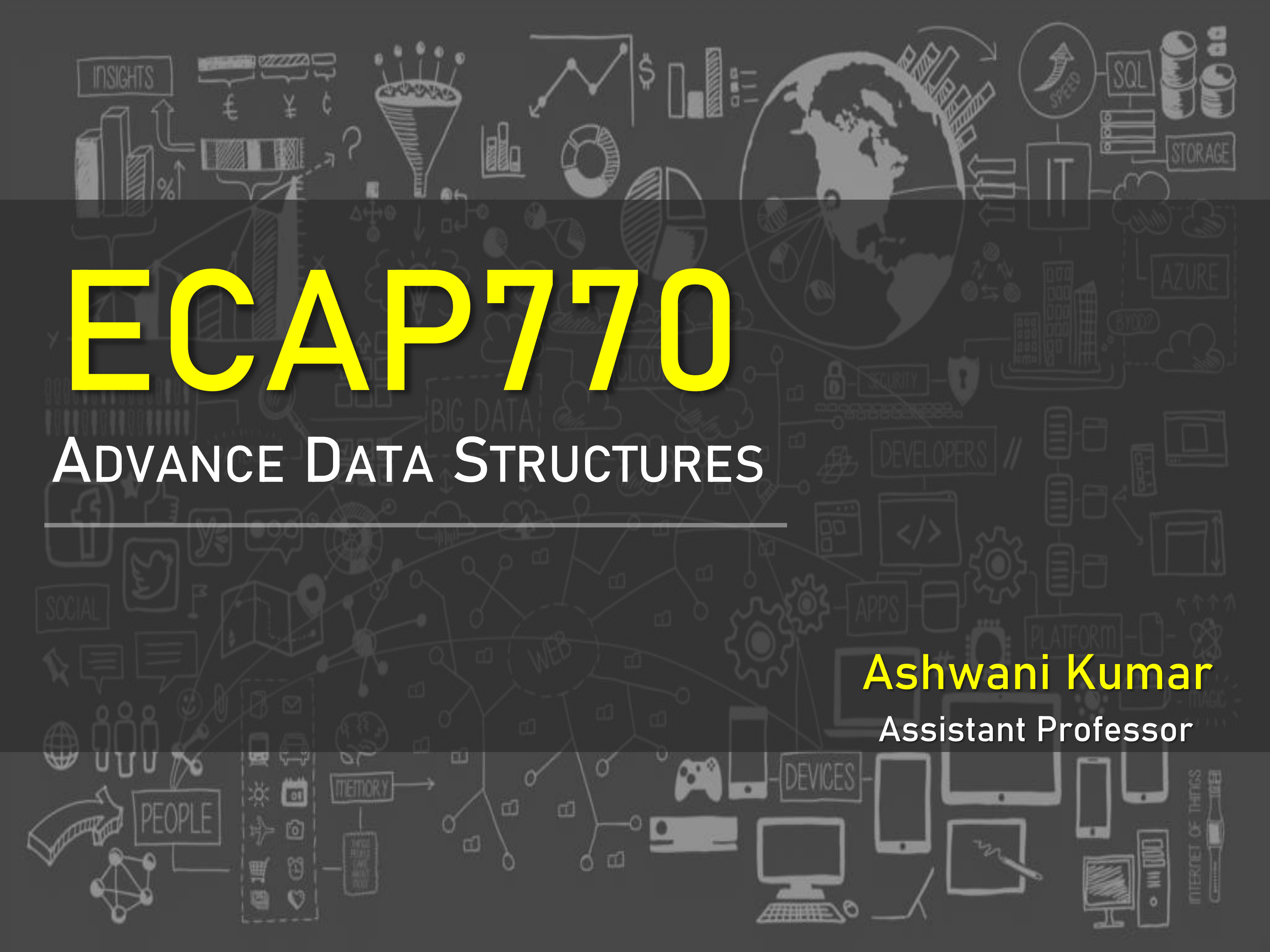


ECAP770

Advance Data Structures

Ashwani Kumar

Assistant Professor

Learning Outcomes

After this lecture, you will be able to

- know about asymptotic notations
- know about abstract data type

Asymptotic notations

- To measure the efficiency of an algorithm asymptotic analysis is used.
- The efficiency of an algorithm depends on the amount of time, storage and other resources required to execute the algorithm.
- Performance of algorithm is change with different type of inputs.

Asymptotic notations

- The study of change in performance of the algorithm with the change in the order of the input size is defined as asymptotic analysis.
- Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.

