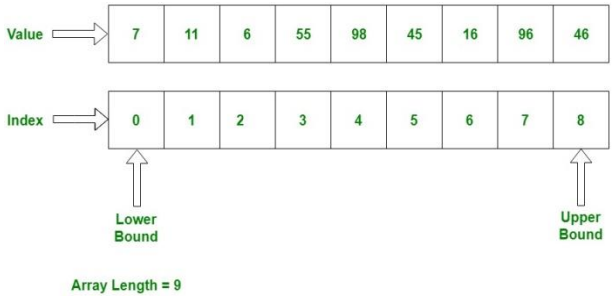
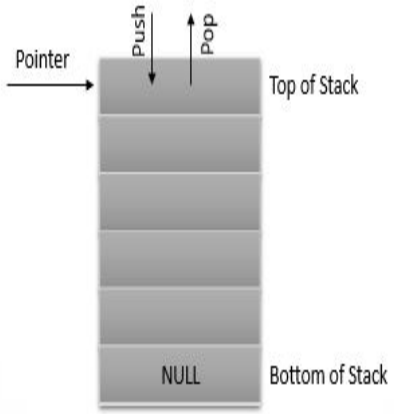
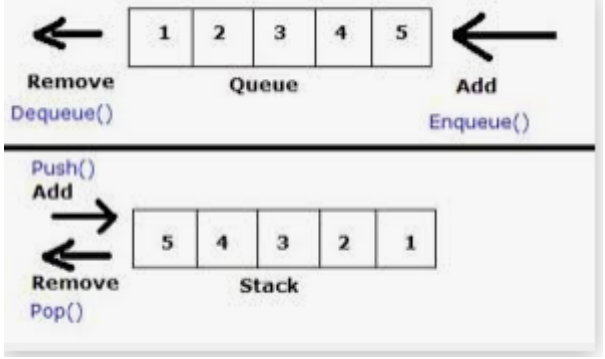
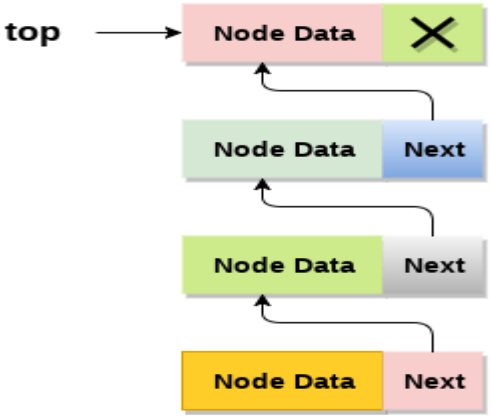
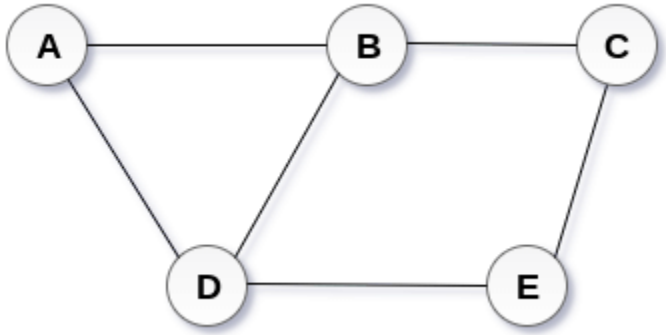


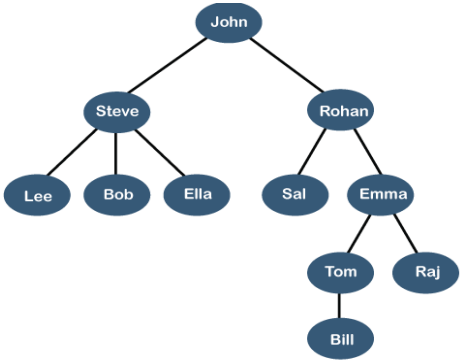




Introduction to Data Structure







Outline

- Data Structures
- Classification of Data Structures
- Operations of Data Structures
- Define Abstract Data Types
- Introduction to Algorithms
- Approaches of Designing an Algorithm
- Control Structures used in Algorithms
- Understanding Basics of Time Complexity
- Introduction to Asymptotic Notation
- Rate of Growth in Algorithm
- Basics of Storage Management.

Basic Terminology

- **Data** : Data means value or set of values. In both the singular and plural form, this term is data. Following arc some examples of data:
  - 34
  - 12/0111965
  - ISBN 81-203-0000-0
- **Entity** : An entity is one that has certain attributes and which may be assigned values. For example, an employee in an organization is an entity. The possible attributes and the corresponding values for an entity in the present example are:

Entity	:	EMPLOYEE			
Attributes	:	NAME	DOB	SEX	DESIGNATION
Values	:	RAVI ANAND	30/12/61	M	DIRECTOR

