Data Structures & Algorithms


Types of Data Structures

Data Structures

WsCube Tech
System For Satisfaction

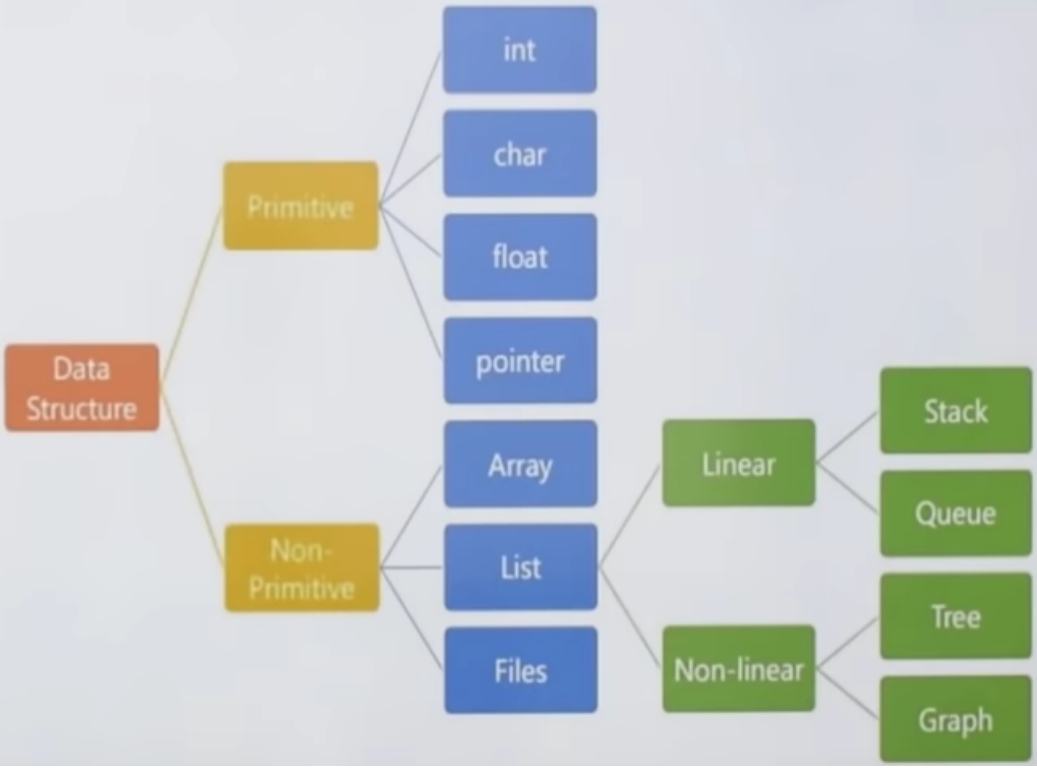
Data Structure is a way of collecting and organizing data in such a way that we can perform operations on these data in an effective way.

Types Of Data Structures

- Primitive data structures
- Non-primitive data structure

Classification of Data Structures

WsCube Tech
System For Satisfaction



Primitive and Non-primitive DS

Primitive Data Structures: Primitive Data Structures are the basic data structures that directly operate upon the machine instructions.

Non-primitive Data Structures

Non-primitive data structures are more complicated data structures and are derived from primitive data structures.

They emphasize on grouping same or different data items with relationship between each data item.

linear and non-linear data structure

Linear DS:

- every item is related to its previous and next time.
- data is arranged in linear sequence.
- data items can be traversed in a single run.
- implementation is easy

Non-linear DS:

- every item is attached with many other items.
- data is not arranged in sequence.
- data cannot be traversed in a single run.
- implementation is difficult.

Static and Dynamic DS

WsCube Tech
System For Satisfaction

Static	Static data structures are those whose sizes and structures associated memory locations are fixed, at compile time. Example: **Array**
Dynamic	Dynamic structures are those which expands or shrinks depending upon the program need and its execution. Also, their associated memory locations changes. Example: **Linked List created using pointers**

Homogeneous and Non-Homogeneous DS

Homogeneous	In homogeneous data structures, all the elements are of same type. Example: **Array**
Non-Homogeneous	In Non-Homogeneous data structure, the elements may or may not be of the same type. Example: **Structures**

What is a Good Algorithm?

