St. Peter's Engineering College (Autonomous) Dullapally (P), Medchal, Hyderabad – 500100. **QUESTION BANK**					Dept.	:	CSE, CSM, CSD, CSC
					Academic Year 2023-24		
Subject Code	:	AS22-05ES07	Subject	:	Data Structures		
Class/Section	:	B.Tech.	Year	:	I	Semester :	II

BLOOMS LEVEL					
Remember	L1	Understand	L2	Apply	L3
Analyze	L4	Evaluate	L5	Create	L6

*****

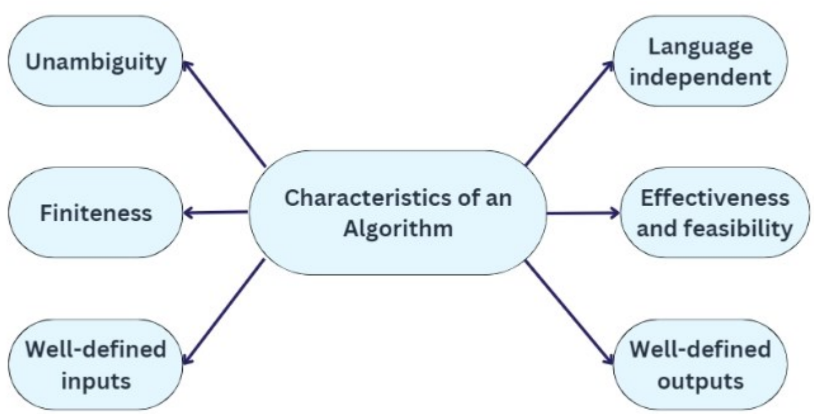
Q. No	Question (s)	Marks	BL	CO
	UNIT - I			
1	**a) What is the definition of a data structure?**	**1M**	**L1**	**C123.1**
	Data Structure can be defined as the group of data elements which provides an efficient way of storing and organizing data in the computer. Data structures allow programs to store and process data effectively.			
	b) Define a linear data structure and provide an example.	**1M**	**L2**	**C123.1**
	Linear Data Structure consists of data elements arranged in a sequential manner where every element is connected to its previous and next elements. Some Examples of Linear Data Structure are Arrays, Linked Lists, Stacks & Queues.			
	c) Define a non-linear data structure and provide an example.	**1M**	**L2**	**C123.1**
	Non-linear Data Structures do not have any set sequence of connecting all its elements and every element can have multiple paths to attach to other elements. Some Examples of non-linear data structures are Tree & Graphs.			
	d) What is performance analysis of an algorithm?	**1M**	**L2**	**C123.1**
	Performance of an algorithm means predicting the resources which are required to an algorithm to perform its task. Performance analysis of an algorithm is the process of calculating space and time required by that algorithm.			
	e) Define the Big Theta Notation	**1M**	**L2**	**C123.1**
	Big - Theta notation always indicates the average time required by an algorithm for all input values, it means describes the average case time complexity of an algorithm.			

2	a) Differentiate between static and dynamic data structures	3M	L2	C123.1

Aspect	Static Data Structure	Dynamic Data Structure
Memory allocation	Memory is allocated at compile-time	Memory is allocated at run-time
Size	Size is fixed and cannot be modified	Size can be modified during runtime
Memory utilization	Memory utilization may be inefficient	Memory utilization is efficient as memory can be reused
Access	Access time is faster as it is fixed	Access time may be slower due to indexing and pointer usage
Examples	Arrays, Stacks, Queues, Trees (with fixed size)	Lists, Trees (with variable size), Hash tables

	b) Elaborate on the characteristics that define an algorithm	3M	L2	C123.1

An algorithm is a sequence of unambiguous instructions and is used to convert our problem solution into step-by-step statements. Here, the program takes required data as input, processes data according to the program instructions and finally produces a result.