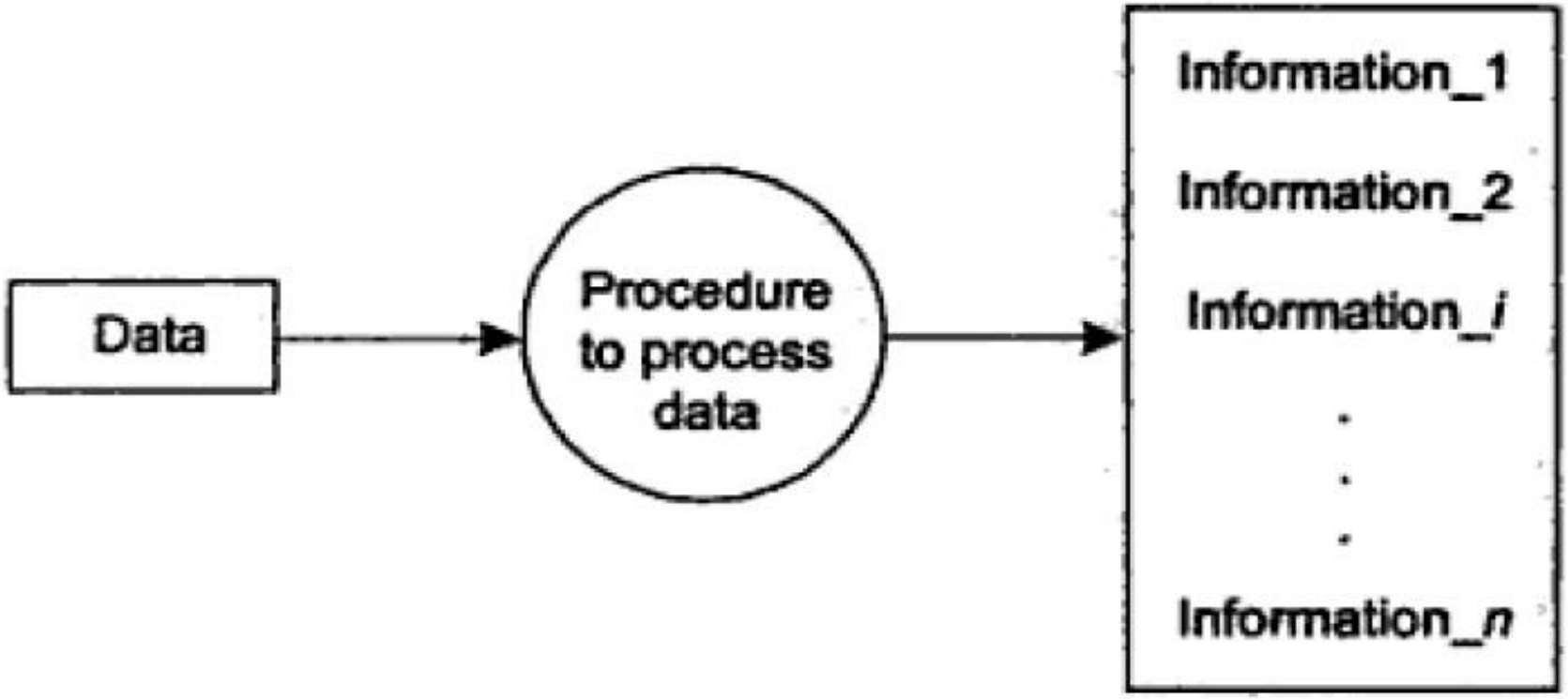
Basic Terminology

- **Information**: The term information is used for data with its attribute(s). In other words, information can be defined as meaningful data or processed data. For example, a set of data is presented during defining the term data, all these data become information if we impose the meaning as mentioned below related to a person:

Data	*Meaning*
34	Age of the person
12/01/1965	Date of birth of the person
ISBN 81-203-0000-0	Book number, recently published by the person
| | |	Number of awards achieved by the person in tally mark
Pascal	Nick name of the person
Æ	Signature of the person
21, 25, 28, 30, 35, 37, 38, 41, 43	Important ages of the person

Difference between data and Information

- From the definition of data and information it is evident that these two are not the same, however. there is a relation between them. Figure shows the interrelation between data and information



BASIC TERMINOLOGY

- This three-step approach to select an appropriate data structure for the problem at hand supports a data-centred view of the design process.
- In the approach, the first concern is the data and the operations that are to be performed on them.
- The second concern is the representation of the data, and
- The final concern is the implementation of that representation.

BASIC TERMINOLOGY

- We know how to write, debug, and run simple programs in C language.
- Our aim has been to design good programs, where a good program is defined as a program that
  - runs correctly
  - is easy to read and understand
  - is easy to debug
  - is easy to modify
- A program should give correct results, but along with that it should also run efficiently.
- A program is said to be efficient when it executes in minimum time and with minimum memory space.
- In order to write efficient programs we need to apply certain data management concepts.

BASIC TERMINOLOGY

- Data structure is a crucial part of data management.
- A data structure is basically a group of data elements that are put together under one name, and which defines a particular way of storing and organizing data in a computer so that it can be used efficiently.

Algorithms are Everywhere

- Search Engines
- GPS navigation
- Self-Driving Cars
- E-commerce
- Banking
- Medical diagnosis
- Robotics
- Algorithmic trading
- and so on …

2020 This Is What Happens In An Internet Minute

What is Data and Data Structure?

- **Data** is the basic fact or entity that is utilized in calculation or manipulation
- There are two different **types of data Numeric** data and **alphanumeric** data
- When a programmer collects such type of data for **processing**, he would require **to store them in computer's main memory**.
- The process of storing data items in computer's main memory is called ***representation/ structure***
- **Data Structure** is a representation of the logical relationship existing between individual elements of data.
- A data structure is a way of organizing all data items that considers not only the elements stored but also their relationship to each other

What is Data and Data Structure?

- We can also define data structure as a mathematical or logical model of a particular organization of data items
- Data Structure mainly specifies the following four things
  - Organization of Data
  - Accessing Methods
  - Degree of Associatively
  - Processing alternatives for information

What is Data and Data Structure?

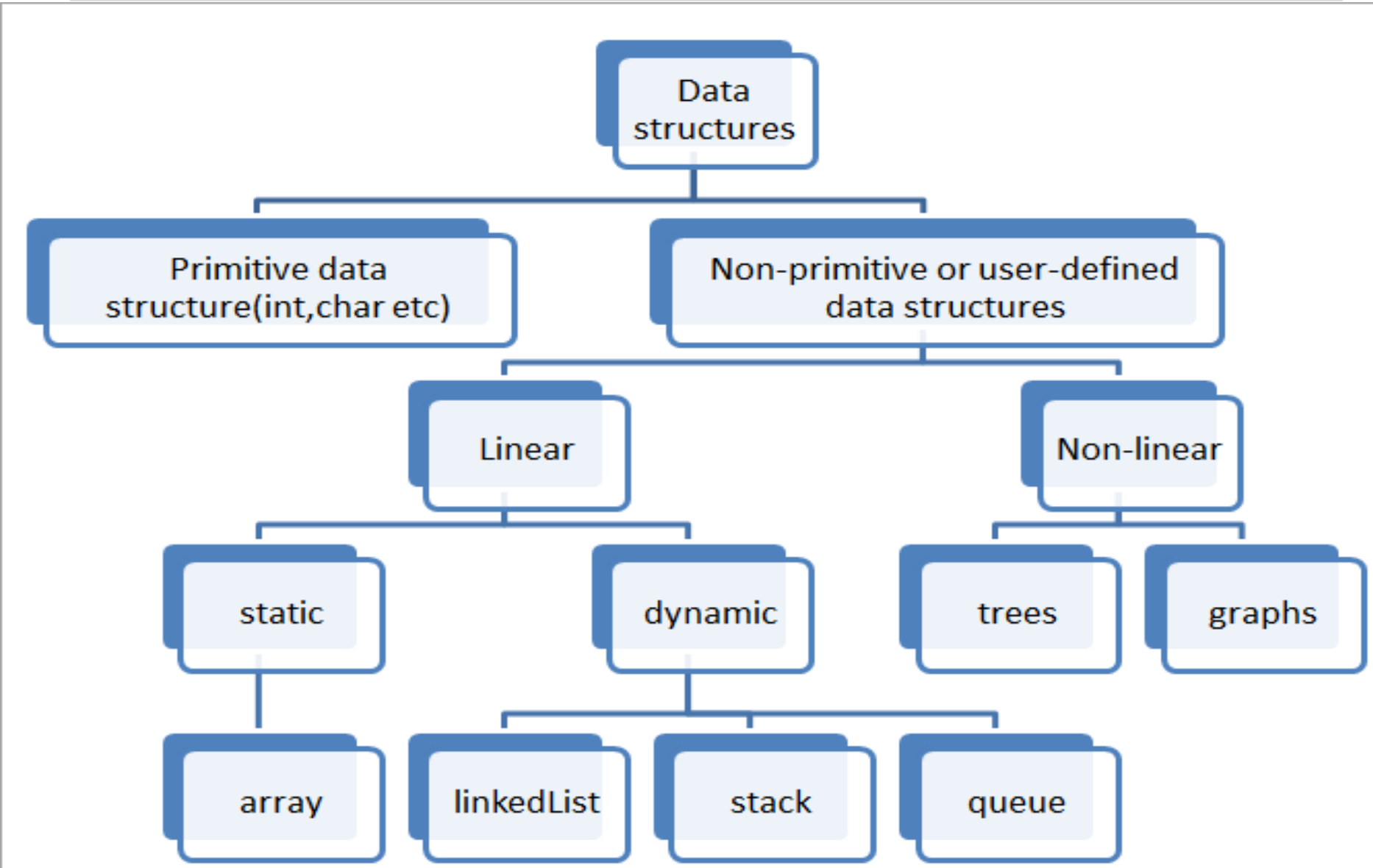
- Data is just the raw material for information, analytics, business intelligence, advertising, etc
- The **representation** of a particular data **structure in the memory** of a computer is called ***Storage Structure***
- The storage structure **representation in auxiliary memory** is called as ***File Structure***

