CSC 223 LECTURE NOTES
FILES ORGANIZATION, DATA STRUCTURE, DECISION TABLES, SORTING AND MERGING

THE DATA HIERARCHY

A computer system organizes data in a hierarchy that starts with bits and bytes and progresses to fields, records, files, and databases.

1. Bit: This represents the smallest unit of data a computer can handle. A group of bits, called a byte, represents a single character, which can be a letter, number or other symbol.

2. Field: A field is a particular place in a record where an item of information can be held or a grouping of characters into a word, group of words or a complete number (e.g. a person's first name or age), is called a field.

3. Record: A group of related fields, such as a student's name, class, date admitted, age or record is a collection of related items of data treated as a unit.

4. File:A file is organized collection of related records which are processed together. It is also referred to as a data set. The files is a collection of records relating to some class of object e.g. records of all insurance policies issue by an insurance company, records of all employees of firm, student records etc. A group of records of the same type (e.g. the records of all students in the class) is called a file.

5. Database: A group of related files (e.g. the personal history, examinations records and payments history files) make up a database. A record describes an entity. An entity is a person, place, thing, or event on which we maintain information. An employee record is an entity in a personnel records file and maintains information on the employees in that organization. Each characteristic or quality describing a particular entity is called an attribute. For example, employee name, address, age, gender, date employed is an attribute each of the entity personnel. The specific values that these attributes can have can be found in the field of the record describing the entity. Every record in the file contains at least one field that uniquely identifies that record so that the record can be retrieved, changed, modified or sorted. This identifier is called the key field. An example of a key field is the employee number for a personnel record containing employee data such as name, address, age, job title etc.

COMPUTER FILE

A computer file is a collection of related records / information that is stored in a computer system and can be identified by its full path name. File can also be defined as a group of records pertaining to a specific item e.g. payroll file. This contains all the payroll records within the organization's records. Computer files are so called because they are the computer equivalent of card, paper, or microfiche files in the traditional office environment. Computer files provide a way to organize the resources used to permanently store information inside a computer.

A file has got a layout which show, for each different type of record in each file, the name of each file, its length and its locations within the record. Files are created when need for a particular collection of records is recognized. However, the creation and maintenance of files are major factor in the work load of a computer information processing system.

Purpose: A file holds data which is required for providing information. Some files are processed at regular intervals to provide this information (e.g. payroll file) and others which is required at regular intervals (e.g. file containing prices of items).

Ways of viewing file

There are two common ways of viewing files:

1. Logical Files

A logical file is a file viewed in terms of what data items its records, contain and what processing operations may be performed upon the file. The user of the file will normally adopt such a view.

2. Physical Files

A physical file is a file viewed in terms of how the data are stored on a storage device such as a magnetic disk and how the processing operations are made possible.
A logical file usually gives rise to a number of alternative physical file implementation

Protecting Files

Many modern computer systems provide methods for protecting files against accidental and deliberate damage. Computers that allow for multiple users implement file permissions to control who may or may not modify, delete, or create files and folders. A given user may be granted only permission to modify a file or folder, but not to delete it; or a user may be given permission to create files or folders, but not to delete them. Permissions may also be used to allow only certain users to see the contents of a file or folder. Permissions protect against unauthorized tampering or destruction of information in files, and keep private information confidential by preventing unauthorized users from seeing certain files.

Storing Files

In physical terms, most computer files are stored on hard disks – spinning magnetic disks inside a computer that can record information indefinitely. Hard disks allow almost instant access to computer files.

On large computers, some computer files may be stored on magnetic tape. Files can also be stored on other media in some cases, such as write-able Compact discs, Zip drives etc.

Backing up Files

When computer files contain information that is extremely important, a back up process is used to protect against disasters that mighty destroy the files. Backing up files simply means making copies of the files in a separate location so that they can be restored if something happens to the computer, or if they are deleted accidentally.

There are many ways to back up files. Most computer systems provide utility programs to assist in the back-up process, which can become very time-consuming if there are many files to safeguard. Files are often copied to removable media such as write-able CDs or cartridge tapes. Copying files to another hard disk in the same computer protects against failure of one disk, but if it is necessary to protect against failure or destruction of the entire computer, then copies of the files must be made on other media that can be taken away from the computer and stored in a safe, distant location.

