bubble sort is the slowest method of all the methods of sorting considered.

4.4 Analysis of Data From Sorting Array of Type String

From the graph, it can be seen that, the time taken for merge sort to sort the elements of type string increases at lower rate as the list size increases. While for insert sort, the time taken to sort the elements increases at low rate as the list size increases when compared to Quick and Bubble sort. While for the Quick sort, the time taken to sort element fluctuates at some intervals of list sizes. For the Bubble sort, the time increases at faster rate than the remaining sorting methods as the list size increases.

Thus, this enable us to say that Merge sort is the fastest of all the sorting techniques considered when sorting elements of type string, and Bubble sort is the, slowest when compared to the remaining sorting techniques.

CHAPTER FIVE

CONCLUSIONS AND RECOMMENDATIONS

5.1 CONCLUSIONS

From the research conducted, we can conclude that, the sorting algorithm which is the best for sorting elements of type string is MERGE SORT. While the sorting algorithm which is best for sorting elements of type integer is Quick sort. And also we can conclude from the graph and the analysis, that the size of the data to be sorted affects the time taken for a particular sorting algorithm to sort the elements in an array, this is so because, as the list size increases the time taken to sort also increases depending on the speed of the system. Finally the optimal sorting algorithm, which is good for both integer sorting and string sorting, is INSERT SORT.

5.2 RECOMMENDATIONS

I will like to make recommendation to programmers in general, that is to say, since time in particular is very important in every aspect of life, and computer programming is not an exception. Then, I will suggest that when the need of sorting elements of integer elements arises, the programmer should simply make use of Quick SORT. While in terms of sorting elements of type string, the MERGE SORT method should be used to solve the problems.

Finally, I will like to make use of this medium, to call on researchers who are interested in this topic to try and make research on the performance of other sorting algorithms such as Radix sort, Selection sort, Heap sort e.t.c.

REFERENCES

Adhert, O.J. (1968): Van Nostrands Scientific Encyclopedia. Fifth Edition. Can Nostrand Reinbod Company, pg. 87

George, O.A. (1987): Encyclopedia of Science and Technology I, McGraw-Hill pp.339-240.

Hedetnienmi, S.T. and S. E. Goodman, (1977): Introduction to the Design and Analysis of Algorithms. McGraw-Hill International Editions, Computer Science Series, University of Virgina, pp.207, 210, 215.

Holmes, B.J. (1989): Pascal Programming, Second Edition, DP Publications Ltd. Pp. 199-158.

Neil, W.W. and F.S. Daniel, (1989): Data Structures with Abstract Data types and Pascal Second Edition. California Polytechnic State University, pp.240-265.

Tremblay, J.P. and Hedetniemi, S.T. (1967): An Introduction to Data Structures with Applications. McGraw-Hill book company, Department of Computational Science University of Saskatchewan, Saskatoon, pp. 540-545.