- Computational efficient ways of analyzing, storing, searching, modeling data

What is Data and Data Structure?

- Data structure study covers the following points
  - Amount of memory require to store.
  - Amount of time require to process.
  - Representation of data in memory.
  - Operations performed on that data.

Classification of Data Structure



Primitive/Non-primitive data structures

- **Primitive data structures**
  - Primitive data structures are basic structures and are directly operated upon by machine instructions
  - ***Integers***, ***floats***, ***character*** and ***pointers*** are examples of primitive data structures
- **Non primitive data structure**
  - These are derived from primitive data structures
  - The non-primitive data structures emphasize on structuring of a group of homogeneous or heterogeneous data items
  - Examples of Non-primitive data type are ***Array***, ***List***, and ***File***

Classification data structures

- **Primitive and Non-Primitive Data Structures**
  - *Primitive data*: Primitive data structures are those data structures that can be directly operated upon the machine instructions. These data structures include int, char, float type of data structures.
  - *Non-Primitive data*: Non-Primitive data structures are those data structures that are derived directly from the primitive data structures. Examples of Non-Primitive data structures include class, structure, array, linked lists.

Classification data structures

- **Homogeneous and Heterogeneous Data Structures**
  - *Homogeneous data*: Homogeneous data structures are those data structures that contain only similar type of data e.g. like a data structure containing only integral values or float values. The simplest example of such type of data structures is an Array.
  - *Heterogeneous Data*: Heterogeneous Data Structures are those data structures that contains a variety or dissimilar type of data, for e.g. a data structure that can contain various data of different data types like integer, float and character. The examples of such data structures include structures, class etc.

Classification data structures

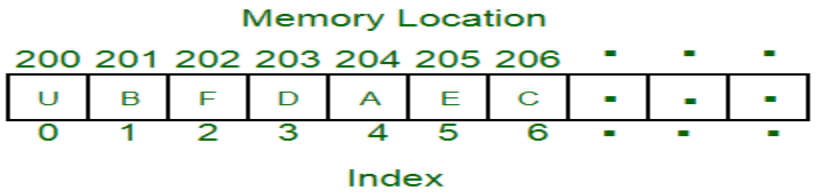
- **Static and Dynamic Data Structures**
  - *Static data*: Static data structures are those data structures to which the memory is assigned at the compile time, means the size of such data structures has to be predefined in the program and their memory can be increased or decreased during the execution. The simplest example of static data structure is an array.
  - *Dynamic data*: Dynamic Data Structures are those data structures to which memory is assigned at the runtime or execution time and hence their size can be increased or decreased dynamically as per requirement of the program. The example of such data structures include class, linked lists etc.

Classification data structures

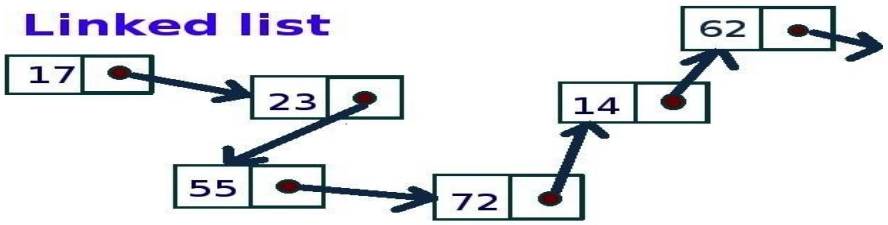
- **Linear and Non-Linear Data Structures**
  - *Linear data*: In linear data structures the data is kept in the sequential manner which means that the previous data has knowledge of next data or we can say that the data is interrelated. The example such data structures include arrays, linked lists etc.
  - *Non-linear data*: Non-linear data structures does not maintain any kind of relationship among the data. The data is stored in non-sequential manner and examples of these types of data structures includes graphs, trees etc.

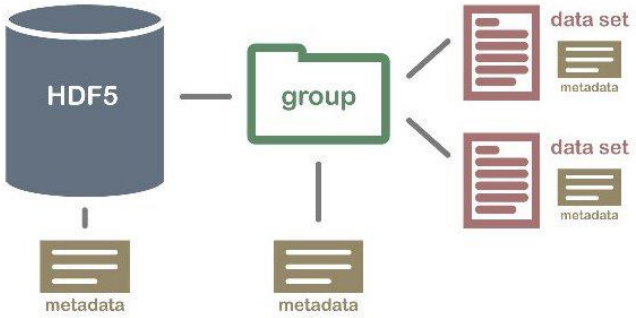
Non-primitive data structures

- **Array:** An array is a fixed-size sequenced collection of elements of the same data type.



- **List:** An ordered set containing variable number of elements is called as Lists.