Every algorithm must satisfy the following characteristics
1. Input - Every algorithm must take zero or more number of input values from external.
2. Output - Every algorithm must produce an output as result.
3. Definiteness - Every statement/instruction in an algorithm must be clear and unambiguous (only one interpretation).
4. Finiteness - For all different cases, the algorithm must produce result within a finite number of steps.
5. Effectiveness - Every instruction must be basic enough to be carried out and it also must be feasible

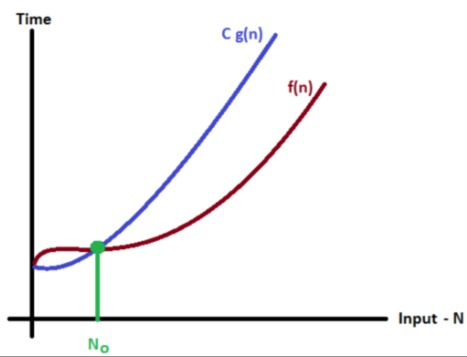
	c) List the advantages of using data structures	3M	L2	C123.1

The Advantages of data structures are
1. Data structures allow storing the information on hard disks.

	2. An appropriate choice of ADT (Abstract Data Type) makes the program more efficient. 3. Data Structures are necessary for designing efficient algorithms. 4. It provides reusability and abstraction. 5. Using appropriate data structures can help programmers save a good amount of time while performing operations such as storage, retrieval, or processing of data. 6. Manipulation of large amounts of data is easier.

	d) Define Big Oh Notation and provide a brief explanation	**3M**	**L2**	**C123.1**

Big - Oh notation is used to define the upper bound of an algorithm in terms of Time Complexity.
Big - Oh notation always indicates the maximum time required by an algorithm for all input values, it means describes the worst-case time complexity of an algorithm.
Let us Consider function $f(n)=O(g(n))$, if there exist a positive constants c and n_o.
Such that $f(n) <= C\ g(n)$ for all $n >= n_0$.
Example: - Consider the following f(n) and g(n)
$f(n) = 3n + 2$
$g(n) = n$
$f(n) <= C\ g(n)$
$3n + 2 <= 5\ n$
Therefore, $f(n)=O(n)$



	e) Enumerate the differences between Linear and Binary search	**3M**	**L3**	**C123.1**

Parameters	Linear Search	Binary Search
Definition	Linear Search sequentially checks each element in the list until it finds a match or exhausts the list.	Binary Search continuously divides the sorted list, comparing the middle element with the target value.
Time Complexity	The time complexity is O(n), where n is the number of elements in the list.	The time complexity is O(log n), making it faster for larger datasets.
Efficiency	Less efficient, especially for large datasets.	More efficient, especially for large datasets.
Data Requirement	Does not require the list to be sorted.	Requires the list to be sorted.
Implementation	Easier to implement.	Requires a more complex implementation.
Search Space	Examines each element sequentially.	Eliminates half of the search space with each comparison.
Use Case	Suitable for small and unsorted datasets.	Ideal for large and sorted datasets.

3	a) Differentiate between linear and non-linear data structures	5M	L3	C123.1

Parameter	Linear Data Structure	Non-Linear Data Structure
Arrangement of Data Element	In a linear data structure, the data elements connect to each other sequentially. A user can transverse each element through a single run.	In a non-linear data structure, the data elements connect to each other hierarchically. Thus, they are present at various levels.
Complexity of Implementation	The linear data structures are comparatively easier to implement.	The non-linear data structures are comparatively difficult to implement and understand as compared to the linear data structures.
Levels	A user can find all of the data elements at a single level in a linear data structure.	One can find all the data elements at multiple levels in a non-linear data structure.
Traversal	You can traverse a linear data structure in a single run.	It is not easy to traverse the non-linear data structures. The users need multiple runs to traverse them completely.
Utilization of Memory	It is not very memory-friendly. It means that the linear data structures can't utilize memory very efficiently.	The data structure is memory-friendly. It means that it uses memory very efficiently.
Complexity of Time	The time complexity of this data structure is directly proportional to its size. It means that the time complexity increases with increasing input size.	Non-linear data structure's time complexity often remains the same with an increase in its input size.
Applications	Linear data structures work well mainly in the development of application software.	Non-linear data structures work mainly well in image processing and Artificial Intelligence.
Examples	List, Array, Stack, Queue.	Map, Graph, Tree.