Types of file

1. Master file

These are files of a fairly permanent nature e.g. customer ledger, payroll, inventory, etc. A feature to note is the regular updating of these files to show a current position. For example customer' order will be processed by increasing the "balance owing" figure on a customer ledger record. It is seen therefore that master records will contain both data of a static nature, e.g. a customer name and address, and data which by its nature will change each time a transaction occurs, e.g. the "balance" personal file.

2. Movement file/Transaction file

This is made up of the various transactions created from the source documents. In a sales ledger application, the file will contain all the orders received at a particular time. This file will be used to update the master file. As soon as it has been used for this purpose, it is no longer required. It will therefore have a very short life; because it will be replaced by a file containing the next batch addresses E.g. Sales Invoicing file, purchase order file and payroll file.

3. **Reference file**

This is a file with a reasonable amount of permanency. Example of data used for reference purposes are price lists, tables of rates of pay, names and addresses.

4. Working file: Working file:

A file that is used to support computer procedures. Its content could be same with the transaction and or master file. It is usually of temporary nature and derived by interrogating the master/transaction files. An example is a file containing the list of debtors to an organization.

Processing activities on a file

We will need to have access to particular records in the files in order to process them. The major processing activities are given below:

1. **Updating:** This is when data on the master record is changed to reflect a current position e.g. updating a customer ledger record with new orders. Note that the old data on the record is replaced by the new data.

2. **Referencing:** When access is made to a particular record to ascertain therein e.g. reference is made to a “price” file during an invoicing run. Note that this does not involve any alteration to the record itself.

3. **Sort file:**This can be defined as a working file or record to be sequenced. This may be the original or a copy of the transaction file, master file or report file.

4. **File maintenance:** New records must be added to a file and records need to be deleted. Prices change, and the file must be altered. Customer’s addresses also change and new addresses have to be inserted to bring the file up to date. These particular activities come under the heading of “maintaining” the file. File maintenance can be carried out as a separate run but the insertion and deletion of records is sometimes combined with Updating.

5. **File enquiry or interrogation:** This is similar in concept to referencing. It involves the need to ascertain a piece of information from, say, a master record. For example, a customer may query a statement sent to him. A “file enquiry” will get the data in dispute from the record so that the query may be settled.

File organization

Organization of a file on tape is simply a matter of placing the records one after the other on to the tape.

There are four basic methods of organizing files on disk:

1. **Serial:** Exactly as for tape. Records are placed onto the disk one after the other with no regard for sequence.
2. **Sequential:** Again as for tape; records are written onto the disk but in a defined sequence according to the record keys.
3. **Indexed sequential:** Records are stored in sequence as for 20b but with one important difference – an index is provided to enable individual records to be located.
4. **Random:** Records are actually placed onto the disk “at random”, that is to say there is no obvious relationship between the records.

DATA STRUCTURES

Data Structure can be defined as the group of data elements which provides an efficient way of storing and organising data in the computer so that it can be used efficiently. Some examples of Data Structures are arrays, Linked List, Stack, Queue, etc. Data Structures are widely used in almost every aspect of Computer Science i.e. Operating System, Compiler Design, Artifical intelligence, Graphics and many more.

Data Structures are the main part of many computer science algorithms as they enable the programmers to handle the data in an efficient way. It plays a vitle role in enhancing the performance of a software or a program as the main function of the software is to store and retrieve the user's data as fast as possible

The data structure is not any programming language like C, C++, java, etc. It is a set of algorithms that we can use in any programming language to structure the data in the memory. Data Structure is a way to store and organize data so that it can be used efficiently.

To structure the data in memory, 'n' number of algorithms were proposed, and all these algorithms are known as Abstract data types. These abstract data types are the set of rules.

Basic Terminology

Data structures are the building blocks of any program or the software. Choosing the appropriate data structure for a program is the most difficult task for a programmer. Following terminology is used as far as data structures are concerned

Data: Data can be defined as an elementary value or the collection of values, for example, student's name and its id are the data about the student.

Group Items: Data items which have subordinate data items are called Group item, for example, name of a student can have first name and the last name.

Record: Record can be defined as the collection of various data items, for example, if we talk about the student entity, then its name, address, course and marks can be grouped together to form the record for the student.