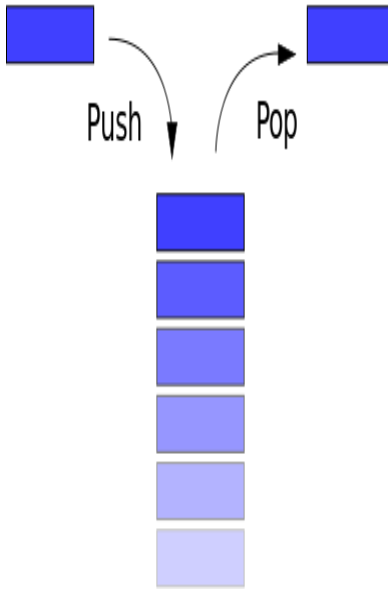


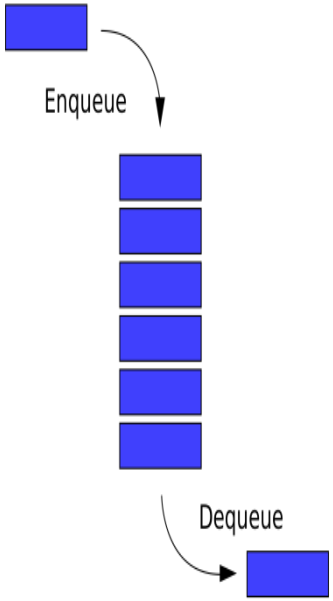


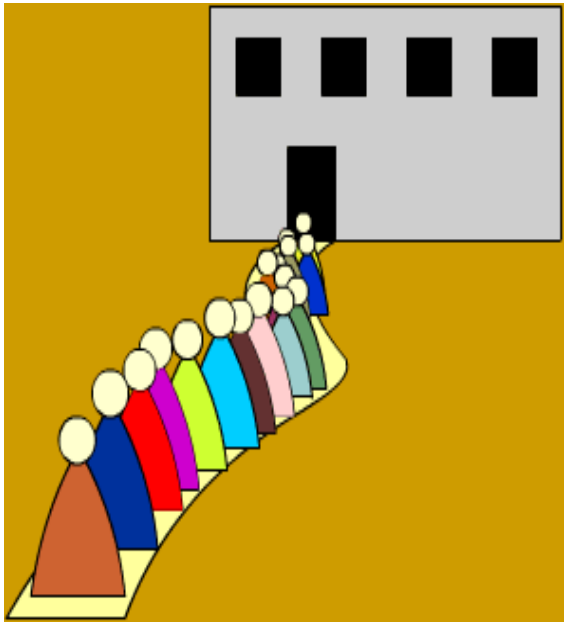
- **File:** A file is a collection of logically related information. It can be viewed as a large list of records consisting of various fields.

Linear / Non-Linear data structure

- **Linear data structures**
  - A data structure is said to be Linear, if its elements are connected in linear fashion by means of logically or in sequence memory locations
  - Examples of Linear Data Structure are ***Stack [LIFO]*** and ***Queue[FIFO]***.

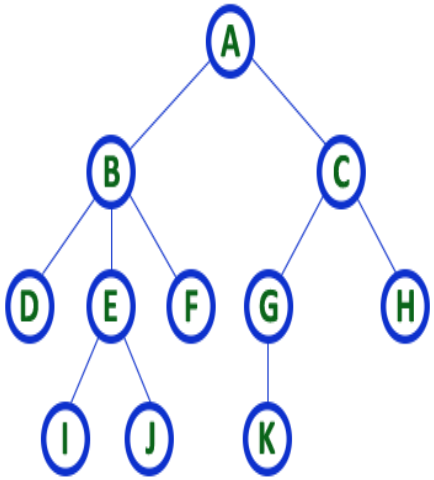


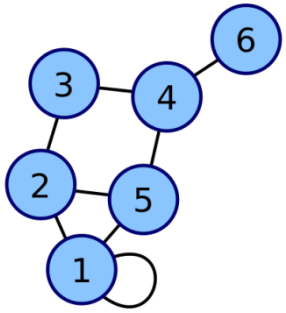


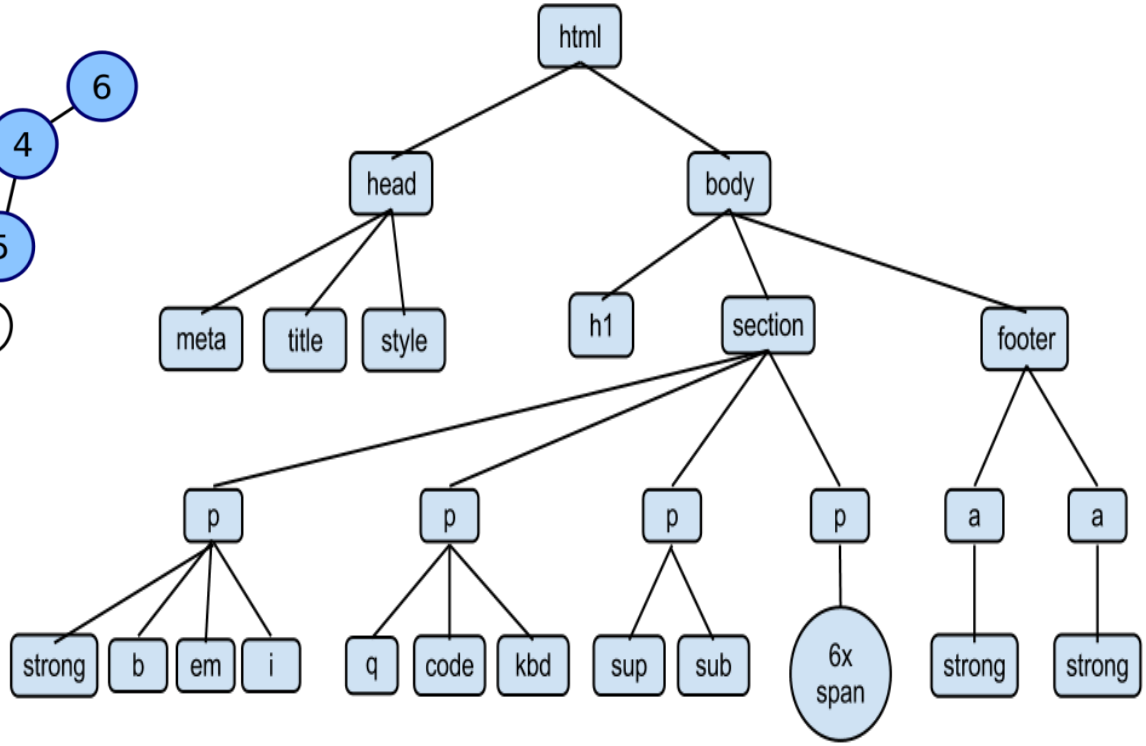


Linear / Non-Linear data structure

- **Nonlinear data structures**
  - Nonlinear data structures are those data structure in which data items are not arranged in a sequence
  - Examples of Non-linear Data Structure are ***Tree*** and ***Graph***







Differences Between Linear/Non-Linear data structures

Linear Data Structure	Non-Linear Data Structure
Every item is related to its previous and next time.	Every item is attached with many other items.
Data is arranged in linear sequence.	Data is not arranged in sequence.
Data items can be traversed in a single run.	Data cannot be traversed in a single run.
Eg. Array, Stacks, linked list, queue.	Eg. tree, graph.
Implementation is easy.	Implementation is difficult.