APPENDIX

```
program  sorter;
 uses crt, dos;
 const lim = 2560; no_time = 30;
 TYPE
  Data = ARRAY [1..LIM] OF integer;

 VAR
  D : DATA; N,I : INTEGER;
  choice,a : char; st_time : real;
      HR,MIN,sec,SEC100:word; END_TIME:real;
      outf :text;


   PROCEDURE Generation (VAR Table:data; VAR N:integer);

    VAR  i, NUMB, SEED : INTEGER;
     PROCEDURE Randomize(VAR SEED:INTEGER);
      VAR HR,MIN,SEC,SEC100:WORD;
     BEGIN
      GETTIME(HR,MIN,SEC,SEC100);
      SEED := HR*360+MIN*60+SEC+SEC100;
     END;
    FUNCTION RANDOM (VAR SEED : INTEGER):REAL;
     CONST M = MAXINT; A = 2743; C =5923;
    BEGIN
     SEED := (SEED * A + C)MOD M;
     IF SEED < 0 THEN SEED := SEED + M;
     RANDOM := SEED/M;
    END;
    BEGIN

     RANDOMIZE(SEED);
     FOR i := 1 to N DO
      if (i <> 1) AND (table[i+1] <> table[i]) THEN
        table[i] := TRUNC(RANDOM(SEED)*10000);
    END;

   PROCEDURE Merge_Sort (var A:DATA; N:Integer);

     PROCEDURE MSort (low,high:integer);
      VAR mid : integer;

      PROCEDURE Merge (Low,mid,high :Integer);
       VAR i,j,k : integer; T : Data;
```

```
    h :=low ;i:=Low;j:=mid +1;
    WHILE (h <= mid) AND (j <= high) DO
     BEGIN
      IF  A[h] <= A[j] THEN
       BEGIN
        B[i] := A[h]; h := h+1;
       END
      ELSE
       BEGIN
        B[i] := A[j]; j:=j+1;
       END: { if }
      i := i+1:
     END; { while }
    IF h > mid THEN
     FOR k := j TO high DO
      BEGIN
       B[i] := A[k]; i:=i+1;
      END
    ELSE
     FOR k := h TO mid DO
     BEGIN
      B[i] := A[k]; i:=i+1:
     END; { if }
    FOR k := low TO high DO
     A[k] := B[k];
  END;   { MERGE }

 BEGIN     { MSORT }
 IF low < high  THEN
  BEGIN
   mid := (low + high) div 2;
   msort(low,mid);
   msort(mid+1,high);
   merge(low,mid,high);
  END; {if}
END; { MSORT }

BEGIN
 MSORT (1,N);
END; { MergeSort }

PROCEDURE InsertSort (var D:data;N:integer);

label 1;
    VAR j,k, save :integer;
```

```
    BEGIN

     FOR k := n-1 downto 1 DO
      BEGIN
       j := k+1; save := d[k];
       WHILE save > d[j]   DO
       BEGIN
        d[j-1] := d[j];
        j := j+1;
        IF j > n THEN GOTO 1
       END; { while }
      1: d[j-1] := save;
      END;

    END;{ insertsort}

procedure swap (var x, y :integer);
 var t:integer;
begin
 t := x;
 x:=y;
 y:=t;
end;
procedure BUbblesort (var A :data; N:integer);
 var i,j:integer;
begin
 for i := 2 to n do
  begin
   for j := n downto i do
    if a[j-1] > a[j] then
     swap(a[j-1],a[j]);
  end;
end;

{++++++++++++ +++++++ BEGINING OF QUICK SORT ++++++++++++++++++++}

  procedure quick _Sort2 (var X:data;n:integer);

     PROCEDURE QSORT2 (LO,HI:INTEGER);
      VAR
       MID:INTEGER;
      FUNCTION PARTITION (L,H:INTEGER):INTEGER;
      VAR  TK:INTEGER;
      BEGIN
```

```
      TK:= X[L];
      WHILE (L < H) DO
       BEGIN
        WHILE (TK <= X[H]) AND (L< H) DO
          H:=H-1;
        IF (L < H) THEN
         BEGIN
          X[L] := X[H]; L:=L+1;
          WHILE (TK >= X[L]) AND (L < H) DO
           L:=L +1;
          IF( L< H) THEN
           BEGIN
             X[H]:= X[L];
