Data Structures	➢ Scheme for organizing related pieces of information ➢ Actual implementation of a particular abstract data type ➢ Linear Data Structures : In these data structures the elements form a sequence. Such as Arrays, Linked Lists, Stacks and Queues are linear data structures. ➢ Non-Linear Data Structures : In these data structures the elements do not form a sequence. Such as Trees and Graphs are non-linear data structures.
Algorithms	➢ A finite set of instructions that, if followed, accomplishes a paticular task ➢ Manipulate the data in these structures in various ways, such as searching for a particular data item and sorting the data.
Object Oriented Programming	➢ A type of programming in which programmers define not only the data type of a data structure, but also the types of operations (functions) that can be applied to the data structure. ➢ A step-by-step procedure for calculations. ➢ Algorithms are used for calculation, data processing, and automated reasoning. • OBJECT • Objects are just data, with added properties and methods. • Basic unit of object-oriented programming • An Object is a collection of data members and associated member functions also known as methods • An object contains both methods and variables.

	• CLASSES
	  • Data types based on which objects are created.
	  • Class represents a set of individual objects.
	  • A class is a specification—a blueprint—for one or more objects.
	• INHERITANCE
	  • Inheritance is the process of forming a new class from an existing class or base class.
	  • The base class is also known as parent class or super class.
	  • The new class that is formed is called derived class.
	  • Derived class is also known as a child class or sub class. Inheritance helps in reducing the overall code size of the program, which is an important concept in object-oriented programming.
	• INTERFACE
	  • An interface is a contract between a class and the outside world. When a class implements an interface, it promises to provide the behavior published by that interface.
	• PACKAGE
	  • A package is a namespace for organizing classes and interfaces in a logical manner. Placing your code into packages makes large software projects easier to manage.
	• DATA ABSTRACTION
	  • Data Abstraction increases the power of programming language by creating user defined data types.
	  • Data Abstraction also represents the needed information in the program without presenting the details.
	• DATA ENCAPSULATION
	  • Data Encapsulation combines data and functions into a single unit called Class.

	• When using Data Encapsulation, data is not accessed directly; it is only accessible through the functions present inside the class. • Data Encapsulation enables the important concept of data hiding possible. • POLYMORPHISM • Polymorphism allows routines to use variables of different types at different times. • An operator or function can be given different meanings or functions. • Polymorphism refers to a single function or multi-functioning operator performing in different ways. • OVERLOADING • Overloading is one type of Polymorphism. • It allows an object to have different meanings, depending on its context. • When an existing operator or function begins to operate on new data type, or class, it is understood to be overloaded. • REUSABILITY • This term refers to the ability for multiple programmers to use the same written and debugged existing class of data. • This is a time saving device and adds code efficiency to the language. • Additionally, the programmer can incorporate new features to the existing class, further developing the application and allowing users to achieve increased performance. • This time saving feature optimizes code, helps in gaining secured applications and facilitates easier maintenance on the application.
Array	• An *array* is a container object that holds a fixed number of values of a single type. The length of an array is established when the

<table>
<tr><td></td><td>array is created.
Each item in an array is called an element, and each element is accessed by its numerical index.
We can use arrays to help read and analyze repetitive data with a minimum of coding.</td></tr>
<tr><td>Logarithm</td><td>The logarithm of a number is the exponent to which another fixed value, the base, must be raised to produce that number.

e.g. logN</td></tr>
<tr><td>Big O Notation</td><td>Shorthand way to say how efficient a computer algorithm is.