Operations of Data Structure

- **Create**: it results in reserving memory for program elements
- **Destroy**: it destroys memory space allocated for specified data structure
- **Selection**: it deals with accessing a particular data within a data structure
- **Updating**: it updates or modifies the data in the data structure
- **Searching**: it finds the presence of desired data item in the list of data items
- **Sorting**: it is a process of arranging all data items in a data structure in a particular order. For example, preparing roll number students are sorted on surname
- **Merging**: it is a process of combining the data items of two different sorted list into a single sorted list

Operations of Data Structure

- **Splitting**: it is a process of partitioning single list to multiple list
- **Traversal**: it is a process of visiting each and every node of a list in systematic manner exactly once. For example, to print the names of all the students in a class.
- ***Inserting:*** It is used to add new data items to the given list of data items. For example, to add the details of a new student who has recently joined the course.
- ***Deleting:*** It means to remove (delete) a particular data item from the given collection of data items. For example, to delete the name of a student who has left the course.

ABSTRACT DATA TYPE

- An abstract data type (ADT) is the way we look at a data structure, focusing on what it does and ignoring how it does its job.
- For example, stacks and queues are perfect examples of an ADT.
- We can implement both these ADTs using an array or a linked list.
- Abstract Data Type : An *abstract data type(ADT)* is a data type that is organized in such a way that the specification of the objects and the operations on the objects is separated from the representation of the objects and the implementation of the operations.

ABSTRACT DATA TYPE

- In C, an Abstract Data Type can be a structure considered without regard to its implementation.
- It can be thought of as a "description" of the data in the structure with a list of operations that can be performed on the data within that structure.
- The end user is not concerned about the details of how the methods carry out their tasks.
- They are only aware of the methods that are available to them and are concerned only about calling those methods and getting back the results but not HOW they work.

Algorithm

- What is program :
  - A Set of Instructions
  - Data Structures + Algorithms = Programs
  - Data Structure = A Container stores Data
  - Algorithm = Logic + Control

Algorithm

- An ***algorithm*** is a finite set of instructions that accomplishes a particular task.
  - Criteria

    input: zero or more quantities that are externally supplied
    output: at least one quantity is produced
    definiteness: clear and unambiguous
    finiteness: terminate after a finite number of steps
    effectiveness: instruction is basic enough to be carried out
  - A program does not have to satisfy the **finiteness** criteria.
  - Representation
    - A natural language, like English or Chinese.
    - A graphic, like flowcharts.
    - A computer language, like C. [Sequential search vs. Binary search]

```
algorithm  Get_Circumference
// constants
    PI is 3.14159265
// declarations
    radius = 3
    diameter = 6
    circumf = 18.85
// program

// Get the data
   print("Enter the radius")
   read( radius )

// Compute circumference
   diameter <- radius * 2
   circumf <- diameter * PI

// Output the answer
   print("Diameter is: ", diameter)
   print("Circumference is: ", circumf )
endalgorithm // Get_Cirumference
```

Algorithm

- Algorithms are mainly used to achieve software reuse.
- Once we have an idea or a blueprint of a solution, we can implement it in any high-level language like C, C++, or Java.
- An algorithm is basically a set of instructions that solve a problem.
- It is not uncommon to have multiple algorithms to tackle the same problem, but the choice of a particular algorithm must depend on the time and space complexity of the algorithm.
- Good Algorithm : Run in less time , Consume less memory. But computational resources (time complexity) is usually more important

Design an algorithm to add two numbers and display the result.

```
Step 1 – START
Step 2 – declare three integers a, b & c
Step 3 – define values of a & b
Step 4 – add values of a & b
Step 5 – store output of step 4 to c
Step 6 – print c
Step 7 – STOP
```

Algorithms help the programmers how to code the program

```
Step 1 – START ADD
Step 2 – get values of a & b
Step 3 – c ← a + b
Step 4 – display c
Step 5 – STOP
```

Time and space analysis of algorithms

- Sometimes, there are more than one way to solve a problem.
- We need to learn how to compare the performance different algorithms and choose the best one to solve a particular problem.
- While analyzing an algorithm, we mostly consider time complexity and space complexity
- ***Time complexity*** of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input
- ***Space complexity*** of an algorithm quantifies the amount of space or memory taken by an algorithm to run as a function of the length of the input

Time and space analysis of algorithms

- Time & space complexity depends on lots of things like hardware, operating system, processors, etc.
- However, we don't consider any of these factors while analyzing the algorithm. We will only consider the execution time of an algorithm

What is Space complexity?

- When we design an algorithm to solve a problem, it needs some computer memory to complete its execution.
- For any algorithm, memory is required for the following purposes…
  - To store program instructions.
  - To store constant values.
  - To store variable values.
  - And for few other things like function calls, jumping statements etc,.
- Space complexity of an algorithm can be defined as follows…
  - Total amount of computer memory required by an algorithm to complete its execution is called as space complexity of that algorithm.

What is Space complexity?

- To calculate the space complexity, we must know the memory required to store different datatype values (according to the compiler). 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.

What is Space complexity?

- **Example 1**

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

- In the above piece of code, 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 this 4 bytes of memory is fixed for any input value of 'a'. This space complexity is said to be *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.**

What is 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;
}
```

- In the above piece of code it requires **'n*2'** bytes of memory to store array variable **'a[ ]',** 2 bytes of memory for integer parameter **'n',** 4 bytes of memory for local integer variables **'sum'** and **'i'** (2 bytes each), 2 bytes of memory for **return value**.

What is Space complexity?

- **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.**
- **This type of space complexity is said to be *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**.

What is Time complexity?

- Every algorithm requires some amount of computer time to execute its instruction to perform the task.
- This computer time required is called time complexity.
- The time complexity of an algorithm can be defined as follows..
  - The time complexity of an algorithm is the total amount of time required by an algorithm to complete its execution.

What is Time complexity?

- Generally, the running time of an algorithm depends upon the following…
  - Whether it is running on **Single** processor machine or **Multi** processor machine.
  - Whether it is a **32 bit** machine or **64 bit** machine.
  - **Read** and **Write** speed of the machine.
  - The amount of time required by an algorithm to perform **Arithmetic** operations, **logical** operations, **return** value and **assignment** operations etc.,
  - **Input** data

What is Time complexity?

- **Example 1**

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

- In the above sample code, 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.
- If any program requires a fixed amount of time for all input values then its time complexity is said to be **Constant Time Complexity.**

What is Time 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;}
```

- For the above code, time complexity can be calculated as follows...

What is Time complexity?

	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

What is Time complexity?

- **Cost** is the amount of computer time required for a single operation in each line.
- **Repetition** is the amount of computer time required by each operation for all its repetitions.
- **Total** is the amount of computer time required by each operation to execute.
- So above code requires **'4n+4' Units** of computer time to complete the task.
- 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.
- **Totally it takes '4n+4' units of time to complete its execution and it is *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.

What is Time complexity?