Types of asymptotic notations

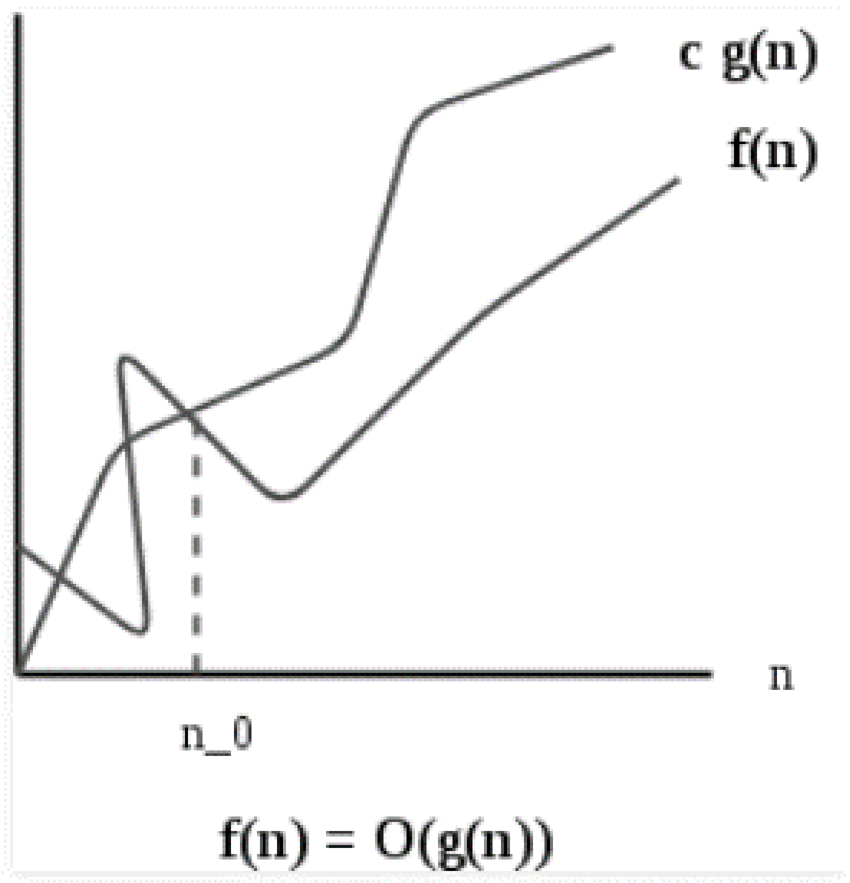
There are three major asymptotic notations

- Big-O notation
- Omega notation
- Theta notation

Big-O Notation (O-notation)

- Big-O notation represents the upper bound of the running time of an algorithm. It gives the worst case complexity of an algorithm.
- O(n) is useful when we only have an upper bound on the time complexity of an algorithm.
- It is widely used to analyse an algorithm as we are always interested in the worst-case scenario.

Big-O Notation (O-notation)