WsCube Tech
System For Satisfaction

Efficiency

- Running Time
- Space used

Asymptotic Analysis

- To Simply the analysis of running time by getting the rid of details which may be affected by specific implementation hardware.
- Capturing the essence: how the running time of an algorithm increases with the size of the input in the limit.

Two Basic Rules:-

- Drop all lower order terms
- Drop all constants

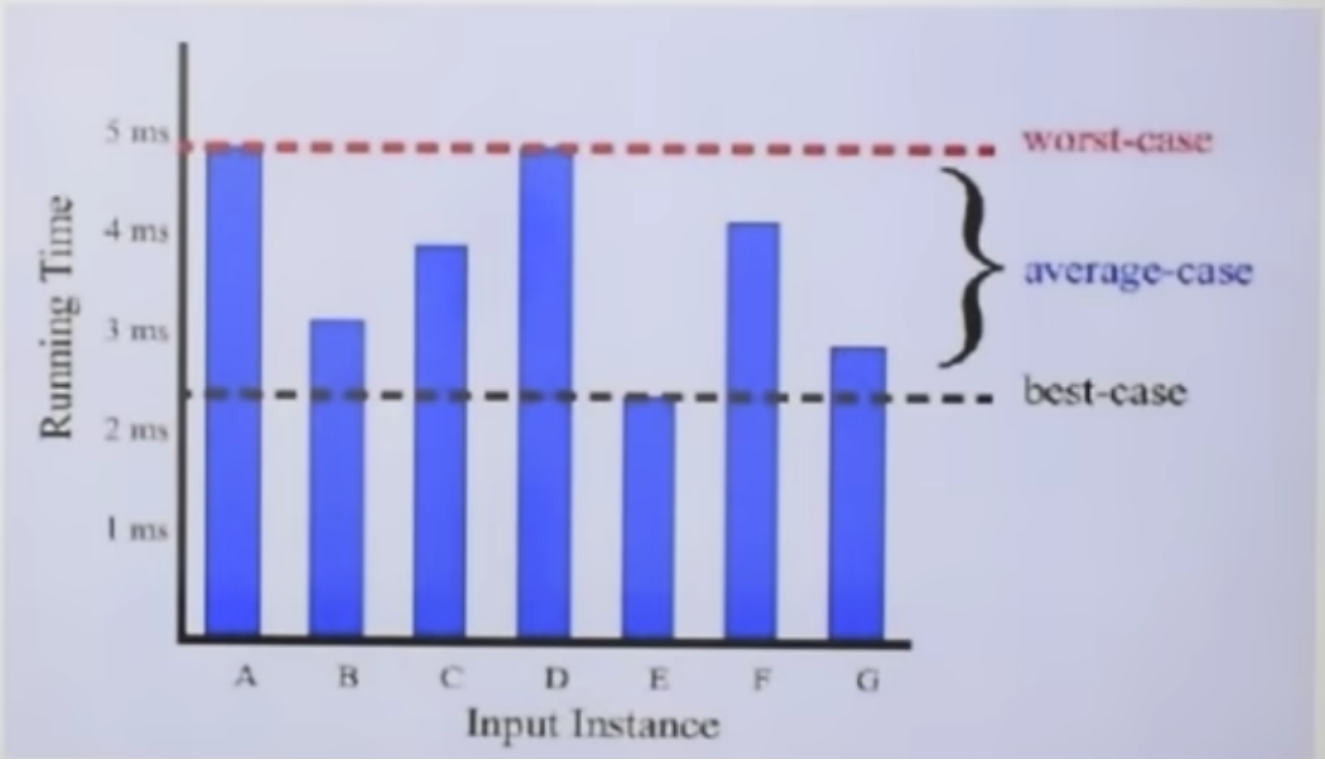
Asymptotic Notations

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

There are three types of Asymptotic Notations...