- Imagine a classroom of 100 students in which you gave your pen to one person.
- Now, you want that pen.
- Here are some ways to find the pen and what the O order is.
  - **$O(n^2)$:** You go and ask the first person of the class, if he has the pen. Also, you ask this person about other 99 people in the classroom if they have that pen and so on, This is what we call $O(n^2)$.
  - **O(n):** Going and asking each student individually is O(N).
  - **O(log n):** Now I divide the class into two groups, then ask: "Is it on the left side, or the right side of the classroom?" Then I take that group and divide it into two and ask again, and so on. Repeat the process till you are left with one student who has your pen. This is what you mean by O(log n).

What is Time complexity?

- I might need to do the $O(n^2)$ search if only one student knows on which student the pen is hidden. I'd use the O(n) if one student had the pen and only they knew it. I'd use the O(log n) search if all the students knew, but would only tell me if I guessed the right side.

What is Time complexity?

- Determine the frequency counts of each statements of the following codes

1. x = x + y
2. for i = 1 to n do
   x = x + y
3. for i = 1 to n do
   for j = 1 to n do
   x = x + y

What is Time complexity?

1. The frequency count is 1
2. The frequency count is n
3. The frequency count is n^2

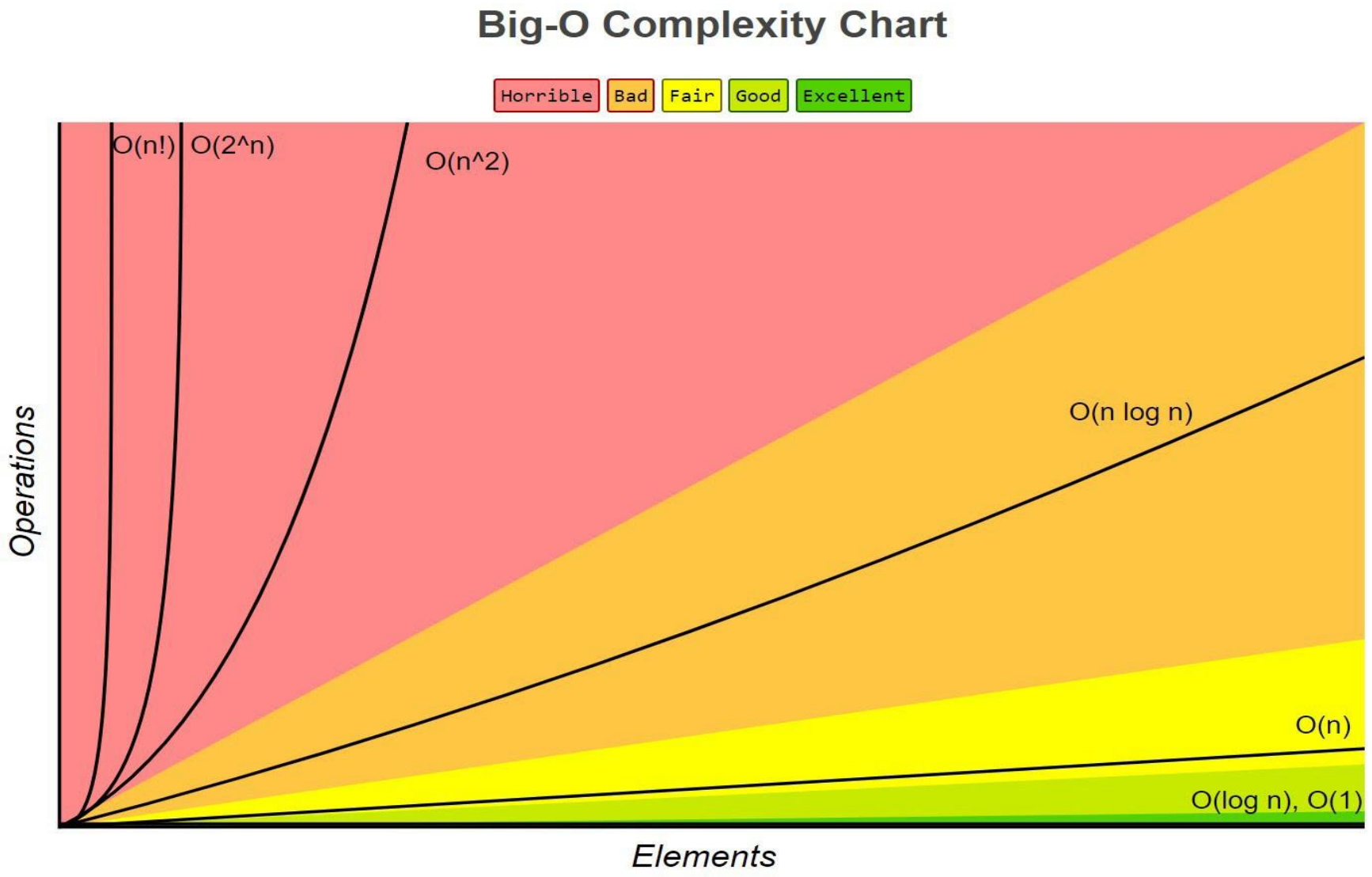
Time Complexity Table?

Big O Notation	Time Complexity Details
O(1)	Constant Time Complexity O(1) occurs when the program doesn't contain any loops, recursive functions or call to any other functions. The run time, in this case, won't change no matter what the input value is.
O(n)	Linear Time Complexity O(n) occurs when the run time of the code increases at an order of magnitude proportional to n. Here n is the size of the input.
O(log n)	Logarithmic Time Complexity O(log n) occurs when at each subsequent step in the algorithm, the time is decreased at a magnitude inversely proportional to n. This generally happens in Binary Search Algorithm.

Time Complexity Table?

Big O Notation	Time Complexity Details
O(n log n)	Linearithmic Time Complexity. One example of an algorithm that runs with this time complexity is Quick Sort, Heap Sort, Merge Sort
$O(n^2)$	Quadratic Time Complexity
O(2^n)	Exponential Time Complexity
O(n!)	Factorial Time Complexity

Time Complexity Table?



What is Asymptotic Notation?

- Whenever we want to perform analysis of an algorithm, we need to calculate the complexity of that algorithm.
- But when we calculate the complexity of an algorithm it does not provide the exact amount of resource required.
- We use that general form (Notation) for analysis process.
- Asymptotic notation of an algorithm is a mathematical representation of its complexity.
- **Note -** In asymptotic notation, when we want to represent the complexity of an algorithm, we use only the most significant terms in the complexity of that algorithm and ignore least significant terms in the complexity of that algorithm (Here complexity can be Space Complexity or Time Complexity).

What is Asymptotic Notation?