<table>
<tr><td>Sorting</td><td>Best</td></tr>
<tr><td>Quick sort</td><td>O(nlog(n))</td></tr>
<tr><td>Merge sort</td><td>O(nlog(n))</td></tr>
<tr><td>Heap sort</td><td>O(nlog(n))</td></tr>
<tr><td>Bubble sort</td><td>O(n)</td></tr>
<tr><td>Insertion sort</td><td>O(n)</td></tr>
<tr><td>Selection sort</td><td>O(n^2)</td></tr>
<tr><td>Bucket sort</td><td>O(n+k)</td></tr>
<tr><td>Radix</td><td>O(nk)</td></tr>
</table>

<table>
<tr><td>DS</td><td>Indexing</td><td>Search</td><td>Insertion</td><td>Deletion</td></tr>
</table>
</td></tr>
</table>

	Basic Array	O(1)	O(n)	-	-
	Singly Linked List	O(n)	O(n)	O(1)	O(1)
	Doubly Linked List	O(n)	O(n)	O(1)	O(1)
	Hash Table	-	O(1)	O(1)	O(1)
	Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))
	B-Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))
	Red-Black Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))
	AVL Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))

Heaps	Insert	Delete
Linked List Sorted	O(n)	O(1)
Linked List Unsorted	O(1)	O(1)

Bubble Sort	➢ The bubble sort works by iterating down an array to be sorted from the first element to the last, comparing each pair of elements and switching their positions if necessary. ➢ This process is repeated as many times as necessary, until the array is sorted.
Selection Sort	The idea of selection sort is rather simple: we repeatedly find the next largest (or smallest) element in the array and move it to its final position in the sorted array.

Insertion Sort	The main idea of insertion sort is • Start by considering the first two elements of the array data. If found out of order, swap them • Consider the third element; insert it into the proper position among the first three elements. • Consider the fourth element; insert it into the proper position among the first four elements. • … …
Bubble VS Selection VS Insertion	• The **bubble sort** is so simple that you can write it from memory. Even so, it's practical only if the amount of data is small. • The **selection sort** minimizes the number of swaps, but the number of comparisons is still high. This sort might be useful when the amount of data is small and swapping data items is very time-consuming compared with comparing them. • The **insertion sort** is the most versatile of the three and is the best bet in most situations, assuming the amount of data is small or the data is almost sorted. For larger amounts of data, quicksort is generally considered the fastest approach
Stack	• allows access to only one data item: the last item inserted. • also a handy aid for algorithms applied to certain complex data structures. • Placing a data item on the top of the stack is called **pushing** it. • Removing it from the top of the stack is called **popping** it. • A stack is said to be a **Last-In-First-Out (LIFO)** storage mechanism because the last item inserted is the first one to be removed.
Queue	A data structure that is somewhat like a stack, except that in a queue the first item inserted is the first to be removed (First-In-First-Out, FIFO), while in a stack, as we've seen, the last item inserted is the first to be removed (LIFO).

Parsing Arithmetic Expression	➢ Infix : Operators are written in-between their operands. This is the usual way we write expressions. ➢ Postfix : Operators are written after their operands. ➢ Prefix : Operators are written before their operands
Single Linked List	➢ A self referential data structure. ➢ A list of elements, with a head and a tail; each element points to another of its own kind. ➢ Insert – Inserts a new element at the end of the list. ➢ Delete – Deletes any node from the list. ➢ Find – Finds any node in the list. ➢ Print – Prints the list. ➢ Nodes in a linked list are linked together using a next field, which stores the address of the next node in the next field of the previous node i.e. each node of the list refers to its successor and the last node contains the NULL reference. ➢ It has a dynamic size, which can be determined only at run time.
Double Linked List	➢ Each node of the list contain two references (or links) – one to the

	previous node and other to the next node. ➢ The previous link of the first node and the next link of the last node points to NULL. ➢ In comparison to singly-linked list, doubly-linked list requires handling of more pointers but less information is required as one can use the previous links to observe the preceding element. ➢ It has a dynamic size, which can be determined only at run time.
Double Ended Linked Lists	➢ Has one additional reference (tail) to the last node. ➢ With additional link to the last node, a double ended linked list is able to insert a new node directly at the end of the list without finding the last node by iterating through the entire list as in a single-ended list.
Traverse a Linked List	• To traverse a linked list, you start at first and then go from link to link, using each link's next field to find the next link. • A link with a specified key value can be found by traversing the list. Once found, an item can be displayed, deleted, or operated on in other ways. • A new link can be inserted before or after a link with a specified key value, following a traversal to find this link.