	b) Describe the common operations performed on data structures	**5M**	**L3**	**C123.1**

The common operations that can be performed on the data structures are as follows:

- Searching – We can easily search for any data element in a data structure.
- Sorting – We can sort the elements either in ascending or descending order.
- Insertion – We can insert new data elements in the data structure.
- Deletion – We can delete the data elements from the data structure.
- Updation – We can update or replace the existing elements from the data structure.

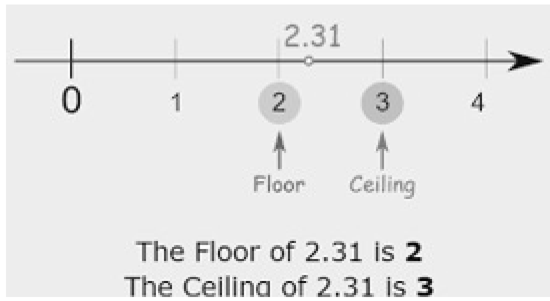
	c) Explain in detail about Mathematical Functions and Notations	**5M**	**L3**	**C123.1**

The Mathematical functions and notations are as follows

a) Floor and Ceiling Functions:

The ceiling function returns the smallest nearest integer which is greater than or equal to the specified number.

The floor function returns the largest nearest integer which is less than or equal to a specified value.



b) Remainder Function or Modular Arithmetic:

Modular arithmetic is a system of arithmetic for integers, which considers the remainder. The integer remainder is obtained where some K is divided by M.

Example: - K (Mod M) = 25(Mod 7) = 4.

c) Integer and Absolute value functions

The integer function will convert ‘x’ into integer and the fractional part is removed.

Example: - INT (3.14) = 3

The absolute value | x | of a real number x is the non-negative value of x without regard to its sign.

Example: - ABS (-5) = 5

d) Summation Function

The Greek capital letter, ∑, is used to represent the summation.

Example: - The series 4+8+12+16+20+24 can be expressed as $6\sum n=14n$. The variable n is called the index of summation.

e) Factorial Function:

The product of the positive integers from 1 to n, inclusive is denoted by n!

N! = 1.2.3…. (n-2) (n-1) n

Example: - 5! = 1 X 2 X 3 X 4 X 5=120

f) Permutations:

Any arrangement of a set of n objects in a given order is called Permutation of Object.

Example: - Suppose the set contains a,b&c. the various permutations of these elements are abc, acb, bac, bca, cab & cba(Total 6 Permutations).

g) Exponents and Logarithms Funtions

Exponent means how many times a number is multiplied by itself.

Example: - 2^4=2 X 2 X 2 X 2 =16

A logarithm is an exponent which indicates to what power a base must be raised to produce a given number.

Example:- $x = \log_b y$ (x is the logarithm of y to the base b).

	d) Explain the concept of time complexity and provide examples of algorithms with different time complexities	**5M**	**L4**	**C123.1**

Time Complexity: -

The time complexity of an algorithm is the total amount of time required by an algorithm to complete its execution.

Calculating Time Complexity of an algorithm based on the system configuration is a very difficult task because the configuration changes from one system to another system. To solve this problem, we must assume a model machine with a specific configuration. So that, we can able to calculate generalized time complexity according to that model machine.

Let us assume a machine with following configuration as a Single processor machine having 32-bit operating system and it performs sequential executions, it will take one unit of time for arithmetic and logical operations, for read and write operations and for returning the value.

There are two types of time complexities.

1. Constant time complexity
2. Linear time complexity

Constant time complexity: -

If any program requires a fixed amount of time for all input values, then its time complexity is said to be Constant Time Complexity.