- For example, consider the following time complexities of two algorithms…
  - **Algorithm 1 : $5n^2 + 2n + 1$**
  - **Algorithm 2 : $10n^2 + 8n + 3$**
- When we analyze an algorithm, we consider the time complexity for larger values of input data (i.e. **'n'** value). In above two time complexities, for larger value of **'n'** the term **'2n + 1'** in algorithm 1 has least significance than the term **'$5n^2$'**, and the term **'8n + 3'** in algorithm 2 has least significance than the term **'$10n^2$'**.
- For larger value of **'n'** the value of most significant terms (**$5n^2$** and **$10n^2$**) is very larger than the value of least significant terms (**2n + 1** and **8n + 3**).

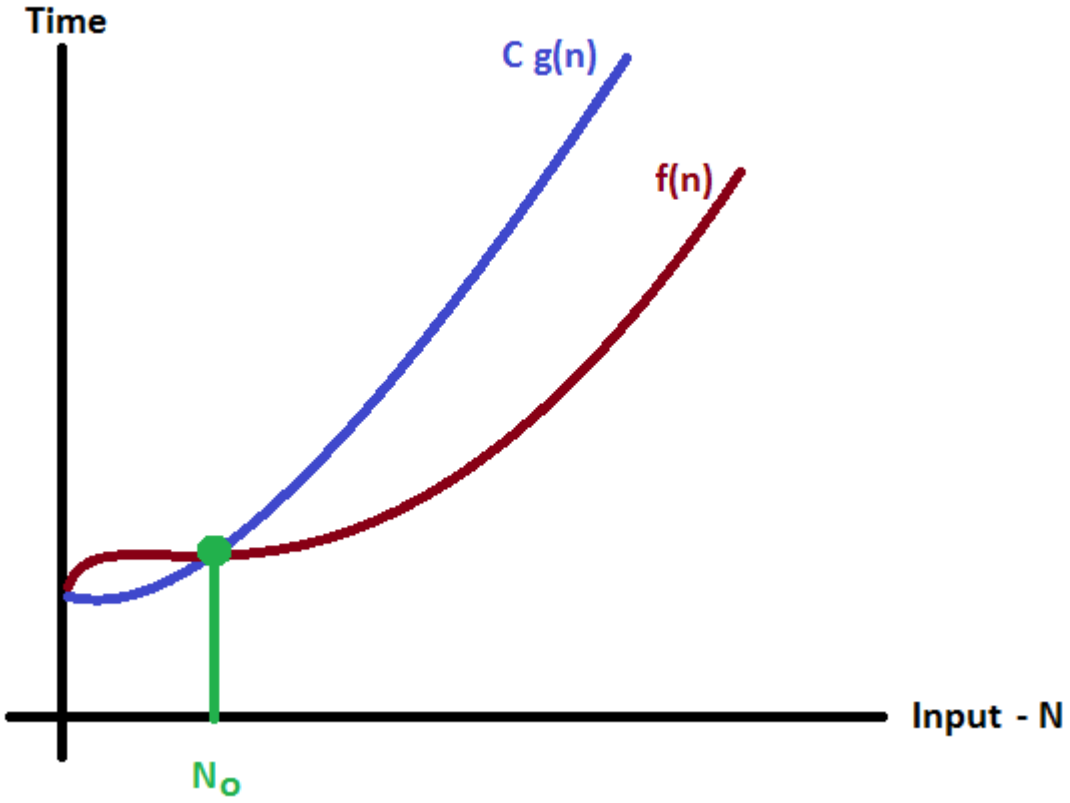
What is Asymptotic Notation?

- So for larger value of **'n'** we ignore the least significant terms to represent overall time required by an algorithm. In asymptotic notation, we use only the most significant terms to represent the time complexity of an algorithm.
- Majorly, we use THREE types of Asymptotic Notations and those are as follows…
  - **Big - Oh (O)**
  - **Big - Omega (Ω)**
  - **Big - Theta (Θ)**

Big - Oh Notation (O)

- Big - Oh notation is used to define the **upper bound** of an algorithm in terms of Time Complexity.
- That means Big - Oh notation always indicates the maximum time required by an algorithm for all input values.
- That means Big - Oh notation describes the worst case of an algorithm time complexity.
- Big - Oh Notation can be defined as follows…
  - Consider function f(n) as time complexity of an algorithm and g(n) is the most significant term. If f(n) <= C * g(n) for all n >= n_0, C > 0 and n_0 >= 1. Then we can represent f(n) as O(g(n)).
  - f(n) = O(g(n))
- Consider the following graph drawn for the values of f(n) and C g(n) for input (n) value on X-Axis and time required is on Y-Axis

Big - Oh Notation (O)



- In above graph after a particular input value n_0, always C g(n) is greater than f(n) which indicates the algorithm's upper bound.

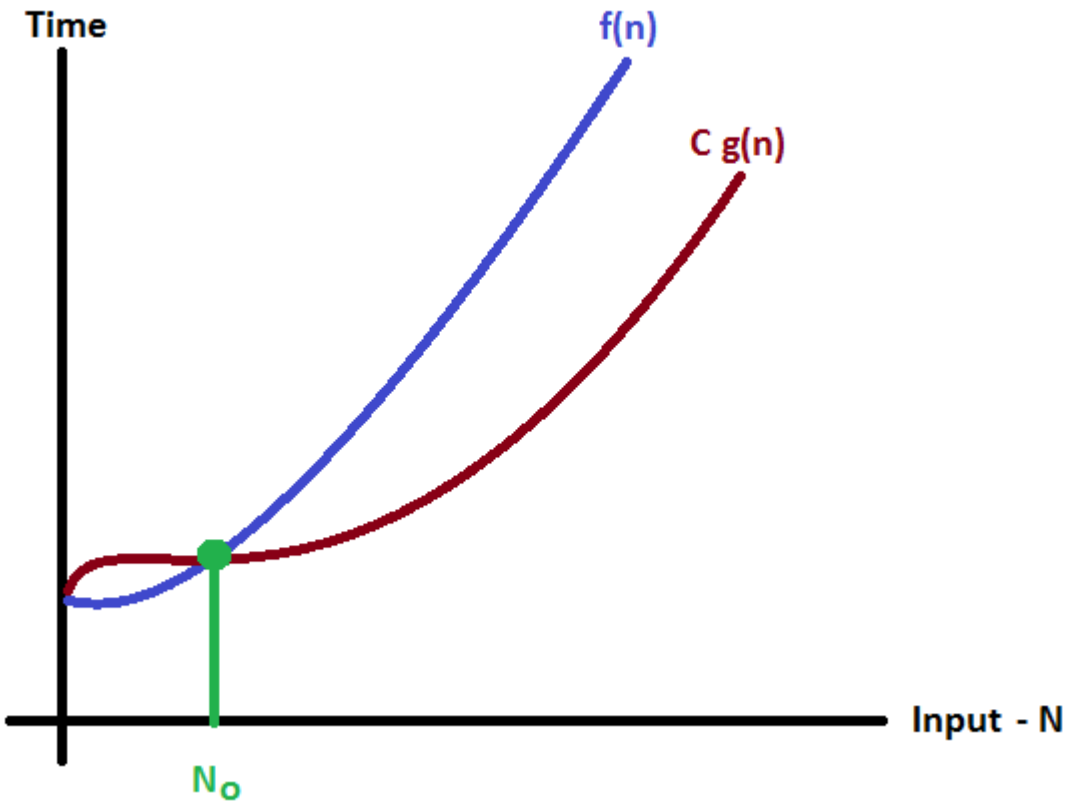
Big - Oh Notation (O)