Recursion	➢ Recursion is a programming technique in which a method (function) calls itself. • It calls itself. • When it calls itself, it does so to solve a smaller problem. • There's some version of the problem that is simple enough that the routine can solve it, and return, without calling itself. ➢ Recursion is usually used because it simplifies a problem conceptually, not because it's inherently more efficient. ➢ Recursion is the programming equivalent of mathematical induction. Mathematical induction is a way of defining something in terms of itself. (The term is also used to describe a related approach to proving theorems.) ➢ Recursion for factorial ➢ Recursion for anagram ➢ Anagram, Suppose you want to list all the anagrams of a specified word—that is, all possible permutations (whether they make a real English word or not) that can be made from the letters of the original word. ➢ Permutation, A permutation is an arrangement of things in a definite order. ➢ Rotate, means to shift all the letters one position left, except

	for the leftmost letter (for Anagram) ➢ Recursion for Binary Search, You divide the big problem into two smaller problems and solve each one separately . In the binary search, there are two such calls, but only one of them is actually executed. ➢ Divide and Conquer Approach, A divide-and-conquer approach usually involves a method that contains two recursive calls to itself, one for each half of the problem. ➢ Mergesort, executes both recursive calls (to sort two halves of an array). ➢ Towers of Hanoi, In our Towers of Hanoi solution, we recurse on the largest disk to be moved. That is, we will write a recursive function that takes as a parameter the disk that is the largest disk in the tower we want to move. Our function will also take three parameters indicating from which peg the tower should be moved (*source*), to which peg it should go (*dest*), and the other peg, which we can use temporarily to make this happen (*spare*).
Shell Sort	The idea of Shellsort is the following: 1. arrange the data sequence in a two-dimensional array 2. sort the columns of the array Actually, the data sequence is not arranged in a two-dimensional array, but held in a one-dimensional array that is indexed appropriately. For instance, data elements at positions 0, 5, 10, 15 etc. would form the first

	column of an array with 5 columns. The "columns" obtained by indexing in this way are sorted with Insertion Sort, since this method has a good performance with presorted sequences. Best Case when Sorted → O(nlogn) Unsorted → O(n^2)
Quick Sort	1. Pick one element in the array, which will be the *pivot*. 2. Make one pass through the array, called a *partition* step, re-arranging the entries so that: • the pivot is in its proper place. • entries smaller than the pivot are to the left of the pivot. • entries larger than the pivot are to its right. 3. Recursively apply quicksort to the part of the array that is to the left of the pivot, and to the right part of the array. Best Case : O(NlogN)
Radix Sort	It functions by sorting the input numbers on each digit, for each of the digits in the numbers. However, the process adopted by this sort method is somewhat counterintuitive, in the sense that the numbers are sorted on the least-significant digit first, followed by the second-least significant digit and so on till the most significant digit.
Binary Tree	➢ A binary tree is made of nodes, where each node contains a "left" reference, a "right" reference, and a data element. ➢ The topmost node in the tree is called the root. ➢ Every node (excluding a root) in a tree is connected by a directed edge from exactly one other node. This node is called a parent. ➢ On the other hand, each node can be connected to arbitrary

	number of nodes, called children. ➢ Nodes with no children are called leaves, or external nodes. ➢ Nodes which are not leaves are called internal nodes. ➢ Nodes with the same parent are called siblings. ➢ A full binary tree.is a binary tree in which each node has exactly zero or two children. ➢ A complete binary tree is a binary tree, which is completely filled, with the possible exception of the bottom level, which is filled from left to right. ➢ A traversal is a process that visits all the nodes in the tree. Since a tree is a nonlinear data structure, there is no unique traversal. We will consider several traversal algorithms with we group in the following two kinds • depth-first traversal • breadth-first traversal There are three different types of depth-first traversals, : • PreOrder traversal - visit the parent first and then left and right children; • InOrder traversal - visit the left child, then the parent and the right child; • PostOrder traversal - visit left child, then the right child and then the parent; There is only one kind of breadth-first traversal--the level order traversal. This traversal visits nodes by levels from top to bottom and from left to