- Big - Oh (O)
- Big - Omega (Ω)
- Big - Theta (Θ)

Best/Worst/Average Case



Big - Oh Notation (O) (worst)

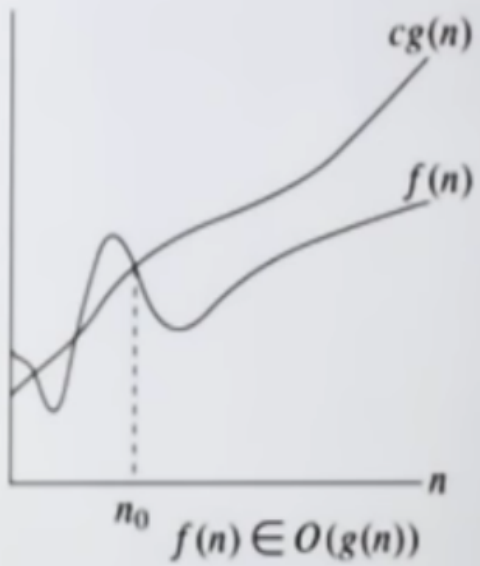
WSCUBE TECH
System For Satisfaction

- Big - Oh notation is used to define the upper bound of an algorithm in terms of Time Complexity. It provides us with an **asymptotic upper bound.**
- 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 (O)

WsCube Tech
System For Satisfaction

- f(n) is your algorithm runtime, and g(n) is an arbitrary time complexity you are trying to relate to your algorithm. f(n) is O(g(n)), if for some real constants c (c > 0) and n0, f(n) <= c g(n) for every input size n (n > n0).



Big - Omega Notation (Ω)

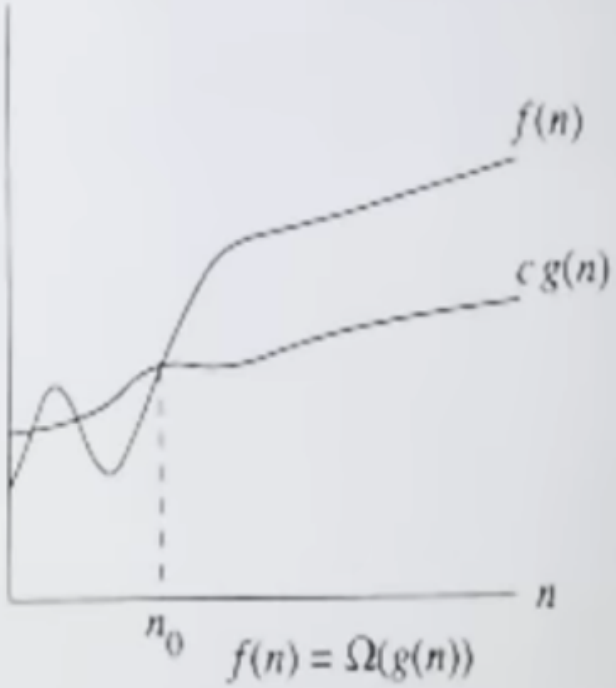
WsCube Tech
System For Satisfaction

- 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. It provides us with an **asymptotic lower bound**.

Big - Omege Notation (Ω)

WSCUBE TECH
System For Satisfaction

- f(n) is Ω(g(n)), if for some real constants c (c > 0) and n0 (n0 > 0), f(n) is >= c g(n) for every input size n (n > n0).



Big - Theta Notation (Θ)