File: A File is a collection of various records of one type of entity, for example, if there are 60 employees in the class, then there will be 20 records in the related file where each record contains the data about each employee.

Attribute and Entity: An entity represents the class of certain objects. it contains various attributes. Each attribute represents the particular property of that entity.

Field: Field is a single elementary unit of information representing the attribute of an entity.

Need of Data Structures

As applications are getting complexed and amount of data is increasing day by day, there may arrise the following problems:

1. Processor speed: To handle very large amout of data, high speed processing is required, but as the data is growing day by day to the billions of files per entity, processor may fail to deal with that much amount of data.

2. Data Search: Consider an inventory size of 106 items in a store, If our application needs to search for a particular item, it needs to traverse 106 items every time, results in slowing down the search process.

3. Multiple requests: If thousands of users are searching the data simultaneously on a web server, then there are the chances that a very large server can be failed during that process in order to solve the above problems, data structures are used. Data is organized to form a data structure in such a way that all items are not required to be searched and required data can be searched instantly.

Advantages of Data Structures

Efficiency: Efficiency of a program depends upon the choice of data structures. For example: suppose, we have some data and we need to perform the search for a perticular record. In that case, if we organize our data in an array, we will have to search sequentially element by element. hence, using array may not be very efficient here. There are better data structures which can make the search process efficient like ordered array, binary search tree or hash tables.

Reusability: Data structures are reusable, i.e. once we have implemented a particular data structure, we can use it at any other place. Implementation of data structures can be compiled into libraries which can be used by different clients.

Abstraction: Data structure is specified by the ADT which provides a level of abstraction. The client program uses the data structure through interface only, without getting into the implementation details.

Types of Data Structures

There are two types of data structures:

- Primitive data structure
- Non-primitive data structure

Primitive Data structure

The primitive data structures are primitive data types. The int, char, float, double, and pointer are the primitive data structures that can hold a single value.

Non-Primitive Data structure

The non-primitive data structure is divided into two types:

- Linear data structure
- Non-linear data structure

Linear Data Structure

The arrangement of data in a sequential manner is known as a linear data structure. The data structures used for this purpose are Arrays, Linked list, Stacks, and Queues. In these data structures, one element is connected to only one another element in a linear form.

When one element is connected to the 'n' number of elements known as a non-linear data structure. The best example is trees and graphs. In this case, the elements are arranged in a random manner.

We will discuss the above data structures in brief in the coming topics. Now, we will see the common operations that we can perform on these data structures.

Data structures can also be classified as:

- **Static data structure:** It is a type of data structure where the size is allocated at the compile time. Therefore, the maximum size is fixed.
- **Dynamic data structure:** It is a type of data structure where the size is allocated at the run time. Therefore, the maximum size is flexible.

Data Structure Classification

Linear Data Structures: A data structure is called linear if all of its elements are arranged in the linear order. In linear data structures, the elements are stored in non-hierarchical way where each element has the successors and predecessors except the first and last element.

Types of Linear Data Structures are given below:

Arrays: An array is a collection of similar type of data items and each data item is called an element of the array. The data type of the element may be any valid data type like char, int, float or double.

The elements of array share the same variable name but each one carries a different index number known as subscript. The array can be one dimensional, two dimensional or multidimensional.

The individual elements of the array age are:

age[0], age[1], age[2], age[3],......... age[98], age[99].

Linked List: Linked list is a linear data structure which is used to maintain a list in the memory. It can be seen as the collection of nodes stored at non-contiguous memory locations. Each node of the list contains a pointer to its adjacent node.

Stack: Stack is a linear list in which insertion and deletions are allowed only at one end, called **top**.

A stack is an abstract data type (ADT), can be implemented in most of the programming languages. It is named as stack because it behaves like a real-world stack, for example: - piles of plates or deck of cards etc.

Queue: Queue is a linear list in which elements can be inserted only at one end called **rear** and deleted only at the other end called **front**.

It is an abstract data structure, similar to stack. Queue is opened at both end therefore it follows First-In-First-Out (FIFO) methodology for storing the data items.

Non Linear Data Structures: This data structure does not form a sequence i.e. each item or element is connected with two or more other items in a non-linear arrangement. The data elements are not arranged in sequential structure.

