

Data Structures

IT 205

Dr. Manish Khare



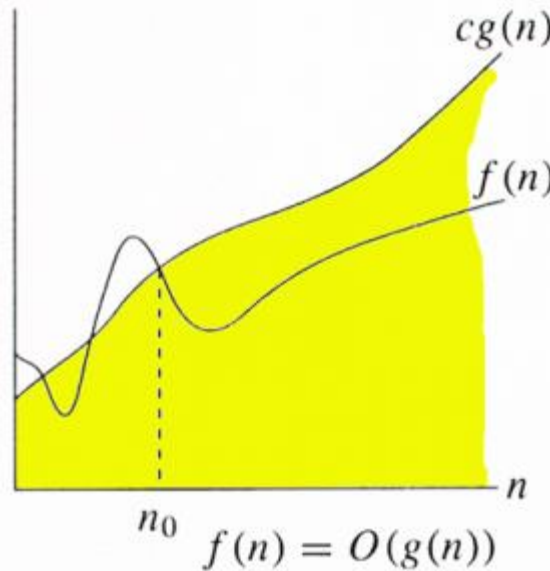
Lecture - 3



Examples on Asymptotic Notations (O , Ω , Θ)

Big Oh Notation, O

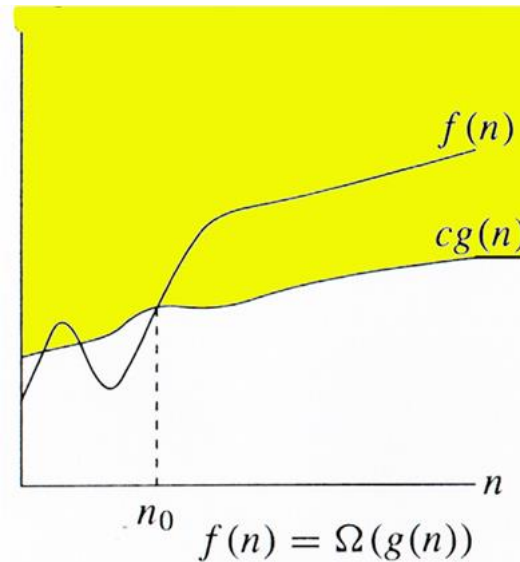
- The notation $O(n)$ is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.



$O(g(n)) = \{f(n): \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}$

Omega Notation, Ω

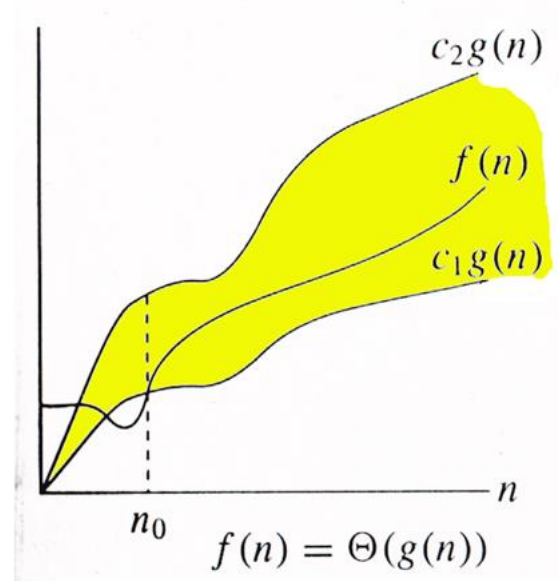
- The notation $\Omega(n)$ is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.



$\Omega(g(n)) = \{f(n): \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\}.$

Theta Notation, Θ

- The notation $\Theta(n)$ is the formal way to express both the lower bound and the upper bound of an algorithm's running time. It is represented as follows.



$\Theta(g(n)) = \{f(n): \text{there exist positive constants } c_1, c_2 \text{ and } n_0 \text{ such that } 0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \text{ for all } n \geq n_0\}$




Data structures basic concepts

Data Definition & Data Objects

- Data Definition defines a particular data with the following characteristics.
 - **Atomic** – Definition should define a single concept.
 - **Traceable** – Definition should be able to be mapped to some data element.
 - **Accurate** – Definition should be unambiguous.
 - **Clear and Concise** – Definition should be understandable.

- Data Object represents an object having a data.

Data Type

- 
- Data type is a way to classify various types of data such as integer, string, etc. which determines the values that can be used with the corresponding type of data, the type of operations that can be performed on the corresponding type of data. There are two data types –
 - Built-in Data Type
 - Derived Data Type

Build-in Data Type

➤ Those data types for which a language has built-in support are known as Built-in Data types. For example, most of the languages provide the following built-in data types.