Example:- 1

```
int sum(int a, int b)
{
    return a+b;
}
```

It requires 1 unit of time to calculate a+b and 1 unit of time to return the value. That means, totally it takes 2 units of time to complete its execution. And it does not change based on the input values of a and b. That means for all input values, it requires the same amount of time i.e. 2 units.

Linear time complexity: -

If the amount of time required by an algorithm is increased with the increase of input value then that time complexity is said to be Linear Time Complexity.

Example: -2

	Cost Time require for line (Units)	Repeatation No. of Times Executed	Total Total Time required in worst case
int sumOfList(int A[ ], int n)			
{			
int sum = 0, i;	1	1	1
for(i = 0; i < n; i++)	1 + 1 + 1	1 + (n+1) + n	2n + 2
sum = sum + A[i];	2	n	2n
return sum;	1	1	1
}			
			4n + 4 Total Time required

Totally it takes '4n+4' units of time to complete its execution. Here the exact time is not fixed. And it changes based on the n value. If we increase the n value then the time required also increases linearly.

	e) Explain the concept of space complexity and provide examples of algorithms with different space complexities	**5M**	**L4**	**C123.1**

Space Complexity: -
Total amount of computer memory required by an algorithm to complete its execution is called as space complexity of that algorithm.

To calculate the space complexity, we must know the memory required to store different datatype values. For example, the C Programming Language compiler requires the following

2 bytes to store Integer value.
4 bytes to store Floating Point value.
1 byte to store Character value.
6 or 8 bytes to store double value.

There are two types of space complexities.

1. Constant space complexity
2. Linear space complexity

Constant space complexity: -
If any algorithm requires a fixed amount of space for all input values then that space complexity is said to be Constant Space Complexity.

Example: -1

```
int square(int a)
{
    return a*a;
}
```

It requires 2 bytes of memory to store variable 'a' and another 2 bytes of memory is used for return value. That means, totally it requires 4 bytes of memory to complete its execution and these 4 bytes of memory is fixed for any input value of 'a'.

Linear space complexity: -
If the amount of space required by an algorithm is increased with the increase of input value, then that space complexity is said to be Linear Space Complexity.

Example: -2

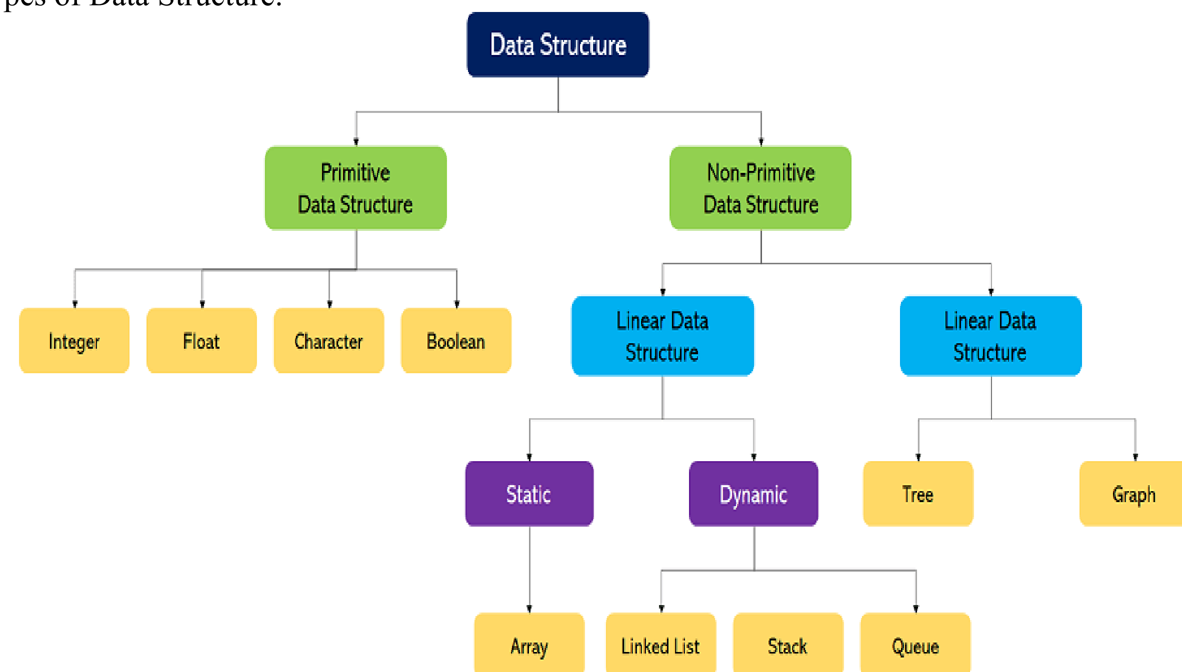
```
int sum(int A[ ], int n)
{
      int sum = 0, i;
      for(i = 0; i < n; i++)
          sum = sum + a[i];
      return sum;
}
```