	right.
Binary Search Trees	BST is a binary tree where nodes are ordered in the following way: • each node contains one key (also known as data) • the keys in the left subtree are less then the key in its parent node, in short L < P; • the keys in the right subtree are greater the key in its parent node, in short P < R; • duplicate keys are not allowed. The insertion procedure is quite similar to searching. We start at the root and recursively go down the tree searching for a location in a BST to insert a new node. Searching in a BST always starts at the root. We compare a data stored at the root with the key we are searching for (let us call it as `toSearch`). If the node does not contain the key we proceed either to the left or right child depending upon comparison. If the result of comparison is negative we go to the left child, otherwise - to the right child. The recursive structure of a BST yields a recursive algorithm. Deletion starategy is the following: replace the node being deleted with the largest node in the left subtree and then delete that largest node. By symmetry, the node being deleted can be swapped with the smallest node is the right subtree.
Red Black Tree	A *red-black tree* is a binary search tree with one extra attribute for each node: the *colour*, which is either red or black. We also need to keep track of the parent of each node.

A red-black tree is a binary search tree which has the following *red-black properties*:

1. Every node is either red or black.
2. Every leaf (NULL) is black.
3. If a node is red, then both its children are black.
4. Every simple path from a node to a descendant leaf contains the same number of black nodes.

3. implies that on any path from the root to a leaf, red nodes must not be adjacent.
   However, any number of black nodes may appear in a sequence.

This demonstrates why the red-black tree is a good search tree: it can always be searched in **O(log n)** time.

A rotation is a local operation in a search tree that preserves *in-order* traversal key ordering.

Insertion is somewhat complex and involves a number of cases. Note that we start by inserting the new node, `x`, in the tree just as we would for any other binary tree, using the `tree_insert` function. This new node is labelled red, and possibly destroys the red-black property. The main loop moves up the tree, restoring the red-black property.

Trees which remain **balanced** - and thus guarantee **O(logn)** search times - in a dynamic environment. Or more importantly, since any tree can be re-balanced - but at considerable cost - can be re-balanced in **O(logn)** time.

B- Trees	➔ A B-tree is a specialized multiway tree designed especially for use on disk. ➔ In a B-tree each node may contain a large number of keys. ➔ The number of subtrees of each node, then, may also be large. ➔ A B-tree is designed to branch out in this large number of directions and to contain a lot of keys in each node so that the height of the tree is relatively small. ➔ This means that only a small number of nodes must be read from disk to retrieve an item. ➔ The goal is to get fast access to the data, and with disk drives this means reading a very small number of records. ➔ Note that a large node size (with lots of keys in the node) also fits with the fact that with a disk drive one can usually read a fair amount of data at once. ➔ If instead this leaf node is full so that there is no room to add the new item, then the node must be "split" with about half of the keys going into a new node to the right of this one. The median (middle) key is moved up into the parent node. (Of course, if that node has no room, then it may have to be split as well.) ➔
Difference (2-3 vs 2-3-4	The 2, 3, and 4 in the name 2-3-4 tree refer to how many links to child nodes can potentially be contained in a given node. For non-leaf nodes,