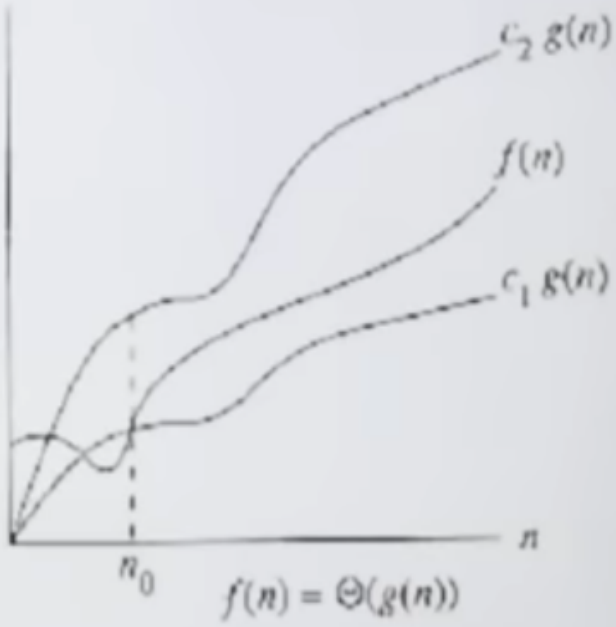
WSCUBE TECH
System For Satisfaction

- Big - Theta notation is used to define the average bound of an algorithm in terms of Time Complexity and denote the ***asymptotically tight bound***
- 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 (Θ)

WSCUBE TECH
System For Satisfaction

- function f(n) as time complexity of an algorithm and g(n) is the most significant term. If C1 g(n) <= f(n) <= C2 g(n) for all n >= n0, C1 > 0, C2 > 0 and n0 >= 1. Then we can represent f(n) as Θ(g(n)).



Summary

WsCube Tech
System For Satisfaction

- Analogy with real numbers
  - $f(n) = O(g(n)) \cong f \leq g$
  - $f(n) = \Omega(g(n)) \cong f \geq g$
  - $f(n) = \Theta(g(n)) \cong f = g$
  - Little 'o' $f(n) = o(g(n)) \cong f < g$
  - Little 'ω' $f(n) = \omega(g(n)) \cong f > g$

Common Asymptotic Notations

NOTATION	NAME
O(1)	constant
O(log n)	logarithmic
O(n)	linear
$O(n^2)$	quadratic
$O(n^c)$	polynomial or algebraic
$O(c^n)$; where c>1	exponential

Row Major Order

In this method elements of an array are arranged sequentially row by row. Thus elements of first row occupies first set of memory locations reserved for the array, elements of second row occupies the next set of memory and so on.

The Location of element A[i, j] can be obtained by evaluating expression:

LOC (A [i, j]) = Base_Address + W [M (i) + (j)]

Here,

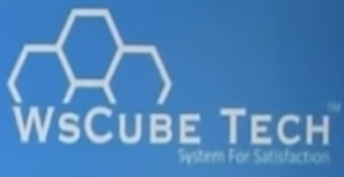
Base Address is the address of first element in the array.

W is the word size. It means number of bytes occupied by each element.

N is number of rows in array.

M is number of columns in array.

Example of Row Major Order



11 Arr[0][0] 1000	**12** Arr[0][1] 1002	**13** Arr[0][2] 1004	**14** Arr[0][3] 1006
21 Arr[1][0] 1008	**22** Arr[1][1] 1010	**23** Arr[1][2] 1012	**24** Arr[1][3] 1014
31 Arr[2][0] 1016	**32** Arr[2][1] 1018	**33** Arr[2][2] 1020	**34** Arr[2][3] 1022

Suppose we want to calculate the address of element Arr[1, 2].

It can be calculated as follow:

Here,

Base Address = 1000, W= 2, M=4, N=3, i=1, j=2

LOC (A [i, j])=Base Address + W [M (i) + (j)]

LOC (A[1, 2])=1000 + 2 *[4*(1) + 2]

= 1000 + 2 * [4 + 2]

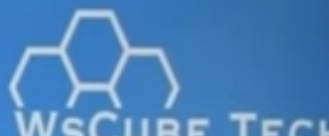
= 1000 + 2 * 6

= 1000 + 12

= **1012**

11	**12**	**13**	**14**	**21**	**22**	**23**	**24**	**31**	**32**	**33**	**34**

Column Major Order



In this method elements of an array are arranged sequentially column by column. Thus elements of first column occupies first set of memory locations reserved for the array, elements of second column occupies the next set of memory and so on.

The Location of element A[i, j] can be obtained by evaluating expression:

LOC (A [i, j]) = Base_Address + W [N (j) + (i)]

Base_Address is the address of first element in the array.
W is the word size. It means number of bytes occupied by each element.
N is number of rows in array.
M is number of columns in array.