Types of Non Linear Data Structures are given below:

Trees: Trees are multilevel data structures with a hierarchical relationship among its elements known as nodes. The bottommost nodes in the herierchy are called **leaf node** while the topmost node is called **root node**. Each node contains pointers to point adjacent nodes.

Tree data structure is based on the parent-child relationship among the nodes. Each node in the tree can have more than one children except the leaf nodes whereas each node can have atmost one parent except the root node. Trees can be classfied into many categories which will be discussed later in this tutorial.

Graphs: Graphs can be defined as the pictorial representation of the set of elements (represented by vertices) connected by the links known as edges. A graph is different from tree in the sense that a graph can have cycle while the tree can not have the one.

Operations on data structure

1) **Traversing:** Every data structure contains the set of data elements. Traversing the data structure means visiting each element of the data structure in order to perform some specific operation like searching or sorting.

Example: If we need to calculate the average of the marks obtained by a student in 6 different subject, we need to traverse the complete array of marks and calculate the total sum, then we will devide that sum by the number of subjects i.e. 6, in order to find the average.

2) **Insertion:** Insertion can be defined as the process of adding the elements to the data structure at any location.

If the size of data structure is **n** then we can only insert **n-1** data elements into it.

3) **Deletion:**The process of removing an element from the data structure is called Deletion. We can delete an element from the data structure at any random location.

If we try to delete an element from an empty data structure then **underflow** occurs.

4) **Searching:** The process of finding the location of an element within the data structure is called Searching. There are two algorithms to perform searching, Linear Search and Binary Search. We will discuss each one of them later in this tutorial.

5) **Sorting:** The process of arranging the data structure in a specific order is known as Sorting. There are many algorithms that can be used to perform sorting, for example, insertion sort, selection sort, bubble sort, etc.

6) **Merging:** When two lists List A and List B of size M and N respectively, of similar type of elements, clubbed or joined to produce the third list, List C of size (M+N), then this process is called merging

Advantages of Data structures

The following are the advantages of a data structure:

- **Efficiency:** If the choice of a data structure for implementing a particular ADT is proper, it makes the program very efficient in terms of time and space.
- **Reusability:** he data structures provide reusability means that multiple client programs can use the data structure.
- **Abstraction:** The data structure specified by an ADT also provides the level of abstraction. The client cannot see the internal working of the data structure, so it does not have to worry about the implementation part. The client can only see the interface.

DECISION TABLES

Decision tables are a concise visual representation for specifying which actions to perform depending on given conditions. They are algorithms whose output is a set of actions. The information expressed in decision tables could also be represented as decision trees or in a programming language as a series of if-then-else and switch-case statements.

Each decision corresponds to a variable, relation or predicate whose possible values are listed among the condition alternatives. Each action is a procedure or operation to perform, and the entries specify whether (or in what order) the action is to be performed for the set of condition alternatives the entry corresponds to.

Would tables be valuable in systems analysis and computer programming?

A decision table is a tabular representation of a particular set of:

(1) Conditions: Variables that must be considered in reaching a decision.

(2) Actions: Operations that must be carried out when a given set of conditions exist.

(3) Rules: Specific sets of conditions and the actions dictated by these conditions.

(4) Entries: Additional information about either a condition or action pertinent to a particular rule.

To make them more concise, many decision tables include in their condition alternatives a don't care symbol. This can be a hyphen[1][2][3] or blank,[4] although using a blank is discouraged as it may merely indicate that the decision table has not been finished.[*citation needed*] One of the uses of decision tables is to reveal conditions under which certain input factors are irrelevant on the actions to be taken, allowing these input tests to be skipped and thereby streamlining decision-making procedures.[5]

Aside from the basic four quadrant structure, decision tables vary widely in the way the condition alternatives and action entries are represented.[6][7] Some decision tables use simple true/false values to represent the alternatives to a condition (similar to if-then-else), other tables may use numbered alternatives (similar to switch-case), and some tables even use fuzzy logic or probabilistic representations for condition alternatives.[8] In a similar way, action entries can simply represent whether an action is to be performed (check the actions to perform), or in more advanced decision tables, the sequencing of actions to perform (number the actions to perform).