trees)	three arrangements are possible: • A node with one data item always has two children. • A node with two data items always has three children. • A node with three data items always has four children. In short, a non-leaf node must always have one more child than it has data items. 2-3 trees are similar to 2-3-4 trees except that, as you might have guessed from the name, they hold one less data item and have one less child. They were the first multiway tree, invented by J. E. Hopcroft in 1970. B-trees (of which the 2-3-4 tree is a special case) were not invented until 1972. In many respects the operation of 2-3 trees is similar to that of 2-3-4 trees. Nodes can hold one or two data items and can have zero, one, two, or three children. Otherwise, the arrangement of the key values of the parent and its children is the same. Inserting a data item into a node is potentially simplified because fewer comparisons and moves are potentially necessary. As in 2-3-4 trees, all insertions are made into leaf nodes, and all leaf nodes are on the bottom level.
Hash Tables	➔ Hashing is the transformation of a string of characters into a usually shorter fixed-length value or key that represents the

	original string. ➔ Hashing is used to index and retrieve items in a database because it is faster to find the item using the shorter hashed key than to find it using the original value. It is also used in many encryption algorithms. ➔ A hash function takes an item of a given type and generates an integer hash value within a given range. The input items can be anything: strings, compiled shader programs, files, even directories. The same input always generates the same hash value, and a good hash function tends to generate different hash values when given different inputs. ➔ In computer science, a **collision** or **clash** is a situation that occurs when two distinct pieces of data have the same hash value, checksum, fingerprint, or cryptographic digest. ➔ Any collision in a hash table increases the average cost of lookup operations. When fingerprints are used to avoid unnecessary file storage or transfer, e.g. in a proxy server or backup system, a collision may cause incorrect operation and even permanent data loss. A successful collision attack on a cryptographic hash function may compromise the security of computer and communication systems. ➔ **You hash the word to obtain its index number but find that the cell at that number is already occupied by the word demystify, which happens to hash to the exact same number (for a certain

size array).

- ✓ One approach, when a collision occurs, is to search the array in some systematic way for an empty cell and insert the new item there,

instead of at the index specified by the hash function. This approach is called **open addressing.**

- ✓ This is called **linear probing** because it steps sequentially along the line of cells.
- ✓ The number of steps they take is the **probe length.**
- ✓ **Clustering** can result in very long probe lengths. This means that accessing cells at the end of the sequence is very slow.
- ✓ One option when a hash table becomes too full is to **expand its array.** You'll need to go through the old array in sequence, cell by cell, inserting each item you find into the new array . This is

called **rehashing .**

- ✓ The ratio of the number of items in a table to the table's size is called the **load factor.**
- ✓ **Quadratic probing** is an attempt to keep clusters from forming. The idea is to probe more widely separated cells,

	instead of those adjacent to the primary hash site. ✓ To eliminate secondary clustering as well as primary clustering, we can use another approach: **double hashing.** ✓ In **open addressing**, collisions are resolved by looking for an open cell in the hash table. ✓ A second approach (mentioned earlier) is to create an array that consists of linked lists of words instead of the words themselves. Then, when a collision occurs, the new item is simply inserted in the list at that index. This is called **separate chaining.** ✓ For each table address, a linked list of the records whose keys hash to that address, is built. This method is useful for highly dynamic situations, where the number of the search keys cannot be predicted in advance.
Heaps	A heap is a binary tree with these characteristics: • It's complete. This means it's completely filled in, reading from left to right across each row, although the last row need not be full. • It's (usually) implemented as an array. We described in Chapter 8, "Binary Trees," how binary trees can be stored in arrays, rather than using

	references to connect the nodes. • Each node in a heap satisfies the heap condition, which states that every node's key is larger than (or equal to) the keys of its children. ➢ A priority queue based on such a heap is a **descending-priority queue.** ➢ A heap is weakly ordered compared with a binary search tree, in which all a node's left descendants have keys less than all its right descendants. ➢ Removal means removing the node with the maximum key. ➢ Inserting a node is also easy. Insertion uses trickle up, rather than trickle down.
Heapsort	Heapsort is an efficient sorting procedure that requires O(N*logN) time. Heapsort can be made to run faster by applying the trickle-down algorithm directly to N/2 items in the unsorted array, rather than inserting N items.