Example of Column Major Order

11 Arr[0][0] 1000	**12** Arr[0][1] 1006	**13** Arr[0][2] 1012	**14** Arr[0][3] 1018
21 Arr[1][0] 1002	**22** Arr[1][1] 1008	**23** Arr[1][2] 1014	**24** Arr[1][3] 1020
31 Arr[2][0] 1004	**32** Arr[2][1] 1010	**33** Arr[2][2] 1016	**34** Arr[2][3] 1022

Suppose we want to calculate the address of element Arr[1, 2].

It can be calculated as follow:

Here,

Base Address = 1000, W= 2, M=4, N=3, i=1, j=2

LOC (A [i, j])=Base Address + W [N (j) + (i)]

LOC (A[1, 2])=1000 + 2 *[3*(2) + 1]

= 1000 + 2 * [6 + 1]

= 1000 + 2 * 7

= 1000 + 14

= **1014**

11	**21**	**31**	**12**	**22**	**32**	**13**	**23**	**33**	**14**	**24**	**34**

Some Conclusion

WsCube Tech
System For Satisfaction

Data Structure	Time Complexity							
	Average				Worst			
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)

Static

Dynamic

Arrays

Pros

Easy to create, Easy to use

Direct indexing: O(1)

Sequential access: O(N)

Cons

Searching: O(N), and O(LogN) if sorted

Inserting and deleting: O(N) because of shifting items.

Linked List

Pros

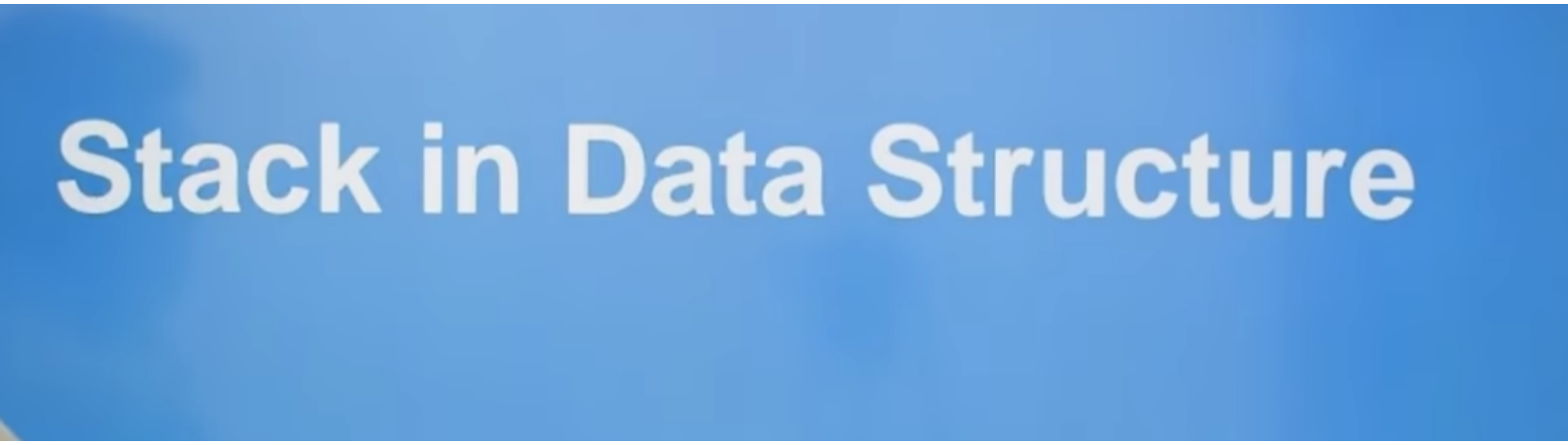
Inserting and deleting: O(1)

Sequential Access: O(N)

Cons

No Direct Access; Only Sequential Access

Searching: O(N)


Stack in Data Structure

What is Stack?

WsCube Tech
System For Satisfaction

- A stack is a linear data structure in which data items are inserted and deleted at one end only, rather than in the middle, called as top of stack.
- Stack follows LIFO(Last In First Out) or FILO(First In Last Out).
- Stack is a restricted data structure.

Operations performed on Stack

Push: Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.

Pop: Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.

Top: Returns top element of stack.

isEmpty: Returns true if stack is empty, else false.