- Integers
- Boolean (true, false)
- Floating (Decimal numbers)
- Character and Strings

Derived Data Type

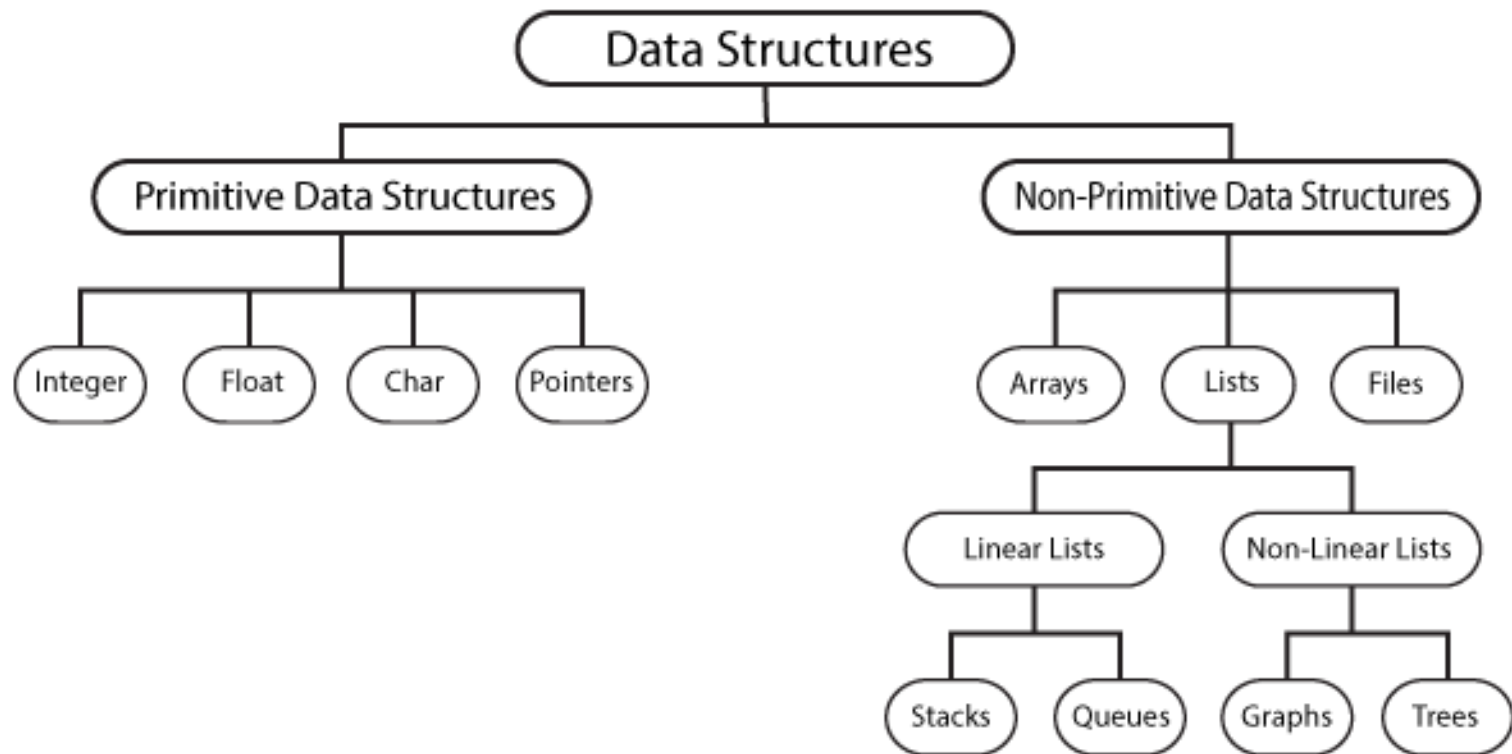
- Those data types which are implementation independent as they can be implemented in one or the other way are known as derived data types. These data types are normally built by the combination of primary or built-in data types and associated operations on them.
- For example –
 - List
 - Array
 - Stack
 - Queue

Basic Operations

➤ The data in the data structures are processed by certain operations. The particular data structure chosen largely depends on the frequency of the operation that needs to be performed on the data structure.