A decision table is considered *balanced* or *complete* if it includes every possible combination of input variables. In other words, balanced decision tables prescribe an action in every situation where the input variables are provided.[4]

While recognizing these advantages, many will point out that tables are merely a systematic way to present static data. Do they have a worthwhile function in a more dynamic situation - that of decision making?

Decision tables may take one of three forms:

(1) Limited entry table

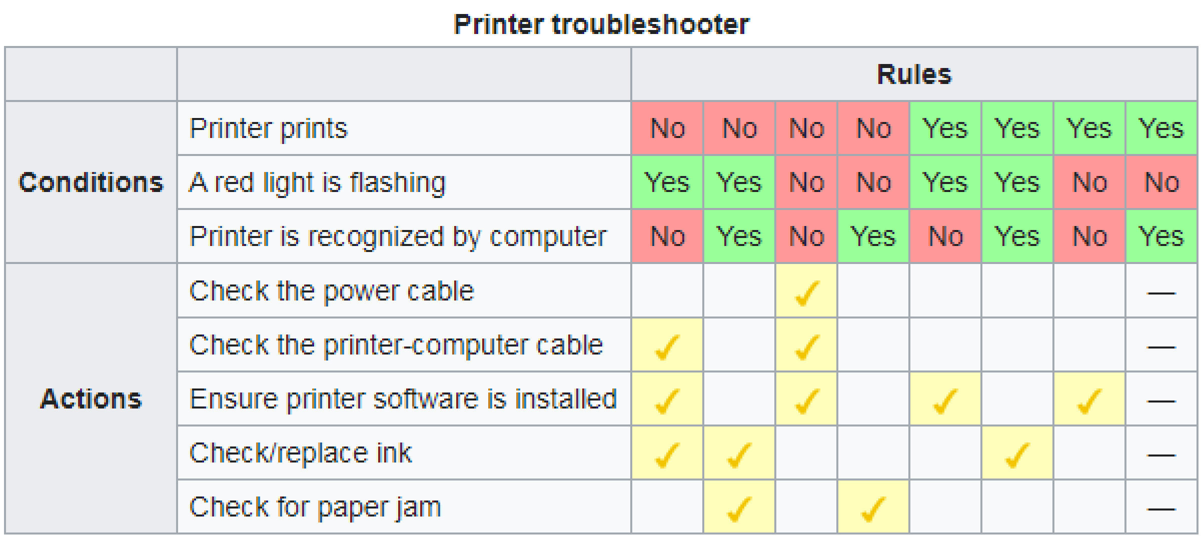
(2) Extended entry table

(3) Mixed entry table

The limited-entry decision table is the simplest to describe. The condition alternatives are simple Boolean values, and the action entries are check-marks, representing which of the actions in a given column are to be performed.

A technical support company writes a decision table to diagnose printer problems based upon symptoms described to them over the phone from their clients.

The following is a balanced decision table.



Printer troubleshooter

		Rules							
Conditions	Printer prints	No	No	No	No	Yes	Yes	Yes	Yes
	A red light is flashing	Yes	Yes	No	No	Yes	Yes	No	No
	Printer is recognized by computer	No	Yes	No	Yes	No	Yes	No	Yes
Actions	Check the power cable			✓					—
	Check the printer-computer cable	✓		✓					—
	Ensure printer software is installed	✓		✓		✓		✓	—
	Check/replace ink	✓	✓				✓		—
	Check for paper jam		✓		✓				—

Sorting Algorithms

Sorting is the process of arranging the elements of an array so that they can be placed either in ascending or descending order. For example, consider an array A = {A1, A2, A3, A4, ?? An }, the array is called to be in ascending order if element of A are arranged like A1 > A2 > A3 > A4 > A5 > ? > An .