             H:=H-1;
           END
         END
       END;
       X[L]:= TK; PARTITION := L
    END;
    BEGIN
      IF (LO < HI) THEN
       BEGIN
        MID:= PARTITION(LO,HI);
        QSORT2(LO,MID-1);
        QSORT2(MID+1,HI)
       END
    END;
   begin { quicksort}
   qsort2(1,N);
  end; {quicksort}
{++++++++++++++++++++ END OF QUICK SORT ++++++++++++++++++++}

  PROCEDURE offerMenu ( VAR choice : char);
   VAR
    Hr, mn, sec, sec10 : word;
  BEGIN
   CLRSCR;
   gotoxy(28,6); Writeln(' MAIN MENU     ') ;WRITELN;
    gotoxy(25,8); Writeln('1.  GENERATE LIST') ;
    gotoxy(25,9); Writeln('2.  INSERT SORT ');
    gotoxy(25,10); Writeln('3.  BUBBLE SORT ');
    gotoxy(25,11); Writeln('4.  MERGE SORT  ');
    gotoxy(25,12); Writeln('5.  QUICK SORT  ');
    gotoxy(25,13); Writeln('6. DISPLAY RESULT');
    gotoxy(24,16); Write('Enter your choice (0 to stop) : '); Readln(choice);
```

```
  choice := upcase(choice);
  {textbackground(10);}
END; { offerMenu}

procedure DispLay (D:data; N:integer);
 var i  :integer;
begin
 CLRSCR;GOTOXY(1,2);
 for i := 1 to N do
  BEGIN
   write(d[i]:5);
   IF I MOD 20 = 0 THEN
     writeln;
  END;
  readln;
END;


procedure disp_Result (VAR time:text);

 VAR  SZ :integer;T_IS,T_BS,T_MS,T_QS:real;
BEGIN
 RESET(time);
 while not eof(TIME) do
  BEGIN
   Readln(time,SZ,T_IS,T_BS,T_MS,T_QS);

     Writeln(SZ:5,T_IS:18:4,T_BS:13:4,T_MS:13:4,T_QS:13:4);
 END;

 WRITELN; WRITELN;
END; {DISP_RESULT}

procedure initialise(var HR,MIN,sec,SEC100:word);
begin
 HR  := 0; MIN := 0; sec := 0;SEC100:=0;
end;



PROCEDURE Display2 (VAR time:text);
 VAR ANS : CHAR;
BEGIN
 clrscr;
 GOTOXY(20,12);WRITE('Want see result ? (Y/N): '); READLN(ANS);
```

```
  IF UPCASE(ANS) = 'Y' THEN
   BEGIN
    CLRSCR;
     writeln('Time taken to sort ':50);
     Writeln;
     Writeln('NO. of Items   Insert      Bubble      Merge     Quick');
     Writeln;
    Disp_Result (TIME);
    WRITE('PRESS ANY KEY TO CONTINUE..'); READLN;
   END;
END;


BEGIN
 assign(outf,'Myfile.dat');rewrite(outf);

 clrscr; textcolor(lighteyan);
 {SETTIME(HR,MIN,SEC,SEC100);}

 REPEAT
   offerMenu(choice);
   CASE choice  OF
   '1' : begin
         clrscr; gotoxy(20,10);
         Write('Number of elements to generate : ');
         Readln(N); Write(outf,N:8);
         END;
   '2' : begin
         initialise(HR,MIN,sec,SEC100);
         getTime(HR,MIN,sec,SEC100);
         Generation(D,N);
         St_time := HR*3600+MIN*60+SEC+SEC100*0.01;
         {WRITELN(ST_TIME);}
         {FOR I := 1 TO no_time DO}
          InsertSort(D,N);

         write('**'); a := readkey;
         gettime(HR,MIN,sec,SEC100);
         end_time := HR*3600+MIN*60+sec+SEC100*0.01;
         {WRITELN(END_TIME);READLN;}
         write(outf,(end_time-st_time):15:4);
         {display(d,n);}
       END;
   '3' : begin
          initialise(HR,MIN,sec,SEC100);
```

```
      getTime(HR,MIN,sec,SEC100);
      Generation(D,N);
      St_time := HR*3600+MIN*60+SEC+SEC100*0.01;
      {FOR I := 1 TO no_time DO}
        BubbleSort(D,N);

      {delay(1000);}
      write('**'); a:= readkey;
      gettime(HR,MIN,sec,SEC100);