- Example
  Consider the following f(n) and g(n)…
  f(n) = 3n + 2
  g(n) = n
- If we want to represent **f(n)** as **O(g(n))** then it must satisfy **f(n) <= C g(n)** for all values of **C > 0** and **n_0>= 1**
- f(n) <= C g(n)
  ⇒3n + 2 <= C n
- Above condition is always TRUE for all values of **C = 4** and **n >= 2**.
- By using Big - Oh notation we can represent the time complexity as follows...
  3n + 2 = O(n)

Big - Omege Notation (Ω)

- Big - Omega notation is used to define the **lower bound** of an algorithm in terms of Time Complexity.
- That means Big-Omega notation always indicates the minimum time required by an algorithm for all input values.
- That means Big-Omega notation describes the best case of an algorithm time complexity.
- Big - Omega Notation can be defined as follows…
  - Consider function f(n) as time complexity of an algorithm and g(n) is the most significant term.
- If f(n) >= C g(n) for all n >= n_0, C > 0 and n_0 >= 1. Then we can represent f(n) as Ω(g(n)).
  f(n) = Ω(g(n))
- Consider the following graph drawn for the values of f(n) and C g(n) for input (n) value on X-Axis and time required is on Y-Axis

Big - Omege Notation (Ω)



- In above graph after a particular input value n_0, always C g(n) is less than f(n) which indicates the algorithm's lower bound..

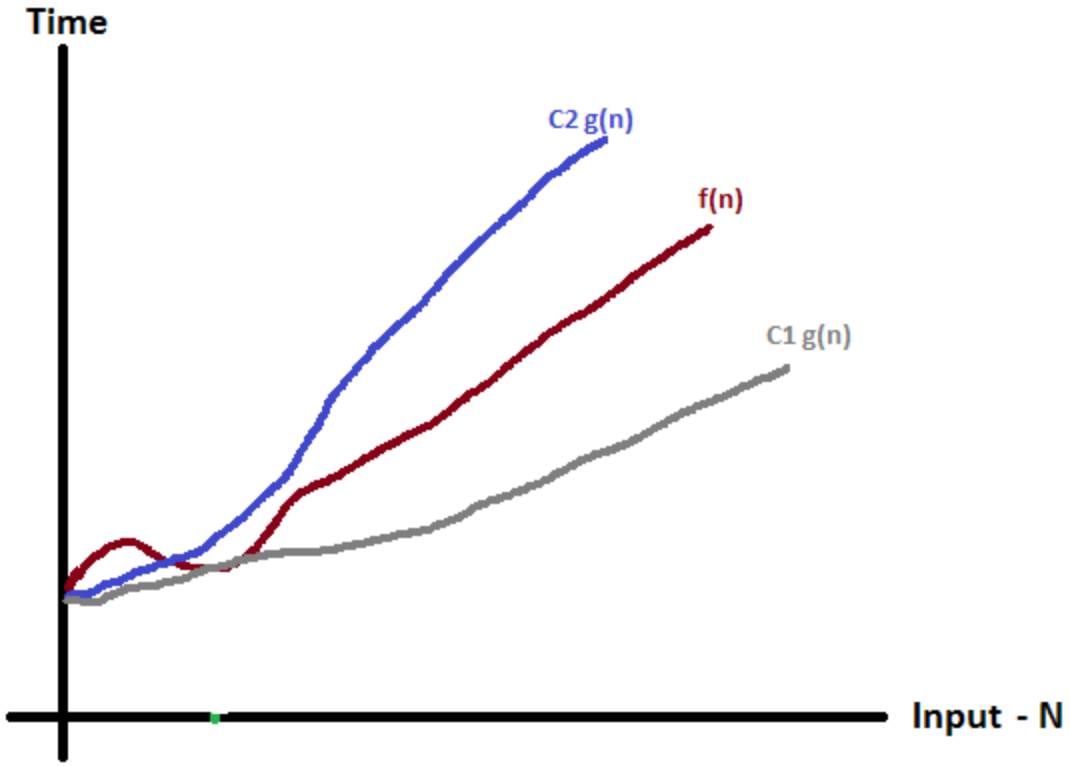
Big - Omega Notation (Ω)

- Example
  Consider the following f(n) and g(n)…
  f(n) = 3n + 2
  g(n) = n
- If we want to represent **f(n)** as **Ω(g(n))** then it must satisfy **f(n) >= C g(n)** for all values of **C > 0** and **n_0>= 1**
  f(n) >= C g(n)
  ⇒3n + 2 >= C 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 as follows...
  3n + 2 = Ω(n)

Big - Theta Notation (Θ)

- Big - Theta notation is used to define the **average bound** of an algorithm in terms of Time Complexity.
- That means Big - Theta notation always indicates the average time required by an algorithm for all input values.
- That means Big - Theta notation describes the average case of an algorithm time complexity.
- Big - Theta Notation can be defined as follows…
  - Consider function f(n) as time complexity of an algorithm and g(n) is the most significant term. If C_1 g(n) <= f(n) <= C_2 g(n) for all n >= n_0, C_1 > 0, C_2 > 0 and n_0 >= 1. Then we can represent f(n) as Θ(g(n)).
  - f(n) = Θ(g(n))
- Consider the following graph drawn for the values of f(n) and C g(n) for input (n) value on X-Axis and time required is on Y-Axis

Big - Theta Notation (Θ)



- In above graph after a particular input value n_0, always C_1 g(n) is less than f(n) and C_2 g(n) is greater than f(n) which indicates the algorithm's average bound.

Big - Theta Notation (Θ)

- Example
  Consider the following f(n) and g(n)…
  f(n) = 3n + 2
  g(n) = n
- If we want to represent **f(n)** as **Θ(g(n))** then it must satisfy **C_1 g(n) <= f(n) <= C_2 g(n)** for all values of **C_1 > 0, C_2 > 0** and **n_0>= 1**
  C_1 g(n) <= f(n) <= C_2 g(n)
  ⇒C_1 n <= 3n + 2 <= C_2 n
  Above condition is always TRUE for all values of **C_1 = 1, C_2 = 4** and **n >= 2**.
- By using Big - Theta notation we can represent the time complexity as follows...
  3n + 2 = Θ(n)