Consider an array;

int A[10] = { 5, 4, 10, 2, 30, 45, 34, 14, 18, 9)

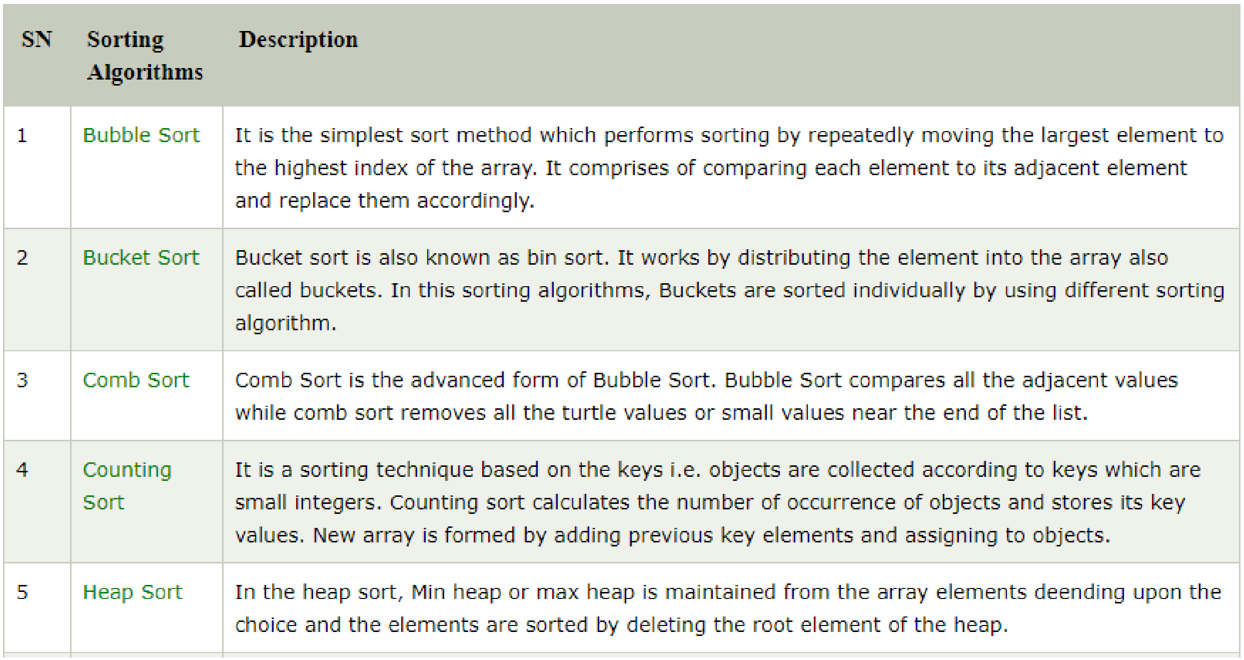
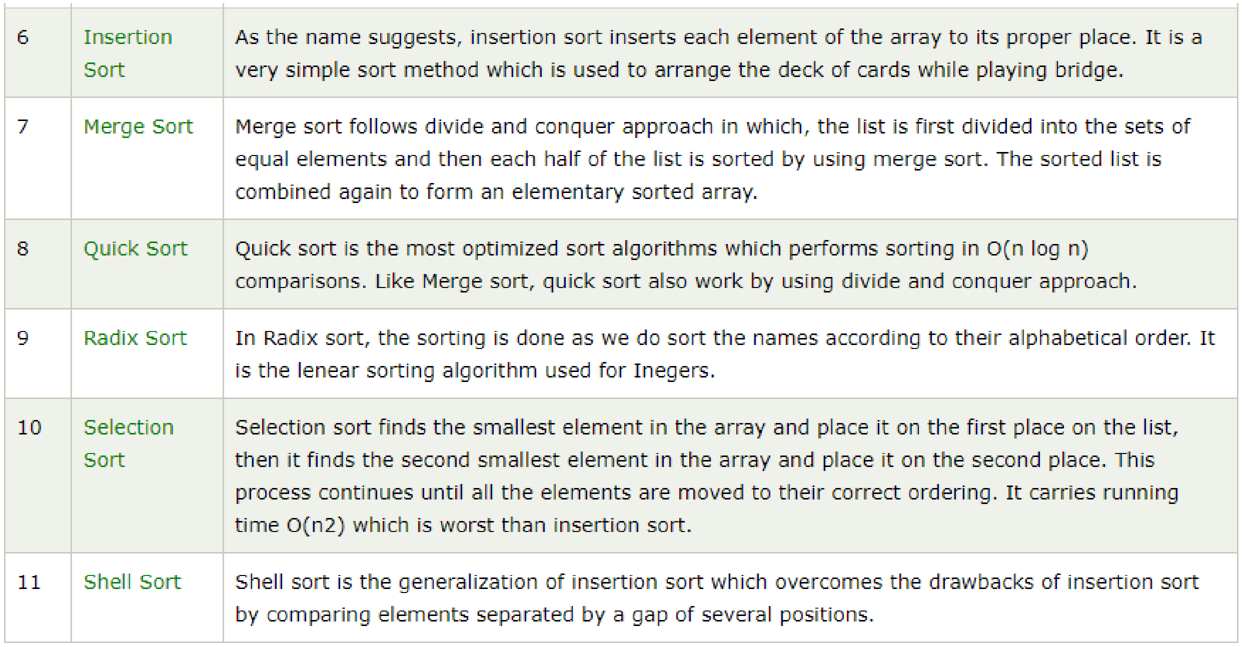
The Array sorted in ascending order will be given as;