      end_time:=HR*3600+MIN*60+sec+SEC100*0.01;
      write(outf,(end_time-st_time):15:4);
      { display(d,n);}
     END;
'4' : BEGIN
      initialise(HR,MIN,sec,SEC100);

      getTime(HR,MIN,sec,SEC100);
      Generation(D,N);
      St_time := HR*3600+MIN*60+SEC+SEC100*0.01;
      {FOR I := 1 TO no_time DO }
       Merge_Sort(D,N);

      {delay(1000);}
      write('**'); a:= readkey;

      gettime(HR,MIN,sec,SEC100);
      end_time:=HR*3600+MIN*60+sec+SEC100*0.01;
      write(outf,(end_time-st_time):15:4);
      {display(d,n);}
     END;
'5' : BEGIN

      initialise(HR,MIN,sec,SEC100);
      getTime(HR,MIN,sec,SEC100);
      Generation(D,N);

      St_time := HR*3600+MIN*60+SEC+SEC100*0.01;
      {FOR I := 1 TO no_time DO}
      Quick_Sort2(D,N);


      gettime(HR,MIN,sec,SEC100);
      end_time:=HR*3600+MIN*60+sec+SEC100*0.01;
      writeln(outf,(end_time-st_time):15:4);
     {display(d,n);}
```

```
        END;

    '6' : BEGIN CLOSE(OUTF); display2(OUTF);END;
    '0' : close(outf);
     END;
  UNTIL (choice = '0') OR (choice = '6');

END.
```

```
program Sort_merge;
uses crt, dos;
const lim = 1000;
TYPE
 STR1 = STRING[10];
 Data = ARRAY [1..LIM] OF STR1;

VAR
 D : DATA; N,i : INTEGER;
 choice,a : char; st_time : real;
     HR,MIN,sec,SEC100:word; END_TIME:real;
     outf,INPF :text;
PROCEDURE Generation (VAR afile:Text; VAR Table:data; VAR N:integer);
  VAR ADNO : STR1;
 BEGIN
  RESET(afile);
  FOR i := 1 to N DO
  BEGIN
   READLN(afile,ADNO);
   Table[i] := ADNO;
  END;
 END;

 PROCEDURE Merge_Sort (var A:DATA; N:Integer);

  PROCEDURE MSort (low,high:integer);
   VAR mid : integer;

   PROCEDURE Merge ( Low,mid,high :Integer);
    VAR h,i,j,k :integer; B : Data;
   BEGIN
    h :=low; i:=Low;j:=mid +1;
    WHILE (h <= mid) AND (j <= high) DO
     BEGIN
      IF A[h] <= A[j] THEN
       BEGIN
        B[i] := A[h]; h := h+1;
       END
      ELSE
       BEGIN
        B[i] := A[j]; j:=j+1;
       END; { if }
      i := i+1;
     END; { while }
    IF h = mid THEN
```

```
    FOR k := j TO high DO
     BEGIN
       B[i] := A[k]; i:=i+1;
     END
    ELSE
     FOR k := h TO mid DO
     BEGIN
       B[i] := A[k]; i:=i+1;
     END; { if }
    FOR k := low TO high DO
     A[k] := B[k];
  END;  { MERGE }

BEGIN     { MSORT }
 IF low < high THEN
  BEGIN
   mid := (low + high) div 2;
   msort(low,mid);
   msort(mid+1,high);
   merge(low,mid,high);
  END; {if}
END; { MSORT }

BEGIN
 MSORT (1,N);
END; { MergeSort }

PROCEDURE InsertSort (var D:data;N:integer);

label 1;
    VAR j,k : integer; SAVE : STR1;

BEGIN

FOR k := n-1 downto 1 DO
 BEGIN
  j := k+1; save := d[k];
  WHILE save > d[j] DO
   BEGIN
    d[j-1] := d[j];
    j := j+1;
    IF j > n THEN GOTO 1
   END; { while }
  1: d[j-1] := save;
 END;
```

```
    END;{ insert sort}

procedure swap (var x, y :STR1);
 var t: STR1;
begin
 t := x;
 x:=y;
 y:=t;
end;
procedure BUbbl :sort (var A :data; N:integer);
 var i,j:integer;
begin
 for i := 2 to n do
  begin
   for j := n downto i do
    if a[j-1] > a[j] then
     swap(a[j-1],a[j]);
  end;
 end;