It requires '2*n' bytes of memory to store array variable 'a[ ]', 6 bytes of memory for integer parameter 'n', ‘sum’ and ‘i’ variables(2Bytes for each) and for return value it will take 2 Bytes of space.That means, totally it requires '2n+8' bytes of memory to complete its execution. Here, the total amount of memory required depends on the value of 'n'. As 'n' value increases the space required also increases proportionately

4	a) What is a data structure? What are the types of data structures and what advantages do they offer?	10M	L2	C123.1

Data Structure can be defined as the group of data elements which provides an efficient way of storing and organizing data in the memory. Data structures allow programs to store and process data effectively. There are many different data structures, each having its own advantages and disadvantages.

Types of Data Structure: -



Primitive Data Structure –

Primitive Data Structures directly operate according to the machine instructions. The primitive or base data types are the building blocks of data structures. The typical base data types are int, char, float and boolean etc..

Non – Primitive Data Structure –

Non-primitive data structures are complex data structures that are derived from primitive data structures. Non – Primitive data types are further divided into two categories.

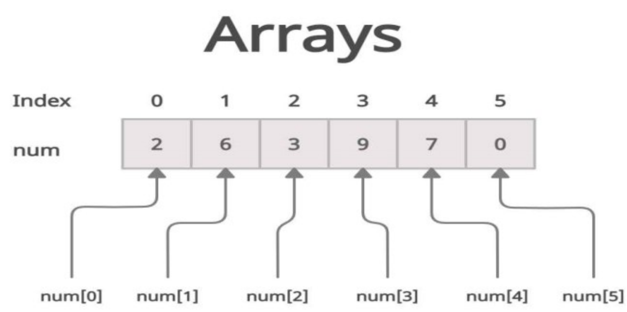
- Linear Data Structure
- Non – Linear Data Structure

Linear Data Structure –

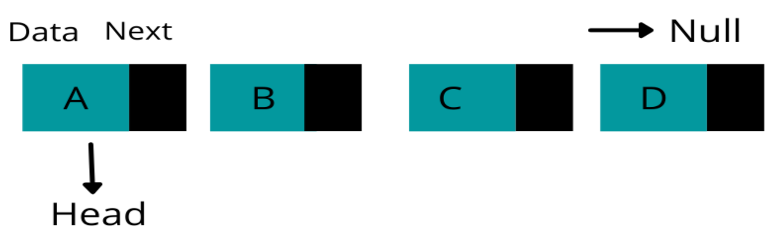
Linear Data Structure consists of data elements arranged in a sequential manner where every element is connected to its previous and next elements. This connection helps to traverse a linear arrangement in a single level and in a single run.

Types of Linear Data Structure

1] Arrays
An array is a collection of similar data elements stored at contiguous memory locations. It is the simplest data structure where each data element can be accessed directly by only using its index number.