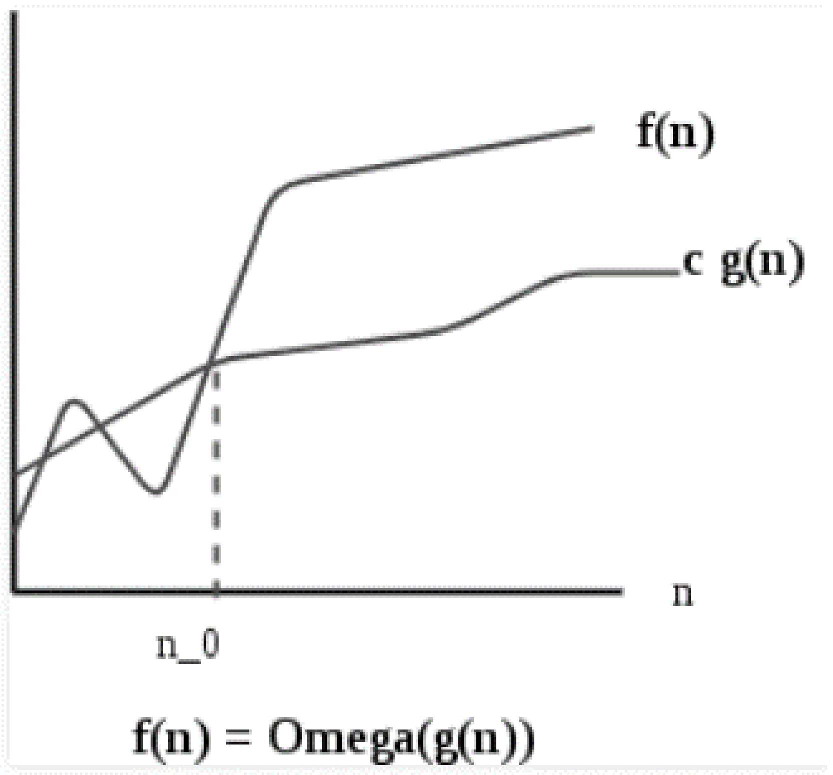


O(g(n)) = { f(n): there exist positive constants c and n0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n0$ }

Omega Notation (Ω-notation)

- Omega notation represents the lower bound of the running time of an algorithm. It provides the best case complexity of an algorithm.
- Omega Notation can be useful when we have lower bound on time complexity of an algorithm.
- Omega notation is the least used notation among all three.

Omega Notation (Ω-notation)



Ω (g(n)) = {f(n): there exist positive constants c and

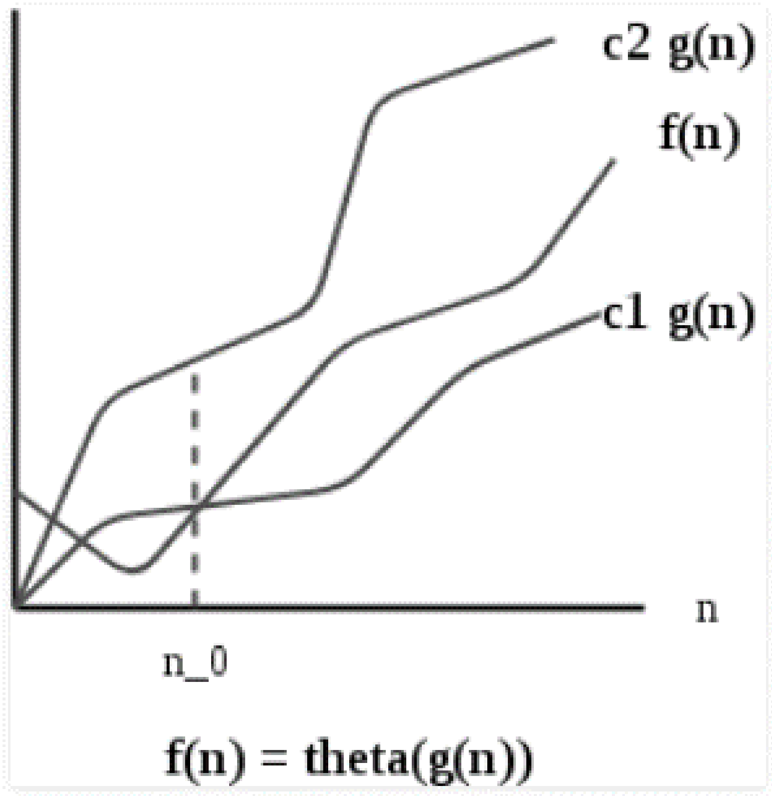
n0 such that 0 <= c*g(n) <= f(n) for

all n >= n0}.

Theta Notation (Θ-notation)

- Theta notation encloses the function from above and below. It represents the upper and the lower bound of the running time of an algorithm, it is used for analysing the average-case complexity of an algorithm.

Theta Notation (Θ-notation)



Θ(g(n)) = {f(n): there exist positive constants c1, c2 and n0 such that 0 <= c1*g(n) <= f(n) <= c2*g(n) for all n >= n0}

Properties of Asymptotic Notations

- General Properties
  - If f(n) is O(g(n)) then a*f(n) is also O(g(n)) ; where a is a constant.
- Transitive Properties
  - If f(n) is O(g(n)) and g(n) is O(h(n)) then f(n) = O(h(n))

Properties of Asymptotic Notations

- Reflexive Properties
  - If f(n) is given then f(n) is O(f(n))
- Symmetric Properties
  - If f(n) is Θ(g(n)) then g(n) is Θ(f(n))
- Transpose Symmetric Properties
  - If f(n) is O(g(n)) then g(n) is Ω (f(n))

Common Asymptotic Notations

constant	–	$O(1)$
logarithmic	–	$O(\log n)$
linear	–	$O(n)$
n log n	–	$O(n \log n)$
quadratic	–	$O(n^2)$
cubic	–	$O(n^3)$
polynomial	–	$n^{O(1)}$
exponential	–	$2^{O(n)}$