{++++++++++++  ++++++ BEGINING OF QUICK SORT ++++++++++++++++++++}

  procedure quick_Sort2 (var X:data;n:integer);

    PROCEDURE QSORT2 (LO,HI:INTEGER);
     VAR
      MID:INTEGER;
     FUNCTION PARTITION(L,H:INTEGER):INTEGER;
     VAR  TK:STR1;
     BEGIN
     TK:= X[L];
     WHILE (L < H) DO
      BEGIN
       WHILE (TK <= X[H]) AND (L< H) DO
        H:=H-1;
       IF (L < H) THEN
       BEGIN
        X[L] := X[H]; L:=L+1;
        WHILE (TK >= X[L]) AND (L < H) DO
         L:=L + 1;
        IF (L < H) THEN
        BEGIN
          X[H]:= X[L];
          H:=H-1;
```

```
          END
        END
      END;
      X[L]:=TE  PARTITION := L
    END;
    BEGIN
      IF (LO <  HI) THEN
        BEGIN
          MID:= PARTITION(LO,HI);
          QSORT2(LO,MID-1);
          QSORT2(MID+1,HI)
        END
    END;
  begin { quicksort}
   qsort2(1,N);
  end; {quicksort}
{++++++++++++++++++ END OF QUICK SORT ++++++++++++++++++++}

  PROCEDURE offerMenu ( VAR choice : char);
   VAR
    Hr, mn, sec, sec10 : word;
  BEGIN
   CLRSCR;
   gotoxy(28,6 ); Writeln(' MAIN MENU    ') ;WRITELN;
   gotoxy(25,8 ); Writeln('1. GENERATE LIST') ;
   gotoxy(25,9 ); Writeln('2. INSERT SORT ');
   gotoxy(25,10); Writeln('3. BUBBLE SORT ');
   gotoxy(25,11); Writeln('4. MERGE SORT  ');
   gotoxy(25,12); Writeln('5. QUICK SORT  ');
   gotoxy(25,13); Writeln('6. DISPLAY RESULT');
   gotoxy(24,15); Write('Enter your choice (0 to stop) : '); Readln(choice);
   choice := upcase(choice);
   {textbackground(10);}
  END; { offerMenu}

  procedure DisplayData(D:data; N:integer);
   var i :integer;
  begin
   CLRSCR; GOTOXY(1,2);
   for i := 1 to N do
    BEGIN
     write(d[i]:10);
     IF (i MOD 7 = 0) THEN
       writeln;
    END;
```

```
 readln;
END;


procedure disp_Result (VAR time:text);

 VAR SZ :integer;T_IS,T_BS,T_MS,T_QS:real;
BEGIN
 RESET(time);
 while not eof (TIME) do
  BEGIN
   Readln(time,SZ,T_IS,T_BS,T_MS,T_QS);

     Writeln(SZ:5,T_IS:18:4,T_BS:13:4,T_MS:13:4,T_QS:13:4);
  END;

 WRITELN; WRITELN;
END; {DISP_RESULT}

procedure initialise(var HR,MIN,sec,SEC100:word);
begin
 HR := 0; MIN := 0; sec := 0;SEC100:=0;
end;


PROCEDURE Display2 (VAR time:text);
 VAR ANS : CHAR;
BEGIN
 clrscr;
 GOTOXY(20,12);WRITE('Want see result ? (Y/N): '); READLN(ANS);
 IF UPCASE(ANS) = 'Y' THEN
   BEGIN
    CLRSCR;
     writeln('Time taken to sort ':50);
     Writeln;
     Writeln('NO. of Items   Insert      Bubble      Merge     Quick');
     Writeln;
    Disp_Result (TIME);
    WRITE('PLEASS ANY KEY TO CONTINUE..'); READLN;
   END;
END;
PROCEDURE ReadSize (var N : integer);
 const flsize = 172;
BEGIN
```

```
  REPEAT
    clrscr; gotoxy(20,10);
    Write('Number of elements to generate : ');
    Readln(N);
  UNTIL N <= FILSIZE;
END;



BEGIN
 assign(outf,'MY FILE.dat');rewrite(outf);  { OPEN OUT FILE}
 assign(INPF,'A:\DATA.DAT');reSET(INPF);  { OPEN INPUT FILE}
  clrscr; textcolor(lightcyan);
 {SETTIME(HR,MIN,SEC,SEC100);}

 REPEAT
   offerMenu(choice);
   CASE choice OF
    '1' : begin ReadSize (N); WRITE(OUTF, N:5); END;

    '2' : begin
          initialise(HR,MIN,sec,SEC100);