A[] = { 2, 4, 5, 9, 10, 14, 18, 30, 34, 45 }

There are many techniques by using which, sorting can be performed. In this section of the tutorial, we will discuss each method in detail.

Sorting Algorithms

Sorting algorithms are described in the following table along with the description.

SN	Sorting Algorithms	Description
1	Bubble Sort	It is the simplest sort method which performs sorting by repeatedly moving the largest element to the highest index of the array. It comprises of comparing each element to its adjacent element and replace them accordingly.
2	Bucket Sort	Bucket sort is also known as bin sort. It works by distributing the element into the array also called buckets. In this sorting algorithms, Buckets are sorted individually by using different sorting algorithm.
3	Comb Sort	Comb Sort is the advanced form of Bubble Sort. Bubble Sort compares all the adjacent values while comb sort removes all the turtle values or small values near the end of the list.
4	Counting Sort	It is a sorting technique based on the keys i.e. objects are collected according to keys which are small integers. Counting sort calculates the number of occurrence of objects and stores its key values. New array is formed by adding previous key elements and assigning to objects.
5	Heap Sort	In the heap sort, Min heap or max heap is maintained from the array elements deending upon the choice and the elements are sorted by deleting the root element of the heap.
6	Insertion Sort	As the name suggests, insertion sort inserts each element of the array to its proper place. It is a very simple sort method which is used to arrange the deck of cards while playing bridge.
7	Merge Sort	Merge sort follows divide and conquer approach in which, the list is first divided into the sets of equal elements and then each half of the list is sorted by using merge sort. The sorted list is combined again to form an elementary sorted array.
8	Quick Sort	Quick sort is the most optimized sort algorithms which performs sorting in O(n log n) comparisons. Like Merge sort, quick sort also work by using divide and conquer approach.
9	Radix Sort	In Radix sort, the sorting is done as we do sort the names according to their alphabetical order. It is the lenear sorting algorithm used for Inegers.
10	Selection Sort	Selection sort finds the smallest element in the array and place it on the first place on the list, then it finds the second smallest element in the array and place it on the second place. This process continues until all the elements are moved to their correct ordering. It carries running time O(n2) which is worst than insertion sort.
11	Shell Sort	Shell sort is the generalization of insertion sort which overcomes the drawbacks of insertion sort by comparing elements separated by a gap of several positions.

Bubble Sort

In Bubble sort, each element of the array is compared with its adjacent element. The algorithm processes the list in passes. A list with n elements requires n-1 passes for sorting. Consider an array A of n elements whose elements are to be sorted by using Bubble sort. The algorithm processes like following.

1. In Pass 1, A[0] is compared with A[1], A[1] is compared with A[2], A[2] is compared with A[3] and so on. At the end of pass 1, the largest element of the list is placed at the highest index of the list.
2. In Pass 2, A[0] is compared with A[1], A[1] is compared with A[2] and so on. At the end of Pass 2 the second largest element of the list is placed at the second highest index of the list.
3. In pass n-1, A[0] is compared with A[1], A[1] is compared with A[2] and so on. At the end of this pass. The smallest element of the list is placed at the first index of the list.

Algorithm:

- **Step 1**: Repeat Step 2 For i = 0 to N-1
- **Step 2**: Repeat For J = i + 1 to N - I
- **Step 3**: IF A[J] > A[i]
  SWAP A[J] and A[i]
  [END OF INNER LOOP]
  [END OF OUTER LOOP
- **Step 4**: EXIT