- Traversing
- Searching
- Insertion
- Deletion
- Sorting
- Merging

Data Structures




Types Of Data Structure



Array Data Structure

Arrays

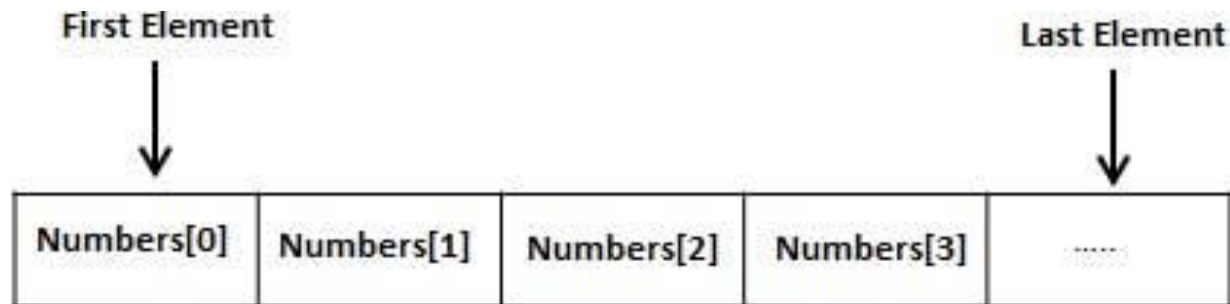


➤ Array is a container which can hold a fix number of items and these items should be of the same type. Most of the data structures make use of arrays to implement their algorithms. Following are the important terms to understand the concept of Array.

- **Element** – Each item stored in an array is called an element.
- **Index** – Each location of an element in an array has a numerical index, which is used to identify the element.

Arrays

- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.
- Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.



Arrays Definition

- A set of pairs $\langle \text{index}, \text{value} \rangle$ where for each value of index there is a value from the set item. Index is a finite ordered set of one or more dimensions.
- For example, $\{0, 1, \dots, n-1\}$ for one-dimension, $\{(0,0), (0,1), (0,2), (1,0), (1,1), (1,2), (2,0), (2,1), (2,2)\}$ for two-dimensions, etc.

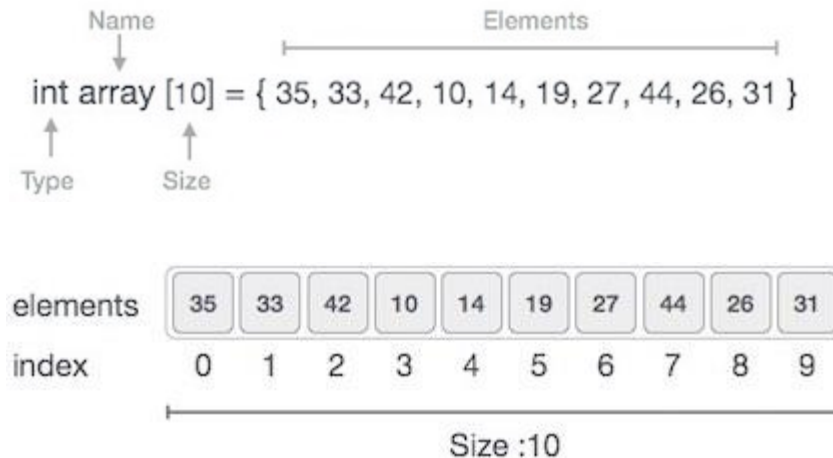
Array Representation

➤ Functions:

- for all $A \in \text{Array}$, $i \in \text{index}$, $x \in \text{item}$, j , $\text{size} \in \text{integer}$
- **Array Create(j, list):** **return** an array of j dimensions where list is a j -tuple whose i th element is the size of the i th dimension. Items are undefined.
- **Item Retrieve (A, i):** **if** ($i \in \text{index}$) **return** the item associated with index value i in Array A **else return** error
- **Array Store(A, i, x):** **if** ($i \in \text{index}$)
return an array that is identical to array A except the new pair $\langle i, x \rangle$ has been inserted **else return** error

Arrays Representation

- Arrays can be declared in various ways in different languages. For illustration, let's take C array declaration.



- As per the above illustration, following are the important points to be considered.
- Index starts with 0.
 - Array length is 10 which means it can store 10 elements.
 - Each element can be accessed via its index. For example, we can fetch an element at index 6 as 9.

Arrays Representation

- A two-dimensional array can be considered as a table which will have x number of rows and y number of columns. A two-dimensional array **a**, which contains three rows and four columns can be shown as follows –

	Column 0	Column 1	Column 2	Column 3
Row 0	<code>a[0][0]</code>	<code>a[0][1]</code>	<code>a[0][2]</code>	<code>a[0][3]</code>
Row 1	<code>a[1][0]</code>	<code>a[1][1]</code>	<code>a[1][2]</code>	<code>a[1][3]</code>
Row 2	<code>a[2][0]</code>	<code>a[2][1]</code>	<code>a[2][2]</code>	<code>a[2][3]</code>

- Thus, every element in the array **a** is identified by an element name of the form `a[i][j]`, where 'a' is the name of the array, and 'i' and 'j' are the subscripts that uniquely identify each element in 'a'.

Arrays – Basic Operations

➤ Following are the basic operations supported by an array.

- **Traverse** – print all the array elements one by one.
- **Insertion** – Adds an element at the given index.
- **Deletion** – Deletes an element at the given index.
- **Search** – Searches an element using the given index or by the value.
- **Update** – Updates an element at the given index.