          getTime( HR,MIN,sec,SEC100);
          Generation(INPF,D,N);
          St_time := HR*3600+MIN*60+SEC+SEC100*0.01;
          {WRITELN(ST_TIME);}
          {FOR I := 1 TO no_time DO}
           InsertSort(D,N);

          write('**'); a := readkey;
          gettime(HR,MIN,sec,SEC100);
          end_time := HR*3600+MIN*60+sec+SEC100*0.01;
          {WRITELN(END_TIME);READLN;}
          writeout (end_time-st_time):15:4);
          {display(D,N);}
         END;
    '3' : begin
           initialise(HR,MIN,sec,SEC100);
           getTime(HR,MIN,sec,SEC100);
           Generation(INPF,D,N);
           St_time := HR*3600+MIN*60+SEC+SEC100*0.01;
           {FOR I := 1 TO no_time DO}
            BubbleSort(D,N);
```

```
        {delay(1000);}
        write('**'); a:= readkey;
        gettime(HR,MIN,sec,SEC100);
        end_time:=HR*3600+MIN*60+sec+SEC100*0.01;
        writeln(out,(end_time-st_time):15:4);
        {display(d.n);}
      END;
  '4' : BEGIN
        initialise(HR.MIN,sec,SEC100);

        getTime(HR,MIN,sec,SEC100);
        Generation(INPF.D,N);
        St_time:=HR*3600+MIN*60+SEC+SEC100*0.01;
        {FOR I:=1 TO no_time DO }
         Merge_sort(D,N);

        write('**'); a:= readkey;

        gettime(HR.MIN.sec.SEC100);
        end_time:=HR*3600+MIN*60+sec+SEC100*0.01;
        writeln(out,(end_time-st_time):15:4);
        {display(d.n);}
      END;
  '5' : BEGIN

        initialise(HR.MIN.sec,SEC100);
        getTime(HR.MIN.sec,SEC100);
        Generation(INPF.D,N);

        St_time:=HR*3600+MIN*60+SEC+SEC100*0.01;
        {FOR I:=1 TO no_time DO}
        Quick_Sort2(D.N);

        write('**'); a := readkey;
        gettime(HR.MIN.sec,SEC100);
        end_time:=HR*3600+MIN*60+sec+SEC100*0.01;
        writeln(out,(end_time-st_time):15:4);
        {display(d.n);}
      END;

  '6' : BEGIN reset(OUTF); display2(OUTF);END;
  '0' : close(output);
   END;
UNTIL (choice = '0') OR (choice = '6');
```

```
        {delay(1000);}
        write('***'); a:=readkey;
        gettime(HR,MIN,sec,SEC100);
        end_time:=HR*3600+MIN*60+sec+SEC100*0.01;
        writeln(output,(end_time-st_time):15:4);
        {display(ln);}
      END;
'4' : BEGIN
      initialise(HR,MIN,sec,SEC100);

      getTime(HR,MIN,sec,SEC100);
      Generation(INPF,D,N);
      St_time:=HR*3600+MIN*60+SEC+SEC100*0.01;
      {FOR I:=1 TO no_time DO }
       MergeSort(D,N);

      write('***'); a:=readkey;

      gettime(HR,MIN,sec,SEC100);
      end_time:=HR*3600+MIN*60+sec+SEC100*0.01;
      writeln(output,(end_time-st_time):15:4);
      {display(ln);}
      END;
'5' : BEGIN

      initialise(HR,MIN,sec,SEC100);
      getTime(HR,MIN,sec,SEC100);
      Generation(INPF,D,N);

      St_time:=HR*3600+MIN*60+SEC+SEC100*0.01;
      {FOR I:=1 TO no_time DO}
      QuickSort2(D,N);

      write('***'); a:=readkey;
      gettime(HR,MIN,sec,SEC100);
      end_time:=HR*3600+MIN*60+sec+SEC100*0.01;
      writeln(output,(end_time-st_time):15:4);
      {display(ln);}
      END;

'6' : BEGIN CLOSE(OUTF); display2(OUTF);END;
'0' : close(output);
 END;
UNTIL (choice='0') OR (choice='6');

END.
```