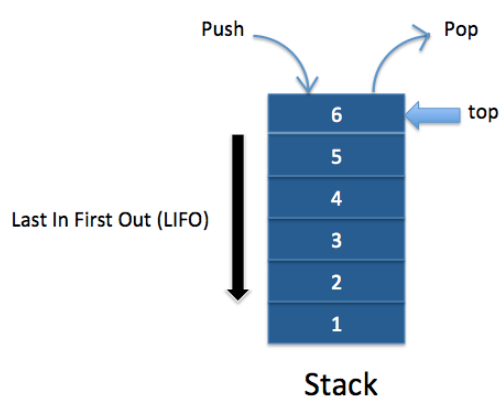


2] Linked List
A linked list is a linear data structure that is used to maintain a list-like structure in the computer memory. It is a group of nodes that are not stored at contiguous locations. Each node of the list is linked to its adjacent node with the help of pointers.



3] Stack
Stack is a linear data structure that follows a specific order during which the operations are performed. The order could be FILO (First In Last Out) or LIFO (Last In First Out).
The basic operations performed in stack are as follows:

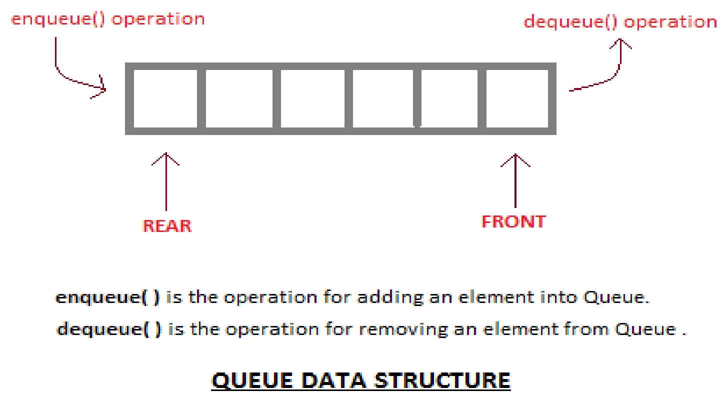
- Push – Adds an item within the stack.
- Pop – Deletes or removes an item from the stack.
- Top – Returns the topmost element of the stack.



4] Queue –
Queue is a linear data structure in which elements can be inserted from only one end which is known as rear and deleted from another end known as front. It follows the FIFO (First In First Out)

order.

- Deque – Adds an element to the queue.
- Enqueue – Deletes or removes an element from the queue.



Non-Linear Data Structure

Non-linear Data Structures do not have any set sequence of connecting all its elements and every element can have multiple paths to attach to other elements. Such data structures support multi-level storage and sometimes can't be traversed in a single run.

Types of Non-Linear Data Structure

1] Tree –

A tree is a multilevel data structure defined as a set of nodes. The topmost node is named root node while the bottom most nodes are called leaf nodes. Each node has only one parent but can have multiple children.

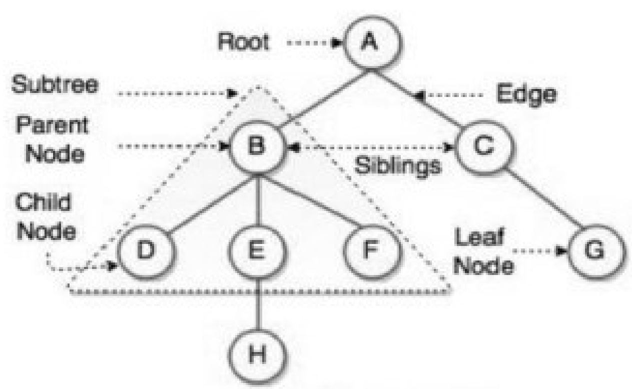
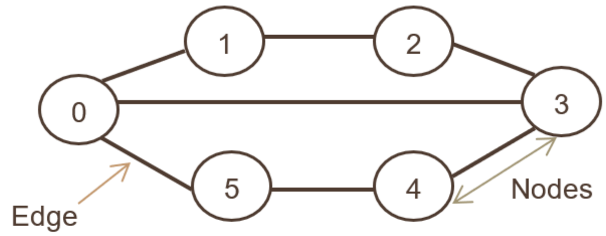


Fig. Structure of Tree

2] Graph

A graph is a pictorial representation of a set of objects connected by links known as edges. The interconnected nodes are represented by points named vertices, and the links that connect the vertices are called edges.