Abstract Data Type

- Data abstraction can be defined as separation of the logical properties of the organization of programs' data from its implementation.
- One can also create user defined data types, decide the range of values as well as operations to be performed on them

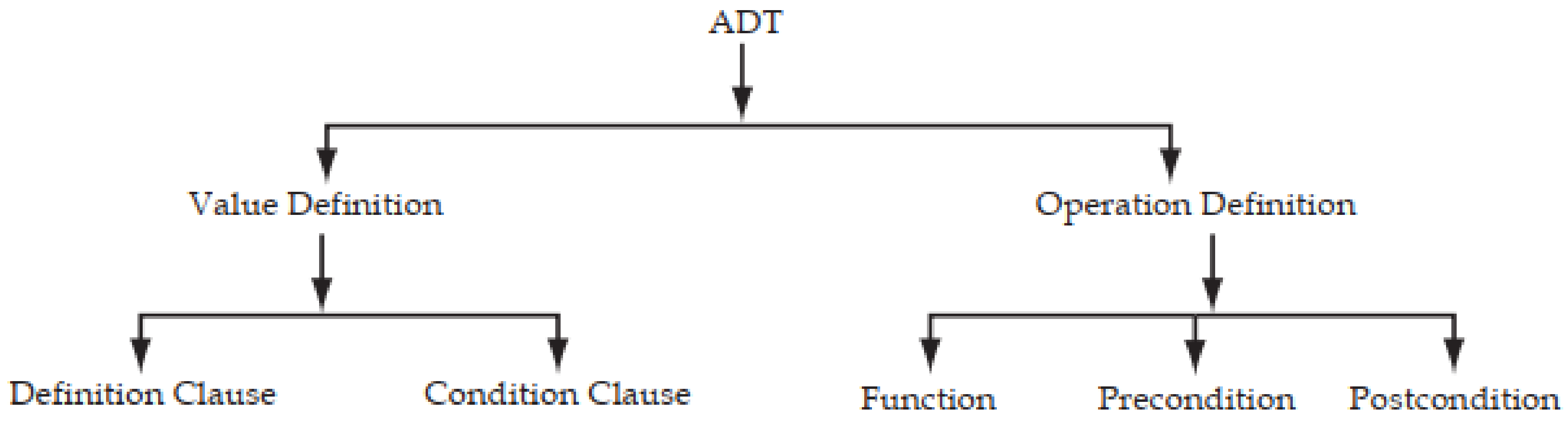
Abstract Data Type

- It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. It is called “abstract” because it gives an implementation-independent view.
- The process of providing only the essentials and hiding the details is known as abstraction.

Abstract Data Type

An ADT has two parts:

- Value definition
- Operation definition



Value definition

Value definition is again divided into two parts:

- Definition clause
- Condition clause
- Definition clause states the contents of the data type and condition clause defines any condition that applies to the data type.
- Definition clause is mandatory while condition clause is optional.

Operation definition

In operation definition, there are three parts:

Function

Precondition

Postcondition

Operation definition

- The function clause defines the role of the operation.
- Precondition specifies any condition that may apply as a pre-requisite for the operation definition.
- Postcondition specifies what the operation does. It specifies the state after the operation is performed.

List ADT

- The data is generally stored in key sequence in a list which has a head structure consisting of count, pointers and address of compare function needed to compare the data in the list.
- Operations on list
- get() – Return an element from the list at any given position.

List ADT

- insert() – Insert an element at any position of the list.
- remove() – Remove the first occurrence of any element from a non-empty list.
- removeAt() – Remove the element at a specified location from a non-empty list.

Stack ADT

- In Stack ADT Implementation instead of data being stored in each node, the pointer to data is stored.
- The program allocates memory for the data and address is passed to the stack ADT.
- push() – Insert an element at one end of the stack called top.

Stack ADT

- pop() – Remove and return the element at the top of the stack, if it is not empty.
- size() – Return the number of elements in the stack.

Queue ADT

- The queue abstract data type (ADT) follows the basic design of the stack abstract data type.
- Each node contains a void pointer to the data and the link pointer to the next element in the queue. The program’s responsibility is to allocate memory for storing the data.

Queue ADT

- enqueue() – Insert an element at the end of the queue.
- dequeue() – Remove and return the first element of the queue, if the queue is not empty.
- size() – Return the number of elements in the queue.

That's all for now…