Advantages of Data Structure –

1. Data structures allow storing the information on hard disks.
2. An appropriate choice of ADT (Abstract Data Type) makes the program more efficient.
3. Data Structures are necessary for designing efficient algorithms.
4. It provides reusability and abstraction.
5. Using appropriate data structures can help programmers save a good amount of time while performing operations such as storage, retrieval, or processing of data.
6. Manipulation of large amounts of data is easier.

b) Analyze the time and space complexity of linear and binary searching algorithms in detail.	**10M**	**L3**	**C123.1**

Linear Search Algorithm

Linear search algorithm is also known as sequential search algorithm. This search process starts comparing search element with the first element in the list. If both are matched then result is element found otherwise search element is compared with the next element in the list.

Repeat the same until search element is compared with the last element in the list, if that last element also doesn't match, then the result is "Element not found in the list". That means, the search element is compared with element by element in the list.

In our worst-case scenario, this is not very efficient. We have to check every single number in the list until we get to our answer.

The Big O notation for Linear Search is **O(n)**. The complexity is directly related to the size of the inputs.

Space and Time Complexity of Linear Search

Time Complexity (best)	O(1)
Time Complexity (average)	O(n)
Time Complexity (worst)	O(n)
Space Complexity	O(1)

Binary Search Algorithm

The binary search algorithm can be used with only a sorted list of elements. The binary search cannot be used for a list of elements arranged in random order. This search process starts comparing the search element with the middle element in the list. If both are matched, then the result is "element found". Otherwise, we check whether the search element is smaller or larger than the middle element in the list.

If the search element is smaller, then we repeat the same process for the left sublist of the middle element. If the search element is larger, then we repeat the same process for the right sublist of the middle element.

We repeat this process until we find the search element in the list or until we left with a sublist of only one element. And if that element also doesn't match with the search element, then the result is "Element not found in the list".

It is efficient because it eliminates half of the list in each pass. Binary search algorithm finds a given element in a list of elements with **O(log n)** time complexity where n is total number of elements in the list. O(log n) means that the algorithm takes an additional step each time the data doubles.

Example: - Let us consider the algorithms divides K times into equal halves then

$$\frac{n}{2^K}=1$$
$$2^K=n$$
$$K=\log n$$

Space and Time Complexity of Binary Search

Time complexity (best)	O(1)
Time complexity (average)	O(log(n))
Time complexity (worst)	O(log(n))
Space complexity	O(1)

c) Explain in detail about Asymptotic Analysis and Asymptotic Notations, and discuss their significance in algorithmic analysis	**10M**	**L4**	**C123.1**

Asymptotic Notations are mathematical tools to represent complexity of algorithm for asymptotic analysis. In Asymptotic analysis, we can evaluate the performance of an algorithm in terms of input size.

Types of Asymptotic analysis is

1. Best Case − Minimum time required for program execution.
2. Average Case − Average time required for program execution.
3. Worst Case − Maximum time required for program execution.

Asymptotic notation of an algorithm is a mathematical representation of its complexity.

For example, consider the time complexities of an algorithms is **$5n^2 + 2n + 1$,**
Generally, when we analyze an algorithm, we consider the time complexity for larger values of input data. For the above complexity the **$5n^2$** is most significant term because it is larger value of 'n'. we ignore the least significant terms to represent overall time required by an algorithm. Therefore, the order of a function is **$O(n^2)$.**

Different types of asymptotic notations are used to represent the running time complexity of an algorithm. Most commonly used asymptotic notations are

1. Big Oh Notation
2. Big omega Notation
3. Big theta Notation

1. Big - Oh Notation (O):

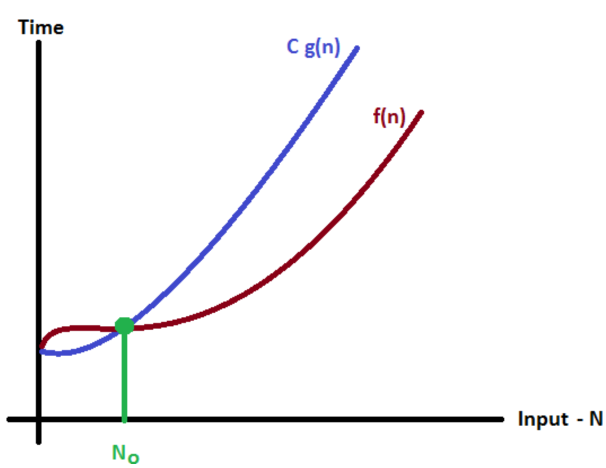
Big - Oh notation is used to define the upper bound of an algorithm in terms of Time Complexity. Big - Oh notation always indicates the maximum time required by an algorithm for all input values and it describes the worst-case time complexity of an algorithm.

Consider function f(n)=O(g(n)), If and only if there exists two positive integers c and n_0 Such that f(n) <= C g(n) for all n >= n_0.

Example
Consider the following f(n) = 2n + 3

$$f(n) <= C\ g(n)$$
$$\Rightarrow 3n + 2 <= 5\ n$$

Above condition is always TRUE for all values of C = 5 and n >= 1
By using Big - Oh notation we can represent the time complexity for 3n + 2 is O(n).



2. Big - Omega Notation (Ω):

Big - omega notation is used to define the lower bound of an algorithm in terms of Time Complexity. Big - omega notation always indicates the minimum time required by an algorithm for all input values and it describes the best-case time complexity of an algorithm.

Consider function f(n)= Ω (g(n)), If and only if there exists two positive integers c and n_0 Such that f(n) >= C g(n) for all n >= n_0.

Example

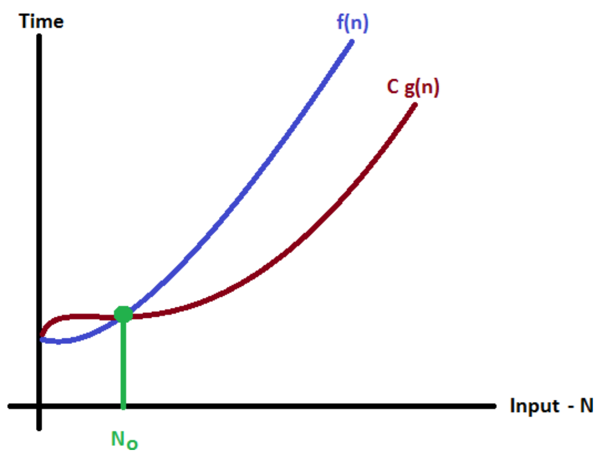
Consider the following f(n) = 2n + 3

$$f(n) >= C\ g(n)$$

$$\Rightarrow 3n + 2 >= 1\ n$$

Above condition is always TRUE for all values of C = 1 and n >= 1

By using Big - Omega notation we can represent the time complexity for 3n + 2 is Ω (n).



3. Big - Theta Notation (Θ):

Big - Theta notation is used to define the Average bound of an algorithm in terms of Time Complexity. Big - Theta notation always indicates the average time required by an algorithm for all input values and it describes the average-case time complexity of an algorithm.

Consider function f(n)= Θ (g(n)), If and only if there exists two positive integers c and n_0 Such that C_1 g(n) <=f(n) <= C_2 g(n) for all n >= n_0.

Example

Consider the following f(n) = 2n + 3

$$C_1\ g(n) <= f(n) <= C_2\ g(n)$$

$$\Rightarrow \quad 1\ n \quad <= 3n + 2 <= 5\ n$$

Above condition is always TRUE for all values of C_1 = 1 and C_2=5 for all n >= 1

By using Big - Theta notation we can represent the time complexity for 3n + 2 is Θ (n).

SR 22